Multi-layer Distributed Coding Solutions for Large-scale Distributed Computing

by

Arash Yazdanialahabadi

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

Master of Science

in

Communications

Department of Electrical and Computer Engineering

University of Alberta

© Arash Yazdanialahabadi, 2020

Abstract

In distributed computing models, where some helper nodes assist the master in a large-scale computation, a big challenge is when these helpers straggle. The straggling of even a single helper node can significantly increase the processing time. Therefore, coded distributed computing has been proposed as a solution in many recent studies. Unfortunately, in some scenarios, the decoding complexity at the master is so significant that it undermines the benefit of distributing the computation. To allow for distributed computing in these cases, we propose a multi-layer coding strategy that allows some helpers to assist with the decoding. In this thesis, we focus on two scenarios, (1) when an extra layer of helpers are introduced to help with the master's decoding, and (2) fitting a distributed coding scheme when the computation naturally requires multiple layers with shuffling (for example FFT). In both scenarios, we propose a fully-coded structure that tolerates straggling in each step. Moreover, using low-complexity codes such as Raptor codes to further reduce the master's decoding load is studied.

Preface

The contributions and results of this thesis have been presented in two journal submissions. The results and algorithms of Chapter 3 have been submitted to IEEE Transactions on Communications under the title "A Distributed Low-complexity Coding Solution for Large-scale Distributed FFT". Chapter 4 also includes the results of the paper "Distributed Decoding for Coded Distributed Computing" submitted to the Journal of Parallel and Distributed Computing.

To my parents

For their love and never ending support.

Acknowledgements

First, I would like to express my appreciation and gratitude to my supervisor, Dr. Masoud Ardakani. His continuous guidance, support, and mentorship have been exceptional. I am grateful that I was able to work under his supervision and learn so many valuable lessons. Second, I would like to thank the committee members, Dr. Yindi Jing, Dr. Hai Jiang, and Dr. Gregory Kish, for their constructive comments and dedicating their time to this thesis.

Also, I would like to express my gratitude and gratefulness to my family, whom I have been blessed by their love and endless support in every step of my life. Finally, I cannot find a better way to thank my friends, who sparked joy in my life throughout my time in Edmonton.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Distributed Computing . . . 2
1.3 Unreliable Distributed Computing . . . 3
1.4 Coded Distributed Computing . . . 4
1.5 Thesis overview . . . 5

2 Background **7**
2.1 Distributed Computing Systems . . . 7
2.1.1 System Model . . . 8
2.1.2 Challenges . . . 10
2.2 Forward Error Correction Codes . . . 12
2.2.1 Mathematical Background . . . 12
2.2.2 MDS codes . . . 13
2.2.3 Fountain codes . . . 15
2.3 Coding in distributed computing . . . 16
2.3.1 Related works . . . 18
2.3.2 Decoding Complexity . . . 20
2.3.3 Proposed Multi-layer Distributed Coding Solutions Overview . . . 20

3 Multi-layer Coded Distributed FFT **22**
3.1 Introduction . . . 22
3.2 Background and motivation . . . 25
3.2.1 FFT computation . . . 25
3.2.2 Coded distributed FFT . . . 26
3.2.3 Remaining challenges and overview of our contributions 28
3.3 Uncoded distributed FFT solution . . . 29
3.4 Coded solution . . . 31
3.4.1 Toy example . . . 32
3.4.2 Coding algorithm . . . 36
3.4.3 Low decoding complexity codes . . . 40
3.5 Performance analysis and numerical results . . . 42
3.5.1 Failure probability . . . 42
3.5.2 Complexity and cost comparison . . . 43
3.6 Conclusion . . . 48

4 Distributed Decoding **49**
4.1 Introduction . . . 49
4.2 System model . . . 50
4.3 A fast and reliable distributed decoding solution . . . 53
4.3.1 Main idea . . . 53
4.3.2 Proposed distributed decoding solution . . . 54

4.3.3 Greedy decoding helper allocation 57
4.4 Numerical results 59
4.5 Conclusion and future work 63

5 Conclusions and future work 65
5.1 Summary of contributions and results 65
5.2 Future research directions 66

References 68

List of Tables

3.1 The master's load is presented for $N = 2^{32}$, $\varepsilon_c = 0.10$, and a range of K. The numerical column is normalized to the first row, i.e., Self-FFT. 44

List of Figures

1.1 Cloud computing schematic 2

2.1 MapReduce structure with K processors and multiple Shuffle stages. The helpers might perform different computations in each layer. In Map, the master assigns the data and tasks to the helpers and they start their computational task. They might shuffle their output data with each other, and start a new processing. In Reduce, the helpers perform final tasks and send the results to the master for the final processing (if needed). In some scenarios, the helpers only perform one layer of computation. 9
2.2 Distributed matrix multiplication model with K helpers . . . 10
2.3 Coded distributed matrix multiplication model with 4 helpers 17

3.1 The butterfly structure of FFT. Vertices may multiply a weight to their source dots and each dot operates the sum of the input vertices. 27
3.2 Distributed FFT example for $N = 16$, $M = 4$ completed in two layers with four helper nodes in each layer. 30
3.3 The coding scheme example of a two-layer distributed FFT. Generating the coded symbols for redundant nodes is not feasible unless the required symbols for encoding exist in the same place. 33
3.4 The complete proposed coded distributed FFT scheme for (N, M, K, L) = (16,4,4,5). The master performs encoding in Map stage. K helpers participate in M–FFT and the successful nodes send their results to Computation Layer 2. There, helpers decode to recover $\mathbf{b}_4$ symbols, multiply twiddle factors, and encode the symbols they need for the second FFT. In this systematic example, only E_{25} does the encoding. The master finally decodes and retrieves $\mathbf{y}$. 35
3.5 System block diagram 41
3.6 Normalized cost of the proposed algorithm for different values of ψ_t. The minimum point follows a trend: it occurs on higher overheads for higher ψ_t values. 45
3.7 Comparison between the proposed algorithm with two computation layers and Raptor coding and single computation layer with MDS coding. 46
3.8 The proposed algorithm's cost for a number of code lengths for $N = 2^{32}$. 47

4.1 The coded matrix-vector multiplication distributed computing scheme based on MapReduce with no Shuffle stage 51
4.2 The coded matrix-vector multiplication distributed computing scheme with perfect decoding helpers 53

4.3 The two-layer coding structure to add reliability for both processing and decoding layer 55
4.4 The behavior of $\mathbb{E}[\tau_x]$ and its components 60
4.5 A comparison between our proposed multi-layer method and single-layer method when a different number of helpers from the network are utilized 61
4.6 The behavior of different distributed computing allocations based on $|\mathcal{S}_d|$ 62
4.7 The effect of different sizes of computations on the optimum $|\mathcal{S}_d|$ 63

List of Acronyms

CDCS	coded distributed computing systems
DCS	distributed computing system
DFT	discrete Fourier transform
FEC	forward error correction
FFT	fast Fourier transform
GDHA	greedy decoding helper allocation
IoT	internet of things
LT	Luby transform
MDS	maximum distance separable
RS	Reed-Solomon

Chapter 1

Introduction

1.1 Motivation

In recent years, with new technologies such as machine learning and data processing growing, the size of data is increasing every day. Companies collect data from their customers more than ever; the researchers have access to an extensive amount of information; graphic files and datasets are getting extensively larger. This increasing trend in the size of data has created several challenges, such as data storage and processing.

With the development of computing resources and the rising popularity of data analysis and machine learning methods, massive datasets are put into processing for various applications [1]. Furthermore, high-quality image processing and big data analysis require a large amount of computation [2]. Therefore, large-scale computing has gained attention in recent years. Even the most straightforward computations can become a challenge when the size of data increases exponentially. Hence, processing massive data has introduced many new challenges.

The problem with large-scale computing is that the personal processors cannot handle the computation due to memory limitations as well as CPU/GPU speed, and capacity limitations. This problem has forced users to move their computation to the cloud. Cloud service providers have contributed to this trend by providing a variety of servers to satisfy the costumer's needs [3], [4]. In this approach, users' computations are transferred to one or multiple servers on the cloud. These servers process the users' data and return the processed

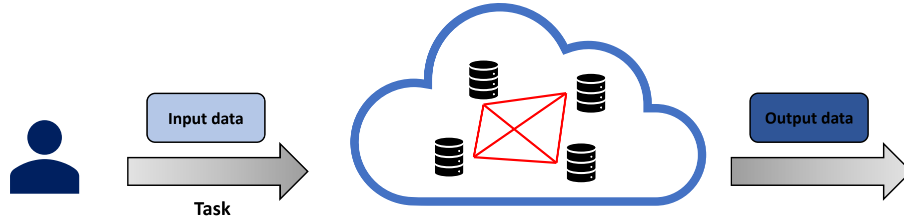

Figure 1.1: Cloud computing schematic

data to the users. Fig. 1.1 demonstrates the main idea.

Cloud services, though gaining much popularity, have several challenges [5], [6]. The systems' reliability is one of the important ones. The user must be assured of the system's success. Moreover, the privacy and security of the system are very important since the users' data could be sensitive. The service provider must also provide the storage and computational needs of the user. Communication limitation is also a contributing factor in designing the system.

1.2 Distributed Computing

Depending on the data size and the needed processing, sometimes a single processing unit – in either the cloud or a local network – is unable to handle all the computations. Moreover, some computational tasks are time-sensitive and must be finished before a tight deadline. The standard solution in such cases is parallel computing, where the tasks are broken into smaller ones and are performed in parallel [7], [8].

Parallel computing at the hardware-level has been a common practice for decades. Having multiple processors in a single device increases the speed of the processing and has been subject to numerous structure designs and hardware development. However, in distributed computing models, several distinct devices are utilized simultaneously, and this introduces new challenges.

Consider a network consisting of a number of helper devices, available to conduct a computational task. This network can be a set of servers in the cloud, devices with limited computational capacity in an Internet of Things (IoT) network, or an Ad Hoc cluster of machines. In any scenario, these

helper nodes possess computational abilities and therefore, can participate in the distributed computing scheme. In this way, some processing units that could potentially be idle benefit other devices.

The helpers can communicate data before, during, and after computation and can receive multiple tasks in order to complete the whole processing. Generally, the master-helper [1] model is the most standard distributed computing model studied [9]–[13]. In this model, data and computational functions are provided to the helpers by the master. When helpers finish their assigned tasks, the master collects the processed data to finalize the computation.

As an example, MapReduce structure introduced in [14], covers several distributed computing models and benefits from the master-helper structure. In this structure, in the Map stage, some helpers receive an assignment to perform a computation based on a fraction of the input data. Commonly, there exist Shuffle stages where the helpers start communicating throughout their computation, and multiple tasks might be completed. In the Reduce stage, some helper nodes perform some final tasks and send the results to the master. Many works [15]–[18] have been developed by adopting this structure. More details on MapReduce is provided in Chapter 2.

1.3 Unreliable Distributed Computing

There are multiple factors that can cause a failure in a distributed computing network [9], [11], [19], [20]. Such failures will result in an unreliable distributed computing system (DCS). It is essential to identify these factors and propose a solution to resolve them in order to obtain a dependable system.

An obvious source of failure is the imperfect communication channel. In particular, many distributed computing networks are wireless [18], [19], [21], [22], i.e., the helper nodes and master communicate through wireless channels. Wireless channels are especially subject to error and erasure. These problems might jeopardize the success of the whole computation.

In addition to channel imperfection, there are other sources of failure in

[1] Also known as master-server, master-worker.

distributed computing. In fact, the main challenge in distributed computing systems is stragglers [11], [23], [24]. These are the helpers that consume more time on their task than expected, which results in the added latency to the system. Resource sharing, power limits, queuing, and many expected or unpredictable reasons contribute to the straggling of helpers [25]. In a time-sensitive computational task, even one straggler can be a bottle-neck for the latency of the whole system and decreases the performance of the distributed computing system.

These factors create unreliability in distributed computing systems. Clearly, one needs to develop a solution to combat these unreliabilities. A trivial approach is to reassign failed tasks to new helpers. Reassigning the tasks to new helpers would both take time and still will not be fully reliable and, thus, is not an efficient solution. Hence, more efficient and more reliable solutions to combat unreliabilities in a distributed computing system are necessary.

1.4 Coded Distributed Computing

To add reliability and allow data recovery in any application, adding redundancy is needed. In distributed computing systems, also, having some extra helpers engaging in redundant tasks is vital because it eliminates the dependency of the system's success on any individual helper. Utilizing these extra resources are equivalent to extra costs; however, the added reliability that is achieved justifies this trade-off.

A trivial method to add redundancy in distributed computing would be to assign the tasks to multiple helpers instead of one. As a result, any node that responds faster would send their result to the master, and the computation would be complete. However, this method is extremely inefficient from a resource management perspective. Therefore, an alternative approach is needed that can offer high reliability but at a reasonable amount of extra cost. In practical settings, and if selected carefully, even 5 % extra helpers can mitigate the straggling effect and reduce the latency [23].

Coding theory has proposed solutions that can be adopted to combat the

unreliability of the helper nodes in distributed computing systems. More specifically, erasure coding, which is a forward error correction (FEC) code, allows the missing blocks of information to be recovered if the blocks are coded in advance. Therefore, if blocks of data are encoded by an erasure FEC code prior to being sent to the helpers, even if some helpers straggle or fail in sending their results, having a sufficient number of the data blocks back, a decoder can recover the erasures and retrieve the results.

Note that if both the computation and the coding are linear, the processed blocks of data are also coded. Therefore, the same decoding method that could be used for helpers' inputs can be applied to their outputs. This approach has been proposed by many recent works and has been studied for several problems and system setups [9], [11]–[13], [16], [24]. The dynamics of coded distributed computing is explained with more details in Chapter 2.

1.5 Thesis overview

Coding techniques can resolve the unreliability of distributed computing, yet they come with some costs. For instance, extra computing resources are required. Moreover, to have coded blocks of data, some pre-processing and post-processing are needed that increase the overall amount of computation. These added processings are called encoding and decoding that the master must perform before sending the blocks, and after receiving the computed results, respectively.

Note that the original goal of a distributed computing system was to reduce the amount of computation in a single machine to reduce the execution time. Now, with the coded distributed computing systems (CDCS), although the original computing is extracted from the master, a new load of computations appears for it. When the added load is insignificant, the reliability gain of coding justifies the added costs. However, there exist cases where the added computations for encoding/decoding is significant and cannot be ignored.

The coding complexity is directly relative to the size of the code, which is typically the number of helpers in the CDCSs. In particular, in setups with

a large number of helpers, the coding complexity, specifically the decoding, is no longer negligible. Therefore, a new method of CDCS is needed to address the high complexity of computations that remain at the master.

In this thesis, we have focused on two problems. First, we have considered the large-scale coded distributed Fourier transform computing. In this type of computation, we have developed a novel distributed computing method evolved from fast Fourier Transform (FFT) structure that allows for low-complexity coding on large-scale distributed FFT computing systems. Moreover, a low-complexity coding solution is adopted to resolve the high-complexity decoding problem discussed earlier.

Second, we consider the high decoding complexity problem in the large-scale CDCSs, i.e., systems with a massive number of helpers. By devising a solution in which helper nodes can participate in the decoding stage, a new coded distributed computing structure is proposed in which the master's load is manageable. In particular, a multi-stage decoding that leaves little work for the master is developed and tested.

The key novelty of the proposed schemes can be summarized as taking advantage of extra stages of computation without involving the master, in order to reduce the master's computation load, and thus, the execution time. This technique, however, requires careful design and consideration that are discussed throughout this work.

The remainder of this thesis is organized as follows. In Chapter 2, we provide some background on the coding theory, distributed computing models, and CDCSs. In Chapter 3, the large-scale distributed FFT computing problem is considered and an efficient solution is proposed. In Chapter 4, we have focused on the possibility of distributed decoding models, and discussed our proposition, formulated some optimization problems, and have suggested efficient solutions for these problems. Finally, in Chapter 5, the overview of contributions and possible future research directions are provided.

Chapter 2

Background

2.1 Distributed Computing Systems

A distributed system is a collection of devices with a single common goal. This system can be a small cluster of devices or a massive complex network. The devices can have a limited computing and memory capacity such as the internet of things (IoT) networks or the network might consist of several high-performance processing units. The communication method between the nodes (devices in a network) could also vary between wired and wireless. Two types of distributed systems, distributed storage and distributed computing systems, have gained a lot of interest in recent years [6], [26].

This thesis particularly focuses on distributed computing systems. The fundamental goal of distributed computing is to perform a computation in the shortest possible time with the available resources. There are a number of attributes that contribute to finding the best DCS for a computing problem. The number of the available nodes and their characteristics such as their memory size, computing capability, and their speed are vital in designing the distributed computing model. When such a network is offered by an external service provider, we can specifically call the system, a cloud computing system. In our discussions, we offer solutions for a general DCS, whether it is a local network or on the cloud.

The main idea behind distributed computing is to break a very large computation into smaller tasks, distribute the small tasks between the computing resources, and collect the results to finalize the original computation. In most

cases, the computing task is either appointed to or offered by a single machine called the master. This node is responsible for delivering the final result to the user. The master employs the other devices in the network, known as helpers, for the tasks. This popular structure is known as the master-helper structure. Although there exist some models that are masterless or have multiple masters [27] or only masters [19], in this work, we specifically focus on the master-helper model, which is the most common in the literature, as well.

2.1.1 System Model

One of the distributed computing models that are compatible with the master-helper is MapReduce [14]. MapReduce splits the steps of distributed computing into two or three categories. These stages are Map, Shuffle, and Reduce, ordered chronologically. In the first stage, i.e., Map, the master provides the helpers the data and their computational task (function). In some cases, the task or the data might be already existing at the helpers. Then, the helpers start to perform the computation assigned to them.

Depending on the computation model, normally, there are some Shuffle stages. In this stage, the helpers exchange their results and reapply the functions to the new data or perform a fresh task. This procedure might be repeated, each time with a new function or different data transmissions between the nodes. In many setups, the master also engages in the shuffling. Here, we have considered a system that allows device to device communication. In some setups, the only means of communication between the helpers might be through the master, working as a relay between the devices.

At last, the master assigns Reduce tasks that the helpers must perform on their received inputs and pass the results to the master or the designated destination [18]. Often, the master will perform some type of post-processing and sorting to the received data to prepare them for the user. Fig. 2.1 demonstrates a general model of MapReduce-derived distributed computing system.

MapReduce illustrates a general but useful structure that DSCs can apply. In most cases, the master breaks very large data into smaller chunks and send them to the helpers for them to apply the same computation on each of them.

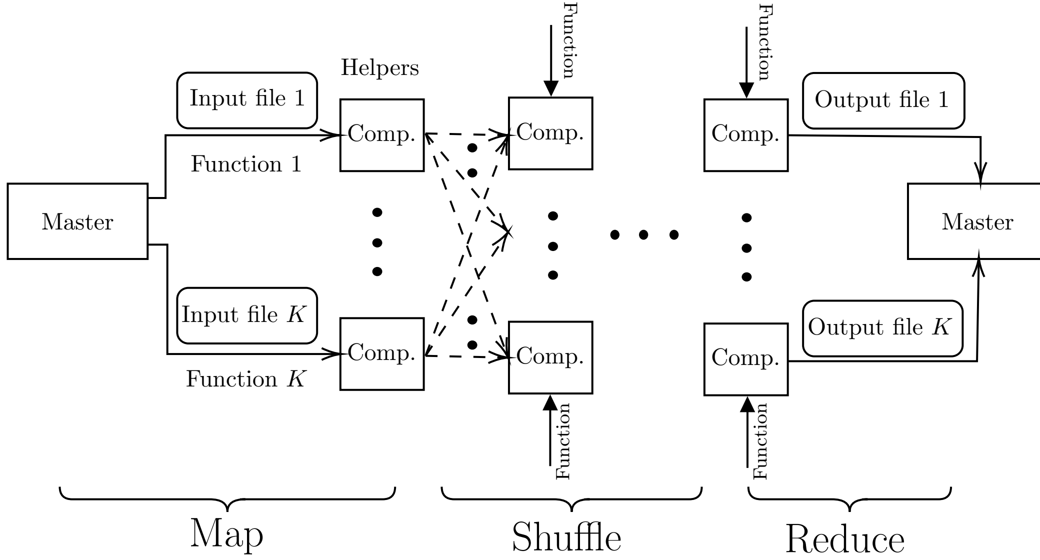

Figure 2.1: MapReduce structure with K processors and multiple Shuffle stages. The helpers might perform different computations in each layer. In Map, the master assigns the data and tasks to the helpers and they start their computational task. They might shuffle their output data with each other, and start a new processing. In Reduce, the helpers perform final tasks and send the results to the master for the final processing (if needed). In some scenarios, the helpers only perform one layer of computation.

Alternatively, all or some of the helpers might be applying different functions on the same blocks of data. Moreover, in the Shuffle stage, the nodes either directly exchange data or send their outputs to the master and it distributes the processed data in a desirable way.

As an example, consider a matrix multiplication task with excessive matrix sizes. The master can break a matrix or both matrices into smaller ones and allocate them to the helpers for the multiplication task. Once the helpers are finished, they can send back their results. The master, then, can gather all the data segments and form the final matrix. Assume that the goal is to calculate $\mathbf{Ax} = \mathbf{y}$ where $\mathbf{A} \in \mathbb{R}^{N \times P}, \mathbf{x} \in \mathbb{R}^{P \times 1}, \mathbf{y} \in \mathbb{R}^{N \times 1}$ and N is large. The master breaks $\mathbf{A}$ into smaller matrices such as $\mathbf{A}_1, \dots, \mathbf{A}_K$ where $\mathbf{A}_i \in \mathbb{R}^{\frac{N}{K} \times P}, 1 \leq i \leq K$, then sends each $\mathbf{A}_i$ with the vector $\mathbf{x}$ to each helper. Now, the helper nodes can do the $\mathbf{A}_i\mathbf{x}$ matrix multiplications and generate $\mathbf{y}_i \in \mathbb{R}^{\frac{N}{k} \times 1}$ vectors. By sending these vectors to the master, the helpers allow it to form the vector $\mathbf{y}$ by combining the $\mathbf{y}_i$ vectors. Therefore, the computation is completed by

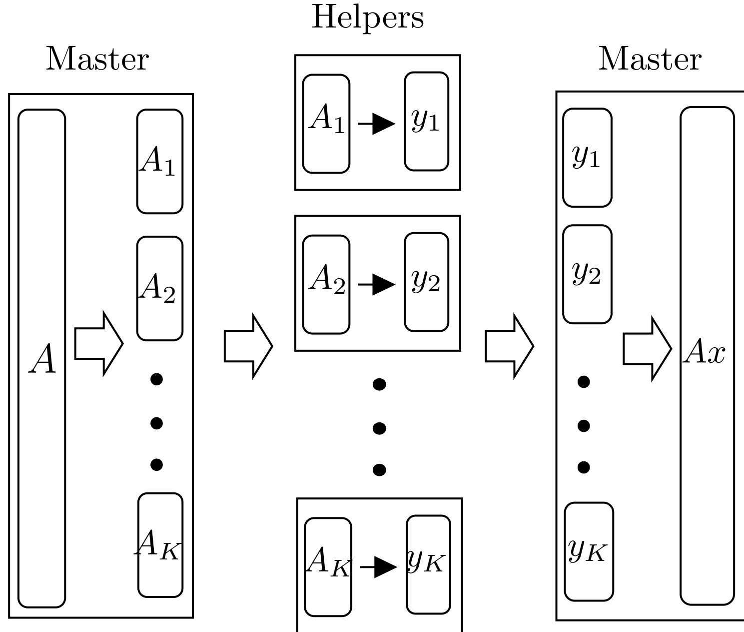

Figure 2.2: Distributed matrix multiplication model with K helpers

the helper's assistance. This process is demonstrated in Fig. 2.2. Note that, if instead of $\mathbf{x}$ the goal was to do $\mathbf{AB}$ matrix-matrix multiplication, the same process could be done.

2.1.2 Challenges

In a DCS, some challenges exist that endanger the success of distributed computing. The key factor in these challenges is that the computation is decentralized. Throughout the whole distributed computing scheme, two things can fail. One is the communication between the nodes, and the other is the task completion in each helper.

Communication networks, in particular wireless ones, are not completely reliable. Complete or partial communication failures are not rare due to the unreliable nature of the wireless (or even the wired) channel. Moreover, in a wireless DCS, the helper nodes might not be fully committed and may leave the network or delay their assigned task for a more prioritized task [22], [28]. This is more common in mobile computing and fog computing networks. However, the most common scenario in the failure of a task is that the computing resources face a problem with the task itself.

In any DSC, the helper nodes might not finish their task by the expected time. There are some nodes that take an excessive amount of time to finish a computational task. These nodes are known as stragglers. The straggling in DSC has an unattractive consequence. A straggling in a single node causes the finish time of the whole computation to be delayed. The main task cannot be finished unless all the subtasks are finished first. If the task is re-assigned to a fresh node, the execution time is at least doubled. Therefore, this problem interferes with the main goal of distributed computing, i.e., completing the task in the shortest possible time. To solve the straggling problem, the best idea is to reduce the dependency of the system on a single node. Such a solution requires implementing a kind of redundancy.

The redundancy helps the computation to be completed even if not all the helpers finish their tasks. To have redundancy, the tasks that are distributed to the helpers cannot be simply partitions of computation but must have a relation to each other. A simple example is to have some replicas for each task. This technique assures if a node straggle, there are other nodes with the same task. Then, it is very likely that one of them responds on time. However, this method is not the optimum way to benefit from computing resources. The reason is that if many nodes perform the same task, the computation cannot be broken into small chunks and the time that each helper must spend on their task is increased. Therefore, although some works have benefitted from this approach [29], it might not be the best practice for many distributed computing problems.

An alternative solution is to consider the output of the stragglers as erasures, i.e., since the outputs take too long to be generated, the system can consider them as erased information. This analogy helps to relate the problem to bit erasures in communication systems and hence, hints at using the solutions that have been developed in that field. In the next part, we will review the coding solutions that are used in faulty communication systems, and then, we will discuss how they can be applied to DCSs.

2.2 Forward Error Correction Codes

In a digital communication channel, it is common that a part of the information (bits) gets erased or be received with an error. Therefore, in order to avoid retransmissions, adding redundancy to the information has been practiced in the transmitter. This act allows the receiver to recover errors or erasures. For example, for an information block of k bits, the process includes generating $n, n > k$ symbols from the original k bits. This process is called encoding with an (n, k) code where n and k are called the size, and the dimension of the code. For this code, the coding rate is $\frac{k}{n}$, and the overhead is $\frac{n-k}{k}$. Thereby, the receiver is able to recover the original k bits by decoding the block of data that may have errors or erasures. These codes are known as forward error correction codes.

FEC codes have been a popular solution in the digital communications area. However, the use of FEC codes is not limited to communication channels. In recent years, the FEC codes' application has expanded to distributed storage and computing systems, where erasure is a common problem. In these applications, the erasure does not just happen to a fraction of bits out of a data block, but a data block is inaccessible, completely. Therefore, predicting such failures in advance, and resolving them by applying FEC codes, makes the distributed storage and computing systems reliable. In Section 2.3, we will discuss the details of this procedure.

2.2.1 Mathematical Background

To understand how the coding work, we need to review some mathematical pre-requisites. In coding theory, we work with the numbers that are members of a finite field $\mathbb{F}$.

Definition 1 *A finite field $\mathbb{F}_q$, also known as Galois field $GF(q)$, is a finite set of q elements if $(\mathbb{F}_q, +, *)$ satisfies the following rules:*

I. $(\mathbb{F}_q, +)$ *forms an Abelian group with additive identity* 0.

II. $(\mathbb{F}_q/\{0\}, *)$ *forms an Abelian group with multiplicative identity* 1.

III. $\forall a, b, c \in \mathbb{F} : (a + b) * c = a * c + b * c$.

A linear (n, k) code $\mathcal{C}$ in a finite field $\mathbb{F}$, generates $n > k$ symbols by doing linear operations on the original k symbols. Linear codes can be represented by a generator matrix $\mathbf{G}^{n\times k}$, and the encoding process can be described as $\mathbf{y} = \mathbf{G}\mathbf{x}$, where $\mathbf{x}^{k\times 1}$ and $\mathbf{y}^{n\times 1}$ are the original, and the coded block, respectively. A coded block is called a codeword. There are 2^k different codewords.

An important feature of linear codes is the minimum distance of the code. The minimum distance of a code is the minimum hamming distance between any pair of codewords. A code with a minimum distance of d, can tolerate up to $d-1$ erasures. This means when $d-1$ elements of $\mathbf{y}$ is missing, $\mathbf{x}$ can still be retrieved.

Definition 2 *For an (n, k) code with a minimum distance of d, it can be proved that*

$$d \leq n - k + 1. \tag{2.1}$$

This inequality is called the Singleton bound [30]. Remember that the code can allow no more than $d-1$ erasures. Therefore, (2.1) implies that the number of erasures must be less than or equal to $n - k$. The codes that achieve (2.1) with equality are called the maximum distance separable (MDS) codes.

2.2.2 MDS codes

MDS codes have been a popular choice of codes for distributed computing problems [9], [12], [31], [32]. Note that for decoding, at least $n-(d-1)$ symbols must be existent. In other words, the decoding overhead, i.e., the number of redundant symbols needed for decoding, is $k + n - (d - 1)$. Since MDS codes achieve the equality in (2.1), they can tolerate up to $n-k$ erasures. This means that with any k coded symbols, the decoding would succeed. In other words, the decoding overhead is zero. This is an attractive feature because it allows a minimal code overhead. Therefore, the number of redundant symbols, in other words, the redundancy cost, is minimized. To construct an MDS code,

one needs to make sure that in the generator matrix, any k rows are linearly independent.

In many applications, it is preferred to have the original k symbols among the n coded symbols. This class of codes is known as systematic codes. The generator matrix of these codes is in

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_{k\times k} \\ \mathbf{G}' \end{bmatrix}$$

format. The advantage of the systematic codes is that when the erasure rate is low, the original symbols could be available without any decoding.

Reed-Solomon (RS) codes

One of the most common types of MDS codes is Reed-Solomon codes [33]. These codes are practical because given n, k, and $\mathbb{F}_q$, the generator matrix of an (n, k) RS code can be defined as follows,

$$\mathbf{G} = \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^k \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \cdots & \alpha_n^k \end{bmatrix} \tag{2.2}$$

where $\alpha_1, \dots, \alpha_n$ are n different numbers in $\mathbb{F}$. Such a generator matrix ensures the criteria that an (n, k) MDS code must have.

The encoding complexity of RS codes is $\mathcal{O}(n)$, and thus, is not a concern. However, the decoding complexity of the codes has proved to be higher. An obvious method of decoding is to use matrix operations which results in a decoding complexity of as high as $\mathcal{O}(n^3)$. However, the decoding complexity was reduced to $\mathcal{O}(n^2)$ [34], and later by the use of fast polynomial multiplication technique, this complexity was reduced to $\mathcal{O}(k \log^2 k \log\log k)$ [1] [31], [35]. Moreover, for a systematic MDS code, the decoding complexity can be shown to be $\mathcal{O}(n(n-k))$. These orders of decoding complexity are not pleasant for some applications. That is why a different method of linear codes known as fountain codes have been developed in recent years.

[1]Throughout this thesis, whenever the base of the log function is not specified, the log is in base 2.

2.2.3 Fountain codes

Fountain codes were first introduced for the broadcast scenarios and have also been suggested for storage systems lately [36]. In broadcast applications, the encoding process is continued until the whole block of data is recovered by all receivers. Hence, these codes are considered as rateless, i.e., there is no fixed n and the coded symbols get generated by the encoder until the job is finished. Therefore, the code overhead of these codes is relatively larger than the MDS codes. In particular, unlike MDS codes, the decoding overhead is not zero. The advantage of the fountain codes, however, is that the generator matrix is not dense. Thus, the decoding complexity is lower.

In fountain codes, for each encoded symbol, d number of the k original data symbols are randomly selected and linearly combined in $\mathbb{F}$. The number d here is called the degree of this encoded symbol. A polynomial $\Omega(\mathrm{x}) = \sum \lambda_i \mathrm{x}^i$ is usually used to represent the fountain codes where λ_i shows the frequency of degree i being used in the encoding. In the decoding, a belief propagation scheme is used where starting with degree 1 and then higher degrees, the original symbols are recovered.

Fountain codes do not guarantee the coverage of all symbols or the linear independence between codewords. Therefore, unlike MDS codes, the decoder needs to receive more than k encoded symbols to succeed. The needed overhead is related to the average degree of the encoded symbols ($\sum \lambda_i i$), where the overhead is smaller when the average degree is higher. Clearly, the decoding complexity is also related to these parameters since the higher degrees require more operations in the decoding process [37]. The earlier versions of fountain codes like Luby Transform (LT) codes [38], required $\mathcal{O}(\sqrt{k})$ average degrees but with the introduction of Raptor codes, using an outer code allowed fountain codes with a fixed average degree and the decoding complexity of $\mathcal{O}(k \log_e(1/\varepsilon))$. For the outer code, the encoder first uses a (K_O, K) to construct K_O intermediate symbols. These symbols will be the inputs of the fountain code encoder.

Now to design a Raptor code, the main goal is to ensure that the decoding

will succeed with a very low failure probability. In the successful decoding process, it is important that the LT part of the decoding would not fail until a $(1-\delta)$ fraction of the intermediate symbols is recovered (Step 1). This will then allow the outer code to recover the k original symbols (Step 2). In [37], it is shown that for a given k, ε_c, and δ, a valid degree distribution for Step 1 must hold the inequality

$$1 - \mathrm{x} - e^{-\Omega'(\mathrm{x})(1+\varepsilon)} \geq \gamma\sqrt{\frac{1-\mathrm{x}}{k}} \tag{2.3}$$

for $\mathrm{x} \in [0, 1-\delta]$. This distribution ensures that the LT part of the decoding will be successful with a failure probability of $1/k^c$ where c is independent of the other parameters. Note that $\Omega'(\mathrm{x})$ is the derivative of $\Omega(\mathrm{x})$, and the coefficient γ must hold the inequality $\delta\sqrt{k} > \gamma$.

In Chapter 3, we will see how Raptor codes can be applied to distributed computing problems.

2.3 Coding in distributed computing

As discussed earlier, in DCSs, we face some challenges such as straggling. When a helper straggles, its output block is considered unavailable. Such a data block will be treated as erasure and a procedure must be done in order to retrieve these missing blocks of information.

As discussed, FEC coding has been proved to be an appealing solution to this problem. With an encoding prior to distributing the data to the helpers, some redundant blocks of data are generated. Therefore, with some straggler nodes, still enough results are accessible to retrieve the needed information.

Coding in distributed computing has some differences with a typical packet coding in communications. Like distributed storage systems, coding in CDS is applied to the blocks of data as a single unit. This means that when a block is not available, the whole symbols are not. So the coding is applied to the K blocks of data no matter the number of symbols each one contains. To generate the symbols of the coded blocks, element-wise coding must be performed, i.e., the ith symbols of the new coded blocks are generated by performing coding on

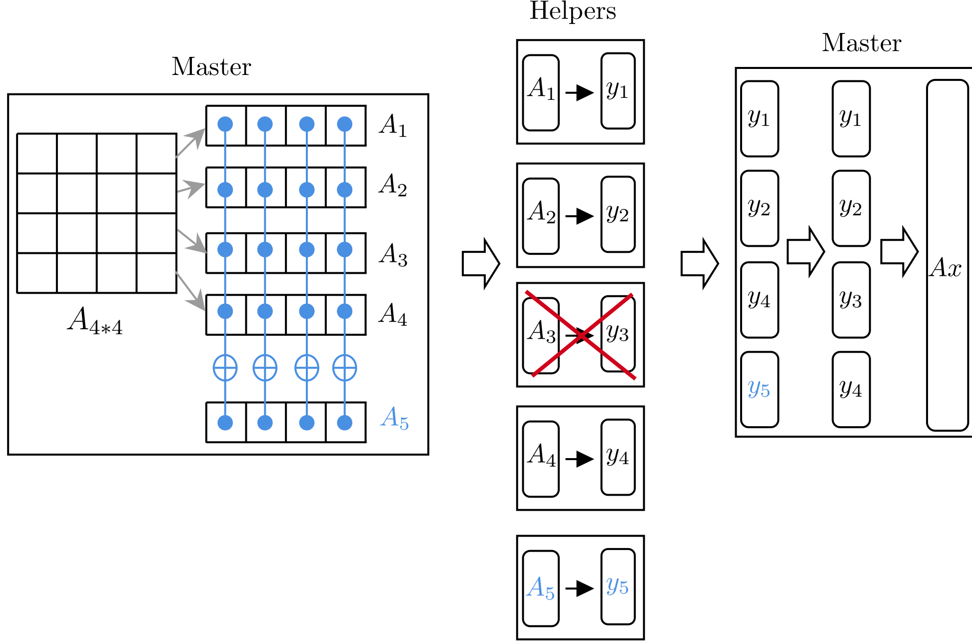

Figure 2.3: Coded distributed matrix multiplication model with 4 helpers

the ith symbols of the original blocks. In the decoding, similarly, element-wise decoding is done.

As an example, assume the computation goal is to find $\mathbf{y} = \mathbf{Ax}$. To distribute this computation we can break the matrix $\mathbf{A}_{(K \times V)}$ into K vectors $\mathbf{A}_1, \ldots, \mathbf{A}_K$. Now, the master can generate redundant blocks such as $\mathbf{A}_{K+1} = \mathbf{A}_1 + \cdots + \mathbf{A}_K$ and assign them along with the original blocks to L nodes. Therefore the ith symbol of $\mathbf{A}_{K+1}$ is the sum of the ith symbols of vectors $\mathbf{A}_j$. An example for $K = 4$ is demonstrated in Fig. 2.3.

Now, note that, unlike storage systems, the data blocks are not the same after the computation is done, i.e., the inputs and outputs of the helpers are not the same. However, when the computation is linear, the outputs will also be coded with the same code that the inputs were coded by. Therefore, even though the symbols are different, the same decoding method can be done to recover the missing symbols.

Consider the example presented earlier. The helpers compute $\mathbf{y}_i = \mathbf{A}_i\mathbf{x}$ and send them to the master. Since the computation in each helper is the same (and linear), we can argue that the ith symbol of $\mathbf{y}_{K+1}$ is the sum of ith symbols of

each $\mathbf{y}_j$. Therefore, for $L = K+1$, any K blocks of $\mathbf{y}_i$ would allow us to decode the desired $\mathbf{y}_1, \ldots, \mathbf{y}_K$. Therefore, a coded distributed computing system can tolerate stragglers as long as the decoding is doable with the received blocks.

Note that the type of computation is not limited to matrix multiplication. If the computation is linear, i.e., the outputs can be claimed to be coded by the same code as the inputs, then coding can be used for that DCS.

Many works in recent years have developed more complex algorithms based on this idea to develop fault-tolerant DCSs. Many works have been developed on optimizing the coding solutions for various types of computations and system models. In the next section, we will review some of these works.

2.3.1 Related works

The works that have proposed FEC codes for distributed computing problems can be categorized in several ways based on the type of computation, the type of code, features of the helpers, and even other applications of coding such as communication and security.

The most common type of computation, matrix multiplication, usually falls under the MapReduce structure. For applications such as machine learning, data analysis, image processing [9], [39], the main computation is transformed into a matrix-vector or matrix-matrix multiplication. In such examples, the matrix can be divided into smaller matrices or vectors and be distributed among helper nodes for the multiplication task. Matrix-vector multiplication has been studied in many recent works [13], [40]. Later, some solutions [31], [32], [34] have been proposed for matrix-matrix multiplication, where both matrices are divided into smaller pieces and are distributed.

Since discrete Fourier transform (DFT) is a linear operation, it can be modeled as matrix multiplication and be implemented using the available coded approaches. However, the complexity of matrix multiplication (that is $\mathcal{O}(N^2)$) is much higher than FFT ($\mathcal{O}(N \log(N))$). Thus, new coded distributed approaches that fit the FFT structure have been recently studied [12], [41], [42].

Distributed learning methods that are based on matrix multiplications have also gained attention, where the most studied type is the gradient descent in

a distributed manner [9], [22], [43], [44].

In some works, no specific codes are suggested, and linear codes as general are proposed for their problem [45], [46]. However, in most works such as [9], [21], [24], [32], MDS codes are suggested. The advantage of MDS codes is their small overhead. Some works [40], [47], however, have studied low-complexity decodings such as LT [38] and Raptor codes [37]. In a recent work [48], polar codes have also been investigated.

Other than the type of the code, the characteristics of the distributed network, (e.g., homogeneous, heterogeneous, number of nodes) also contribute to the performance of the distributed computing systems. In most distributed computing schemes, it is assumed that all the helper nodes are the same type, and thus, the tasks are distributed among them equally. However, despite this assumption of network's homogeneity, in many cases, there are helper nodes with different features available [45], [46]. The memory size, processing unit capacity, the priority they assign to different tasks, and other parameters can all contribute to the performance of a helper, and the processing time it requires to complete a task. Such a network of helpers with different characteristics is called a heterogeneous network.

In many studies, the heterogeneous characteristic has been treated by load-balancing techniques to achieve an optimum performance regarding finishing time or other costs [17], [22]. In works such as [27], based on the helpers' straggling parameters, the sizes of the assigned matrices to the helpers were optimized. The performance of these methods was observed to outperform uniform scheduling or existing load-balancing schemes. In [46], this optimization was conducted when FEC was in place. In [49], the codes for each stage of computation were selected based on the performance of stragglers.

Other than combating stragglers, coding has been proved to be useful for other challenges in distributed computing problems, as well. Several works have been developed on reducing the communication cost of distributed schemes [11], [20], [50]. Moreover, to add security and privacy to distributed computing, coding has been practiced [51]–[53]. In this work, we focus on the straggling problem and benefit from coding to combat straggling.

2.3.2 Decoding Complexity

In coded distributed computing solutions, the computation load is lifted from the master and the coding makes the system reliable. However, new computation loads due to coding are introduced. The decoding load is the largest newly defined computation that the master must handle. In a typical coding solution such as the example given earlier, the size of the code is proportional to the size of the network, i.e., the number of the helpers.

When the computation is massive, the number of helpers needed to assist with the computation becomes very large. The reason is that the helpers' computational capacity is limited. Therefore, the number of needed helpers, and thus, the size of the code becomes very large. Since the decoding complexity is proportional to the size of the code K, the decoding complexity increases, too. Thus, the added load of the master becomes larger. Considering Reed-Solomon as the method of coding used for CDCSs, this complexity would be $\mathcal{O}(K \log^2 K \log\log K)$.

Such complexities could become considerable for a very practical range of K. In some setups, this complexity can even surpass the original computation's load. This will be a very serious issue, questioning the advantage of distributed computing for such setups. Moreover, in scenarios where the master is to perform computations other than decoding, this issue becomes more critical.

2.3.3 Proposed Multi-layer Distributed Coding Solutions Overview

When the master's load due to decoding becomes massive, it seems intuitive to consider distributing its task again. An additional layer of helpers to assist the master with either the remaining computation or the decoding seems tempting. Therefore, one needs to think about the reliability of the added layer as nodes in this added layer can also straggle. Hence, new coded distributed computing structures are needed.

Through this thesis, we work on two scenarios. One, where the master has some computation to do other than the decoding such as distributed FFT

computation. We investigate how another layer of distributed computing can be useful to reduce this complexity. A fully-reliable structure for this purpose requires a novel design that is proposed in Chapter 3. We will also discuss the use of low-complexity codes such as Raptor codes to further reduce the master's load.

In another scenario, we study a solution with an extra layer of helpers engaging only in the decoding. This distributed coding solution will reduce the master's complexity by reducing the size of the code. The master will be only responsible for the coding applied on the decoding helpers' layer. This solution is presented in Chapter 4.

Chapter 3

Multi-layer Coded Distributed FFT

3.1 Introduction

Discrete Fourier transform has been considered as one of the most fundamental operations in computing methods [54]. Many applications of DFT, such as image processing, machine learning, and data analysis, have been identified over the years. As technology advances, the size of data increases, requiring large-scale computations including large-scale DFT [55].

While the computational complexity of direct implementation of DFT is $\mathcal{O}(N^2)$, an improved implementation, called fast Fourier transform, achieves a computational complexity order of $\mathcal{O}(N \log(N))$ [56]. Even with FFT, the computational capacity of a single processing unit may not be sufficient when N is large. This is true for modern applications such as high-resolution image processing and large data set machine learning [2], [57], [58]. In such cases, distributed computing, where the computation load is distributed among multiple devices has been suggested.

For applications such as matrix multiplication, there exist straightforward coded distributed solutions [31], [32]. To benefit from the FFT's low-complexity, different coded distributed solutions are needed. Thus, new coded distributed approaches that fit the FFT structure have been recently studied [12], [41], [42].

For distributed FFT, in [12], a coding idea similar to [31] is proposed that

can tolerate some failures or straggling, without affecting the latency of the computation. This work benefits from polynomial codes (a special case of MDS codes) so that the minimum number of responding nodes can finalize a distributed computation. To do this, first, the large input data is divided into K smaller blocks which by MDS coding are mapped to $L > K$ coded blocks. Then, the FFT of these coded blocks are computed each by a different helper node. A sufficient number of results from the helper nodes lets the master successfully decode for the FFT of the original K data blocks. Then, using the recursive structure of FFT, the master finishes the remaining FFT computation.

In this solution, although the master offloads the FFT computation to the helper nodes, due to the complexity of decoding and the remaining FFT computation, the master still needs to perform a large amount of computation. In fact, for many ranges of parameters, the remaining complexity at the master (which is of order $N \log^2 K \log\log K$) can even exceed the complexity of self-computing the FFT of the whole data. It can be argued that, this scheme works well only when the size of data that the helper nodes can accept M is comparable to the original data size N and hence, the number of helper nodes is small [1]. However, when $M \ll N$ or equivalently when the number of helper nodes is large, we need more efficient solutions. To better see the problem, consider a data of size $N = 2^{30}$ and $K = 2^6$ helper nodes performing M–FFT, where $M = 2^{24}$. Then, in the coded distributed scenario, the master needs to perform about

$$N \log^2 K \log\log K + N \log K = (93 + 6) \times 2^{30}$$

operations which is much higher than the self-computing load of $N \log N = 30 \times 2^{30}$ operations [2].

This motivates us to propose a new coded distributed approach for large-scale FFT computed by the help of a large number of helper nodes. Many

[1] We consider the cases where N, M, K are all powers of two. Thus, their log is an integer.

[2] Here, we used the complexity orders as the actual values. While this is not a perfect comparison, it gives an idea about the problem. Moreover, the constant factor of decoding complexity is usually higher than that of FFT, indicating that the increase in the load of the master can be even worse than the reported numbers.

recent works consider a large number of helper nodes up to thousands of nodes [13], [15], [25], [59]–[61], especially when the size of computation is massive. Moreover, parallel FFT with a large number of processors has been studied in works such as [8], [62]. Therefore, with the growing size of data and distributed computing problems, in this chapter, we have focused on very large scale distributed FFT computation.

Our approach takes advantage of the FFT's own parallel structure to introduce a distributed scheme, specifically for FFT. Using multiple layers of computations, this approach allows for the FFT computation to be fully offloaded on the helper nodes, unlike [12], leaving no FFT computation for the master node, thus reducing its complexity. It will be shown that, unlike existing distributed uncoded FFT solutions [62], [63] that do not readily allow for efficient coding, in our work, coding can be applied on each layer of computation. Also, our algorithm is reliable in every stage and unlike [41] does not rely on the assumption that the helper nodes remain reliable for the shuffle-stage coding.

The challenge for us is that without bringing everything back to the master for the Shuffle stage, we need to perform a reliable distributed shuffle-stage coding by the helper nodes when these nodes might fail or straggle in this process. We meet this challenge by allowing helpers to communicate with each other. As a result, the Shuffle stage encoding and decoding can entirely be assigned to the helper nodes. Such a distributed FFT computation requires a novel distributed coding approach, which is developed in this chapter.

To further improve the master's Reduce stage decoding complexity, Raptor codes are suggested. Raptor codes enjoy a linear coding complexity, hence for larger K, they are much more efficient than MDS codes. After studying the total latency and the offloading cost of the proposed solution, we present numerical studies supporting our findings. We then use our numerical study to adjust the Raptor code parameters to optimize the number of helper nodes in different setups. The results show that the proposed scheme significantly reduces the cost and the master's complexity compared to Self-FFT (by more than 80% in our reported studies).

3.2 Background and motivation

In this section, we first review the idea behind FFT that performs DFT in a faster way. It is shown that there is a distributed nature in FFT structure which we will benefit from when proposing our solution. Later, the challenges that a distributed computing method could face are introduced and it is discussed how coding methods can become useful for resolving these challenges. Finally, the problems that still remain in the existing solutions are explained. These are the problems that this chapter will tackle.

3.2.1 FFT computation

Consider a vector $\mathbf{x}$ of size N. Based on the DFT definition

$$\mathcal{X}_p^{(N)} = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi p}{N}n}, \tag{3.1}$$

where $\mathcal{X}_p^{(N)}$ denotes the pth element of the N-point DFT of the vector $\mathbf{x}$ and x_n denotes the nth element of $\mathbf{x}$. For simplicity, we sometimes use $\mathcal{X}_p$ instead of $\mathcal{X}_p^{(N)}$.

Now, assume that N is even and we divide $\mathbf{x}$ into two equal units $\mathbf{x^o}$ and $\mathbf{x^e}$, where $x_l^o = x_{2l+1}$ and $x_l^e = x_{2l}$, $l = 0, ..., \frac{N}{2} - 1$. Now, using (3.1), we can write

$$\begin{aligned} \mathcal{X}_p^{(N)} &= \sum_{l=0}^{\frac{N}{2}-1} x_l^e e^{-j\frac{2\pi p}{N}(2l)} + \sum_{l=0}^{\frac{N}{2}-1} x_l^o e^{-j\frac{2\pi p}{N}(2l+1)} \\ &= \sum_{l=0}^{\frac{N}{2}-1} x_l^e e^{-j\frac{2\pi p}{\frac{N}{2}}l} + e^{-j\frac{2\pi p}{N}} \sum_{l=0}^{\frac{N}{2}-1} x_l^o e^{-j\frac{2\pi p}{\frac{N}{2}}l} \\ &= \mathcal{X}_p^{e(\frac{N}{2})} + e^{-j\frac{2\pi p}{N}} \mathcal{X}_p^{o(\frac{N}{2})}, \end{aligned} \tag{3.2}$$

where $\mathcal{X}_p^e$ and $\mathcal{X}_p^o$ are the pth element of the $\frac{N}{2}$-point DFT of $\mathbf{x^e}$ and $\mathbf{x^o}$, respectively.

Equation (3.2) indicates that, in order to obtain the DFT of $\mathbf{x}$, we can first calculate the DFT of two smaller vectors whose elements are independent and

then add the elements, respectively, with given weights. Moreover, by defining $W_N^k = e^{-j\frac{2\pi k}{N}}$, it can be shown that

$$W_N^{p+\frac{N}{2}} = e^{-j\frac{2\pi p}{N}+\pi} = -e^{-j\frac{2\pi p}{N}} = -W_N^p. \tag{3.3}$$

Therefore, having the DFT of even and odd parts, the final result can be computed by three operations for any pair of elements as

$$\begin{aligned} \mathcal{X}_p^{(N)} &= \mathcal{X}_p^{e(\frac{N}{2})} + W_N^p \mathcal{X}_p^{o(\frac{N}{2})} \\ \mathcal{X}_{p+\frac{N}{2}}^{(N)} &= \mathcal{X}_p^{e(\frac{N}{2})} - W_N^p \mathcal{X}_p^{o(\frac{N}{2})}. \end{aligned} \tag{3.4}$$

This recursive feature inspired Cooley and Tuckey to introduce the FFT algorithm which requires $\frac{3N}{2}$ operations in each layer. The number of layers depends on how many times the vectors can be divided into two half parts which is $\log(N)$ for any $N = 2^n$ [3]. This proves that the computational complexity of FFT is $\mathcal{O}(N \log(N))$.

The FFT algorithm is demonstrated in Fig. 3.1 for a 16-length vector, where FFT layers are also labeled. This figure helps us discuss the distributed computation design in the next section.

3.2.2 Coded distributed FFT

Consider a master-helper scenario with an FFT computation objective. Assume the master wants to compute FFT of a large vector $\mathbf{x}$ of size $N = 2^n$. The resources at the master are too limited. Hence, instead of self-processing, it is required to distribute the computation load to a number of helpers, where the nodes can store and compute FFT of vectors of maximum size M. So, the master divides the input data into K equal parts of size M.

In [12], they use the fact that Fourier transform is a linear function and thus, treat the computation the same as other linear computations. Therefore, the conventional coding method discussed earlier is applied to the distributed FFT without any Shuffle stages. In other words, the master converts the

[3]The algorithm can be generalized for any value of N. For convenience, we focus on $N = 2^n$ cases.

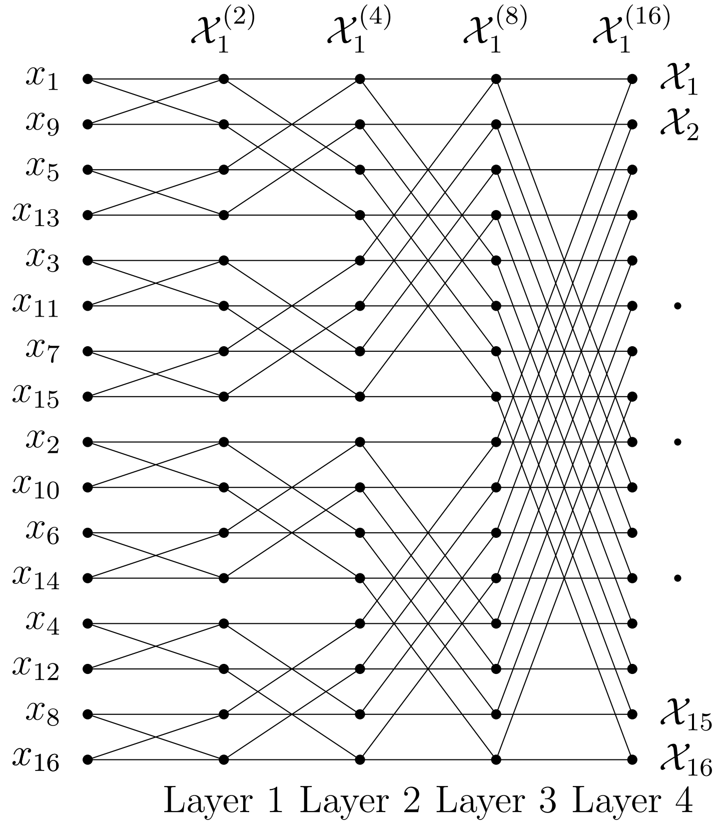

Figure 3.1: The butterfly structure of FFT. Vertices may multiply a weight to their source dots and each dot operates the sum of the input vertices.

original K data block into L coded blocks which are sent to L available helper nodes for M–FFT computation. With the first K results, the master has enough data to continue with the rest of the job by first, decoding and then, completing the rest of the FFT butterfly structure. Notice that since no Shuffle stages are allowed, the FFT butterfly structure must be completed by the master node.

Here, the decoding complexity can be minimized by a combination of Bluestein's algorithm and fast polynomial multiplication [31] with the upper bound of $\mathcal{O}(N \log^2 K \log\log K)$. Moreover, the master does the rest of the FFT computation with the complexity of $\mathcal{O}(N \log K)$. As discussed in the Introduction section, in this approach, the total complexity load on the master can be significant and for some ranges of parameters even an order of magnitude larger than self-computing of the original FFT.

In a different coded FFT approach, developed in [41], the K input blocks are first coded to L blocks for distributed processing. As soon as K processed blocks finish their M–FFT task, the outputs are shuffled for further computing. The helper nodes first decode the shuffled data to recover the uncoded data. Then, the nodes apply the twiddle factor multiplication on the data. Finally, in this stage, the uncoded symbols are sent to the encoder points, so the coded symbols of the second stage can be generated. When the encoding is completed, the L helper nodes will begin the second FFT computation.

The main assumption buried under this approach is that the nodes successful in FFT computation will engage in the Shuffle stage (decoding, multiplication, and encoding), flawlessly. However, the amount of computations needed for the Shuffle stage is in the same order as the FFT computations. Therefore, it is possible that a node succeeds in FFT computation but later fails or straggles in the Shuffle stage. Naturally, we need a coding solution to handle errors and stragglers in the Shuffle stage. Unfortunately, the Shuffle stage in existing coded distributed FFT approaches is not fault-tolerant. Resolving this problem would require rethinking and developing a new approach for distributed coding. In this chapter, such a distributed coding approach is developed and tested.

3.2.3 Remaining challenges and overview of our contributions

The existing coded FFT approaches [12], [41], [42] are efficient solutions when K is small. Note that when K is small, decoding complexity is reasonable, but the helper nodes end up performing very large FFTs. When a large number of helper nodes can be utilized, the above mentioned FFT solutions result in a significant computation load at the master.

In this chapter, we develop efficient coded distributed FFT solutions capable of utilizing a large number of helper nodes. For this, we reorganize the FFT computations into multiple layers [4]. We apply coding on each layer

[4]Note that different layers can employ the same helper nodes. Throughout this chapter, we consider the general case with independent nodes for the ease of demonstration.

separately, making all the layers reliable. Finally, it will be shown that using Raptor codes, the remaining load at the master due to encoding/decoding is drastically reduced. This comes at the cost of a few more helper nodes, but our assumption is that the bottleneck is the master's complexity load and not the number of helper nodes.

3.3 Uncoded distributed FFT solution

In this section, we discuss an uncoded distributed FFT approach, where the FFT computations are all done by the helper nodes in multiple layers. This approach has been practiced in parallel FFT schemes and replaces a large FFT with smaller FFTs. For now, let us make the simplifying assumption that there are no failures or stragglers. Also, consider the MapReduce method with zero or more Shuffle stages, where K available nodes are occupied in each computation layer (Map stage plus Shuffle stages).

Assume that we have decided to break the computation between K nodes. In the first layer, each node receives a vector of size M, denoted by $\mathbf{a}_1, \ldots, \mathbf{a}_K$, in the Map stage. Each node, in this layer, is supposed to compute the M–FFT of its input, denoted by $\mathbf{b}_1, \ldots, \mathbf{b}_K$. Note that, this stage fulfills the computation happening in the first $\log M$ FFT layers of the N–FFT computation demonstrated in Fig. 3.1. Now, still $\log N - \log M = \log K$ FFT layers worth of computation is remaining. If there is more than one computation layer, in the first Shuffle stage, the KM outputs are shuffled, the twiddle factors are multiplied [5], and the symbols are sent to the second computation layer in a way that each node in the next layer can perform FFTs on its M inputs. This process continues until the whole FFT is complete. In the ℓth layer, the jth element of the ith node will be $\mathcal{X}_{Mi+j}^{(M\ell)}$. Fig. 3.2 shows a 16–FFT broken into two 4–FFTs.

In the first computation layer (and with a similar argument in all other

[5] In order to successfully break the N–FFT into smaller FFTs, e.g., multiple M–FFT or an M–FFT and a K–FFT, we need to process the symbols in the Shuffle stage. This processing involves an extra twiddle factor multiplication, which can be done either in the previous layer, after FFT computation, or in the next layer, before FFT computation. This processing includes a smaller than N number of operations.

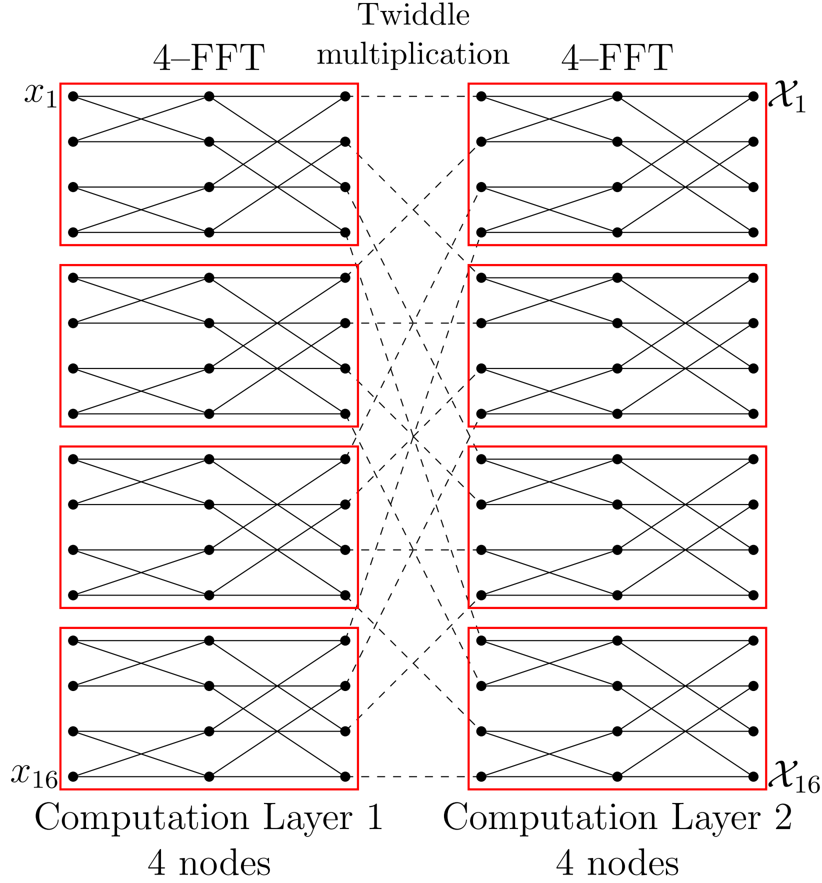

Figure 3.2: Distributed FFT example for $N = 16$, $M = 4$ completed in two layers with four helper nodes in each layer.

computation layers), there exist $KM \log M = N \log M$ operations. Based on the values of M and N, one may choose the number of computation layers. Also, depending on the master's computational abilities, one may choose to assign the computations from layer j onward to the master. In one extreme, as in [12], we have zero Shuffle stages. In other words, only the computations in Layer 1 are done by the helpers and all the rest of FFT computation is done by the master. In the other extreme, the maximum number of shuffle stages is used. That is, the helpers complete all the tasks, leaving no computations for the master node.

Remark 1 *We can assume that in Computation Layer j, there are K_j nodes with M_j inputs such that $M_j K_j = N$. This way, helper nodes in different layers compute FFT operations of different sizes. In that case, the load left for*

the master would be $N\log(N - \sum M_j)$.

Remark 2 *For load balancing purposes* [64]*, and assuming all helper nodes have similar computational abilities, it is desirable to have* $K_j = K$ *for all* j. *When* $K_j = K$, *the maximum number of computation layers is* $\lceil \frac{\log N}{\log M} \rceil$. *In this case, there are* $\log N \pmod{\log M}$ *FFT layers left in the last computation layer. Therefore, the nodes at this layer perform fewer FFT computations per input. In that case, for all the layers but the last one* $M_j = M$, *meaning that* M*–FFTs are repeated.*

Remark 3 *In practical scenarios, it is reasonable to assume that* $M \geq \sqrt{N}$. *In other words, a single helper node can take an FFT of size at least* $\sqrt{N}$, *otherwise, we would need a very large number of helper nodes* $K > \sqrt{N}$. *As a result, we have* $\lceil \frac{\log N}{\log M} \rceil = 2$, *meaning that no more than two layers are needed. Therefore, although the proposed method can be generalized to any number of layers, in the remainder of , we focus on the two-layer designs.*

As a result of the above remarks, we can simplify our general model into a two-layer structure with K nodes in each layer. The first layer is responsible for M-FFT computation in each of the K nodes. After the shuffling, the Computation Layer 2 nodes finish the FFT task by performing multiple K–FFTs on the corresponding K symbols from their M input symbols.

Fig. 3.2 shows an example of the above discussions for $N = 16$ and $M = K = 4$, where the computation is fully distributed, i.e., the number of layers is $\lceil \frac{\log N}{\log M} \rceil = 2$ meaning that the master does not do any computations in the Reduce stage other than resorting the results. Thus, the remaining computational load for the master node is zero. Here, the communication load of this approach is $(1 + \lceil \frac{\log N}{\log M} \rceil)N$.

3.4 Coded solution

In the previous section, we made the simplifying assumption that there is no failure or straggling problems. In reality, these problems exist, and we need to apply coding to resolve them. In this section, we show how a linear coding can

be tailored to the distributed FFT structure discussed above which enables helper nodes to participate in the decoding process.

To ease the notations, and as justified earlier, we assume there are only two computation layers, both layers having the same number of helper nodes. The notation $E_{\ell j}$ will be used for the jth helper node in the Computation Layer ℓ.

3.4.1 Toy example

Consider an example based on Fig. 3.2 setup, where $N = 16$, and $M = 4$. Discussions on this example are demonstrated in Fig. 3.3. Let us assume we want to add only one redundant helper node to each layer. First, the master groups the input vector $\mathbf{x}$ into blocks of size $M = 4$ to form $K = 4$ blocks $(\mathbf{a}_1, \ldots, \mathbf{a}_4)$. The redundant block $\mathbf{a}_5$ is constructed as

$$a_{5,i} = a_{1,i} + a_{2,i} + a_{3,i} + a_{4,i} \qquad \text{for } i \in \{1, \ldots, 4\}, \tag{3.5}$$

where $a_{j,i}$ is the ith element of $\mathbf{a}_j$. This is a simple example of a $(5, 4)$ code. Now, the master sends these five blocks to five helper nodes for FFT computation in the Map stage.

Node E_{1j} receives block $\mathbf{a}_j$ and generates a vector $\mathbf{b}_j$ with the same size, that is the M–FFT of $\mathbf{a}_j$. Since FFT is a linear operator, the relation in (3.5) holds for the FFT blocks, as well. Therefore,

$$b_{5,1} = b_{1,1} + b_{2,1} + b_{3,1} + b_{4,1}, \tag{3.6}$$

and thus, having four of these symbols let the fifth one to be recovered by performing a linear decoding.

Assume that E_{14} is a straggler and its output $\mathbf{b}_4$ is not available for the next layer while the other four nodes have responded. To provide the inputs for the next computation layer, all $b_{j,1}$ elements must be gathered in the node E_{21} but $b_{4,1}$ symbol is missing. In our coded solution, E_{21} can locally recover $b_{4,1}$ symbol from other available symbols, i.e., $b_{1,1}, b_{2,1}, b_{3,1}, b_{5,1}$, using (3.6). Similarly, all E_{2j} nodes will recover $b_{4,j}$ and then have the inputs they need ($c_{i,j} = W_4^{(i-1)(j-1)} b_{j,i}$) to perform the second FFT and complete the whole FFT.

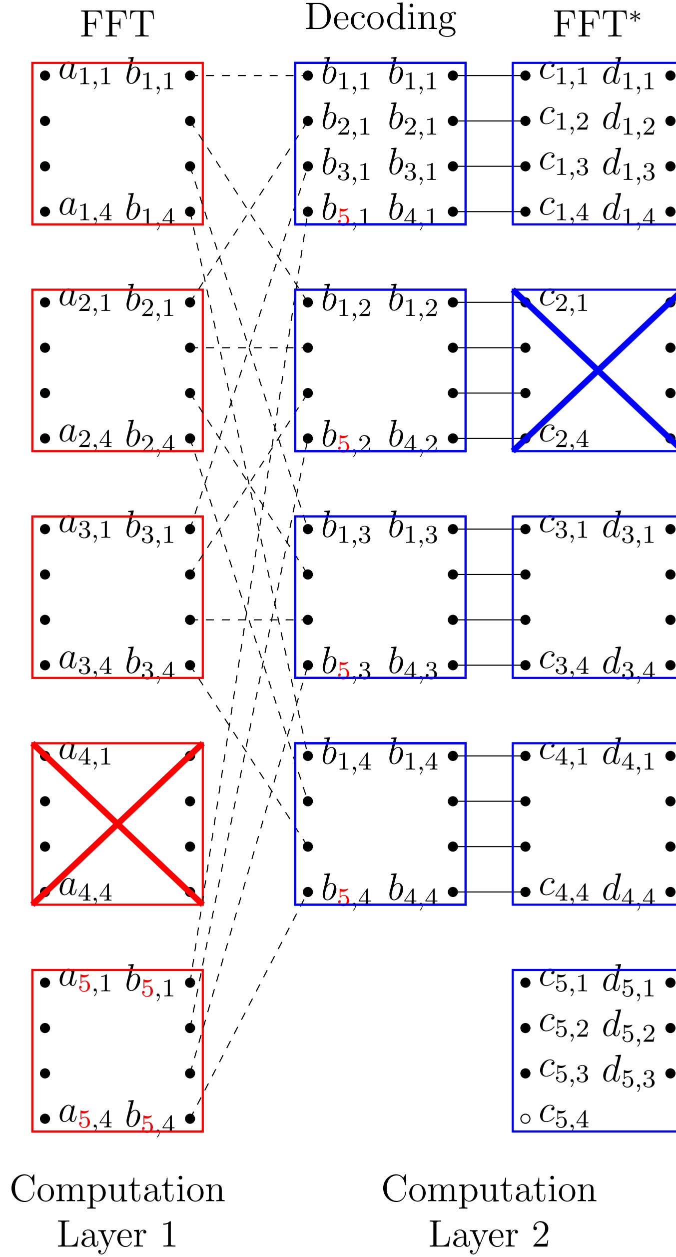

Figure 3.3: The coding scheme example of a two-layer distributed FFT. Generating the coded symbols for redundant nodes is not feasible unless the required symbols for encoding exist in the same place.

Since failure or straggling can happen in the second layer, as well, we need to use coding to make this layer redundant, too. Therefore some blocks such as $\mathbf{c}_5$ are needed that their symbols are element-wise coded symbols of other c_i blocks. In order to do this stage of the encoding, we first need the decoded

symbols of $\mathbf{b}$. However, after decoding, the symbols needed for encoding are located in different nodes. One solution to generate $c_{5,4}$ is to have an encoder node to collect all $b_{4,i}$ symbols from the second layer nodes' decoders outputs. The encoder having this systematic block of the unresponsive node in the previous layer, e.g. $\mathbf{b}_4$, applies the same code as the first layer nodes to construct the missing coded symbols at the second layer ($c_{5,4}$). However, this scheme relies on other nodes for decoding and if one straggles, the encoding process can not be completed in time. Also, note that if the code was non-systematic, this same problem for $c_{5,4}$ would have occurred for all $c_{5,j}$ symbols. When the number of coded blocks increases, this method's unreliability becomes more serious.

To resolve this problem, each node in Computation Layer 2 must be able to decode the desired symbols, apply the twiddle factors, and encode the symbols before starting the FFT. Therefore, each node after decoding has access to uncoded $b_{j,i}$ symbols including $b_{4,i}$. They can multiply twiddle factors to generate $b'_{j,i}$ symbols. Then, each node E_{2j} generates symbols $c_{i,j}$ having the $b'_{j,i}$ symbols. As a result, for example, $c_{i,1}$ symbols are coded even though the encoding was not done in a centralized way but locally in each node. Each E_{2j} performs a different encoding, but it repeats that for all the symbols. In other words, each node performs a row of the encoding matrix, locally. This approach is demonstrated in Fig. 3.4.

To better understand how this idea works, consider a redundant block (node E_{25} in our example). It receives $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_5$ and decodes to recover $\mathbf{b}_4$, as well. Then, it generates $b'_{j,i} = W_N^{(i-1)(j-1)} b_{j,i}$ symbols. Now, assuming the code being systematic, the other nodes simply allocate $c_{i,j} = b'_{j,i}$, and proceed. In E_{25}, the coded symbols are generated as follows,

$$c_{5,j} = b'_{1,j} + b'_{2,j} + b'_{3,j} + b'_{4,j}. \tag{3.7}$$

Therefore, this node will proceed with 4–FFT, too, while we have five nodes where one is redundant.

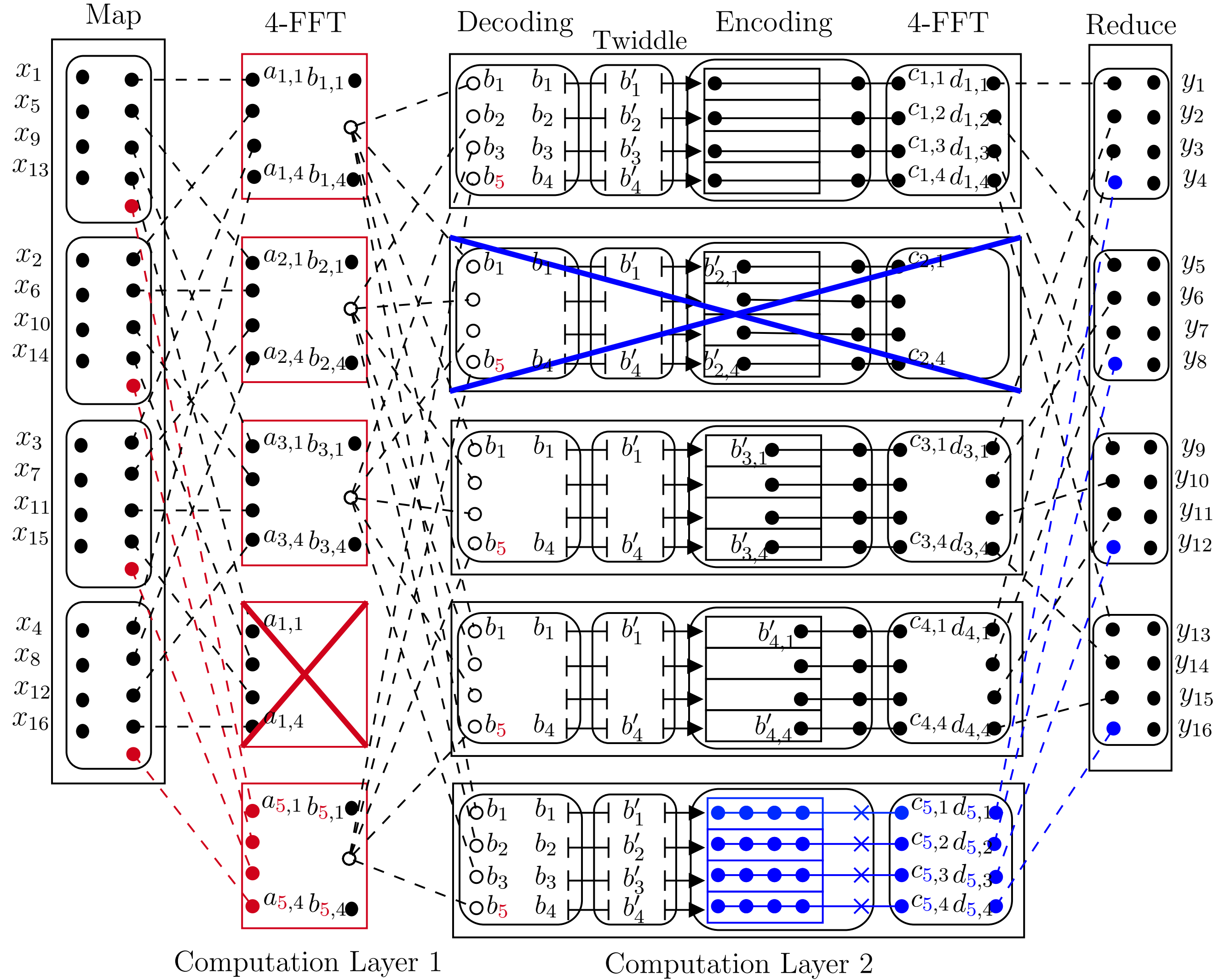

Figure 3.4: The complete proposed coded distributed FFT scheme for $(N, M, K, L) = (16,4,4,5)$. The master performs encoding in Map stage. K helpers participate in M–FFT and the successful nodes send their results to Computation Layer 2. There, helpers decode to recover $\mathbf{b}_4$ symbols, multiply twiddle factors, and encode the symbols they need for the second FFT. In this systematic example, only E_{25} does the encoding. The master finally decodes and retrieves $\mathbf{y}$.

Now, the second layer computation can start which gives output blocks $\mathbf{d}_i$. The master collects the outputs of the second layer nodes and is able to decode them to achieve the systematic symbols. For instance, if the node E_{22} straggles, the master uses the E_{25}'s outputs and can recover the missing symbols of $\mathbf{d}$. The decoded symbols will form the vector $\mathbf{y}$ which is the Fourier transform of $\mathbf{x}$. This solution is demonstrated in Fig. 3.4 as our complete coded approach. Note that in this solution, the master performs only the initial encoding and final decoding. No FFT operation or middle-stage encoding/decoding is done by the master.

Remark 4 *In* (3.7)*, we can expand the equation into*

$$c_{5,j} = b'_{1,j} + b'_{2,j} + b'_{3,j} + b'_{4,j} = b_{1,j} + W_4^{(j-1)} b_{2,j} + W_4^{2(j-1)} b_{3,j} + W_4^{3(j-1)} b_{4,j}. \quad (3.8)$$

Therefore, we can merge the twiddle factor multiplication and encoding at each node. This modified encoding with new weights will reduce the computation load caused by these two stages.

To better understand the difference between our approach and that of [12], [41], here, we discuss how our toy example is performed in [12] and [41]. In [12], $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_5$ will be received by the master, decoded to retrieve $\mathbf{b}_4$, and then, without further encoding, the next stage of FFT computation is fulfilled. This means the master still has a considerable computation load. In [41], $E_{11}, E_{12}, E_{13}, E_{15}$ nodes share their symbols such that E_{1j} will have $b_{1,i}, b_{2,i}, b_{3,i}, b_{5,i}$, and it will decode to recover $b_{4,i}$. The twiddle factors are multiplied to generate $b'_{j,i}$s. Here, $c_{i,j}$ symbols cannot be generated because the symbols of each row are in different nodes, unless, a new shuffling stage is added. Recall that shuffle stages are prune to straggling themselves.

3.4.2 Coding algorithm

Now, consider the general case where a vector $\mathbf{x}$ of size N is divided into K vectors of size M denoted by $\mathbf{x}_1, \ldots, \mathbf{x}_K$, where $\mathbf{x}_i = [x_i, x_{K+i} \ldots, x_{M(K-1)+i}]$. We now define the matrix $\mathbf{X}_{K \times M}$ where its ith row is $\mathbf{x}_i$. The goal is to calculate

$$\mathbf{Y} = \mathbf{F}^* (\mathbf{W} \odot (\mathbf{F} \mathbf{X}^{\mathrm{T}})^{\mathrm{T}}), \quad (3.9)$$

where $\mathbf{F}^*{}_{K\times K}$ represents the second layer K–FFT computations and $\mathbf{F}_{M\times M}$ represents the first layer M–FFT computation. The $\mathbf{W}$ is the twiddle matrix where $W_{ij} = W_N^{(i-1)(j-1)}$ [6], and $\odot$ represents element-wise multiplication known as Hadamard product. We then have

$$\mathcal{F}\{\mathbf{x}\} = \mathbf{y} = [\mathbf{Y}_1^{\mathrm{T}} \cdots \mathbf{Y}_K^{\mathrm{T}}], \tag{3.10}$$

where $\mathbf{Y}_i$ is the ith column of $\mathbf{Y}$.

The master encodes the K data blocks to $L > K$ coded blocks using some (L, K) linear block code. This means we need L helper nodes in each computation layer. Therefore, the master applies the linear code on columns of $\mathbf{X}$ in an element-wise fashion to achieve $\mathbf{a}_1, \ldots, \mathbf{a}_L$. Similarly, we define the matrix $\mathbf{A}_{L\times M}$, where its ith row is $\mathbf{a}_i$. If we represent the code with its generator matrix $\mathbf{G}_{1_{L\times K}}$, we have

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_L \end{bmatrix} = \mathbf{G}_1\mathbf{X} = \mathbf{G}_1 \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_K \end{bmatrix}. \tag{3.11}$$

Now, each helper node performs an M–FFT transform on an $\mathbf{a}_i$ block which can be represented by a matrix as in DFT computation as follows,

$$\mathbf{b}_j = \mathbf{F}\mathbf{a}_j \quad \text{for} \quad j = \{1, \ldots, L\}, \tag{3.12}$$

or equivalently,

$$\mathbf{B} = (\mathbf{F}\mathbf{A}^{\mathrm{T}})^{\mathrm{T}} = (\mathbf{F}\mathbf{X}^{\mathrm{T}}\,\mathbf{G}_1^{\mathrm{T}})^{\mathrm{T}} = \mathbf{G}_1(\mathbf{F}\mathbf{X}^{\mathrm{T}})^{\mathrm{T}}. \tag{3.13}$$

Note that, the encoding is performed on the columns of $\mathbf{X}$, while for FFT, the rows of $\mathbf{A}$, i.e., coded $\mathbf{X}$, are used.

Assume $\widetilde{K}$ number of these blocks are sufficient to successfully decode for $\mathbf{F}\mathbf{X}^{\mathrm{T}}$. The jth elements of all these blocks are the $\widetilde{K}$ coded symbols that can be decoded to the jth column of $\mathbf{F}\mathbf{X}^{\mathrm{T}}$. We define the matrix $\widetilde{\mathbf{B}}_{\widetilde{K}\times M}$ such that its rows are the output blocks of the first $\widetilde{K}$ responsive nodes in the first layer.

[6]Note that linear block codes are defined over Galois fields. To make FFT computation compatible with this, $W_N^1 = e^{-j\frac{2\pi}{N}}$ in the FFT operation must be replaced by w which is the Nth root of unity in the chosen Galois field $\mathbb{F}$.

Now, as discussed earlier, the matrix $\widetilde{\mathbf{B}}$ must be sent to the nodes in Computation Layer 2. Using (3.13), the matrix $\widetilde{\mathbf{B}}$ can be written as $\widetilde{\mathbf{G}}_1(\mathbf{F}\mathbf{X}^{\mathrm{T}})^{\mathrm{T}}$, where $\widetilde{\mathbf{G}}_1$ is the first codes' encoder matrix with missing rows. Therefore, we can represent the decoding as

$$\mathbf{Z} = \widetilde{\mathbf{G}}_1^{-1}\widetilde{\mathbf{B}} = \widetilde{\mathbf{G}}_1^{-1}\widetilde{\mathbf{G}}_1\,(\mathbf{F}\mathbf{X}^{\mathrm{T}})^{\mathrm{T}} = (\mathbf{F}\mathbf{X}^{\mathrm{T}})^{\mathrm{T}}, \tag{3.14}$$

which is the decoded M–FFT result.

The twiddle factor multiplication and encoding for the second layer can be expressed as,

$$\begin{aligned} \mathbf{B}' &= \mathbf{W} \odot \mathbf{Z}, \\ \mathbf{C} &= \mathbf{G}_2\,\mathbf{B}'^{\mathrm{T}} = \mathbf{G}_2\,(\mathbf{W} \odot (\mathbf{F}\mathbf{X}^{\mathrm{T}})^{\mathrm{T}})^{\mathrm{T}}. \end{aligned} \tag{3.15}$$

Now, each row of the matrix $\mathbf{C}$ must be subject to K–FFT computation. In our toy example, we had $M = K$, hence both layers had the same computation size. In general, M may not be equal to K. In this case, the first layer nodes can do M–FFT operations as usual. For the second layer nodes, we define M/K virtual nodes that are grouped to one helper node. This way the computation load of the helper nodes in both layers will be similar. Therefore, each node performs the processing in (3.15) partially to construct the M/K rows that it needs. Therefore, each node can locally merge their twiddle factor multiplication and encoding into a single computation and save computational complexity.

The K–FFT will start in the L nodes in Computation Layer 2. Again, some helper nodes may straggle. When the first sufficient number of nodes respond, we can write their outputs as

$$\begin{aligned} \widetilde{\mathbf{D}} &= (\mathbf{F}^*\widetilde{\mathbf{C}}^{\mathrm{T}})^{\mathrm{T}} = \widetilde{\mathbf{C}}\,\mathbf{F}^{*\mathrm{T}} = \widetilde{\mathbf{G}}_2\,(\mathbf{W} \odot (\mathbf{F}\mathbf{X}^{\mathrm{T}})^{\mathrm{T}})^{\mathrm{T}}\,\mathbf{F}^{*\mathrm{T}} \\ &= \widetilde{\mathbf{G}}_2\,(\mathbf{F}^*(\mathbf{W} \odot (\mathbf{F}\mathbf{X}^{\mathrm{T}})^{\mathrm{T}}))^{\mathrm{T}} = \widetilde{\mathbf{G}}_2\mathbf{Y}^{\mathrm{T}} \end{aligned} \tag{3.16}$$

Now, the master node will collect these blocks and decode the received blocks to

$$\mathbf{Y}^* = (\widetilde{\mathbf{G}}_2^{-1}\widetilde{\mathbf{G}}_2\mathbf{Y}^{\mathrm{T}})^{\mathrm{T}} = \mathbf{Y},$$

which is the desired result.

Remark 5 *If M/K number of virtual nodes become merged to a single helper node, the process of all the decoding and computation remains the same but happens in one place. Since these blocks are in the same helper node, either all of them become available for the decoding or none. Therefore, we can reduce the length of the codes from M to $M\frac{K}{M} = K$ meaning that the parallel elements of the actual helper nodes are coded instead of the virtual nodes. For the same reason, it was more accurate to rearrange $\mathbf{B}'$ such that each M/K row becomes a single row and the matrix ${\mathbf{G}_2}_{\frac{LM}{K} \times M}$ would become ${\mathbf{G}_2}_{L \times K}$. In conclusion, the codes that are actually used are (L, K) codes. The advantage is that there are only L nodes required who perform computation on M symbols which makes the input size in both layers balanced.*

Remark 6 *For the cases where the system can benefit from a systematic code, it is sufficient to find $\mathbf{G}_1$ and $\mathbf{G}_2$ such that,*

$$\mathbf{G}_1 = \begin{bmatrix} \mathbf{I}_{K \times K} \\ \mathbf{G}'_1 \end{bmatrix} \quad \text{and} \quad \mathbf{G}_2 = \begin{bmatrix} \mathbf{I}_{M \times M} \\ \mathbf{G}'_2 \end{bmatrix}.$$

Remark 7 *The proposed coded distributed scheme is applicable to structures with more than two layers. The same procedure for the second layer is repeated for all added layers. The coding and shuffling procedures are the same. The only difference comes from the number of nodes that are involved in the shufflings. We need more than two layers only if $M < \sqrt{N}$, meaning that $K > M$. Therefore, in all but the last layer, the nodes receive their inputs from M nodes of the previous layer and perform M–FFT. So, not all the K nodes interact with each other in these Shuffle stages* [7]*.*

So far, we have presented a coded distributed FFT using two layers of computation which is fully coded and offload the whole FFT computations from the master. The coding scheme was not limited to any of the linear codes. This brings us to our choice of codes which is discussed in the next part.

[7] In this remark, we have assumed $M_1 = M_2 = \cdots = M_j = M$. In the general format, these details might defer, as well.

3.4.3 Low decoding complexity codes

While the literature of distributed coded computing is mainly focused on MDS codes, these codes have a superlinear complexity with the block size. In settings where K is large, the coding complexity can be an issue. Luckily, there exist efficient coding solutions, such as some fountain codes, with linear coding complexity. We discussed these codes in some details in Section 2.2.3.

In a MapReduce distributed computing model, the master sends out the data to other nodes for computation and waits for the results. Since only a fixed number of helper nodes can be reserved for the computation, the code length L must be finite. Therefore, the fountain code that is applied to this problem cannot be rateless. The number L should be large enough to compensate for straggling nodes as well as for the overhead of the fountain code. Hence,

$$L \geq L_{\min} = K(1 + \varepsilon_c + \varepsilon_s), \tag{3.17}$$

where ε_c reflects the fountain code overhead and ε_s the effects of the stragglers. Since utilizing extra helper nodes comes at a cost, choosing a small L is more desirable. Note that rateless coding for distributed computing has been proposed in approaches such as [47], but their rateless coded strategy cannot be directly applied to our proposed distributed computing structure. This is because our structure has multiple layers of coded FFT computation.

To generate these L blocks, the encoder first uses a (K_O, K) outer code to construct K_O intermediate blocks. Then, in the (L, K_O) fountain part with $\Omega(\mathrm{x})$, for each λ_i, the encoder constructs $L_i = \lceil L_{\min}\lambda_i \rceil$ encoded blocks of degree i from the intermediate blocks. Then, the total number of helper nodes used will be $L = \sum L_i$ [8].

Now, we need to select the Raptor code design for our application. A degree distribution based on (2.3) is proposed in [65]. This design is later adapted to a fixed-length code for longer size codes [66] and other designs such as [67] for smaller sizes have been suggested. It is shown that these designs achieve the

[8]Note that L can be slightly larger than $L_{\min}$ because of the ceiling operations. The effect is small, typically resulting in less than 10 extra helper nodes.

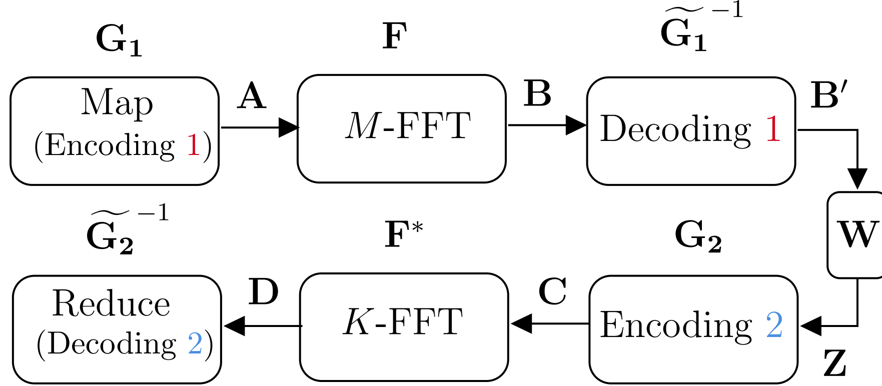

Figure 3.5: System block diagram

linear decoding complexity and their performance for different sizes of codes are studied. The key point in the latter designs is to minimize the number of inactivations which is the bottle-neck of the decoding complexity while satisfying a target failure probability. In all designs, with enough overhead (ε_c), the number of inactivations is shown to be small enough such that the overall decoding complexity (including back substitution, Gaussian elimination, and the outer code) remains linear. An example is provided in Section 3.5. This will then reduce the decoding complexity from $\mathcal{O}(K \log^2 K \log\log K)$ to $\mathcal{O}(\alpha K)$, where α is independent of K. This results in a significant reduction of the master's load.

Remark 8 *The outer code can be represented by its generator matrix* $\mathbf{G}_O$ *which is constructed by combining smaller dense and sparse matrices to achieve a good outer code [37]. Furthermore, the fountain part of the code can also be represented by a sparse matrix* $\mathbf{G}_F$*. This* $L \times K(1+\delta)$ *matrix is the combination of rows where their non-zero elements correspond to the positions of the intermediate symbols that were used to construct a coded symbol. Therefore, the generator matrix of the code is going to be* $\mathbf{G} = \mathbf{G}_F\,\mathbf{G}_O$*. This* $\mathbf{G}$ *can be used as* $\mathbf{G}_1$ *and* $\mathbf{G}_2$ *in the block diagram of Fig. 3.5.*

3.5 Performance analysis and numerical results

In this section, we first analytically study the proposed algorithm's reliability and compare it with [41]. In addition, we will show numerically how the proposed algorithm performs compared to other schemes. Furthermore, for different setups, the choices of design parameters such as ϵ_c and K are studied.

3.5.1 Failure probability

Assume that the added overhead due to ε_s reduces the failure probability of a layer from p to $p_e \ll p$. Here, p is the probability that not enough number of helper nodes in a layer respond in time and thus, make the whole process fail. Since the M–FFT, shuffle-stage decoding and the second layer FFT computations are in the same order of the complexity, we assume that the two layers of our algorithm and the three stages in [41], all can be modeled with the same failure probability p. Therefore, in our scheme, the whole process will fail with the probability of

$$P_1 = 1 - (1 - p_e)^2 = 2\,p_e - {p_e}^2 \simeq 2\,p_e.$$

However, since the shuffle-stage coding in [41] is not coded, the failure probability will be

$$P_2 = 1 - (1 - p_e)(1 - p)(1 - p_e) \simeq p + 2\,p_e \simeq p.$$

Therefore, the reliability of our proposed solution can be significantly higher.

Remark 9 *Although the failure probability is significantly reduced, it is still non-zero. Note that most existing solutions (all MDS-coded solutions) suffer from a similar problem. The only cases that do not suffer from this issue are rateless coded solutions, where waiting longer for more coded blocks may resolve the failure. Although rateless codes cannot be used in a multi-layer structure with master-free shuffling, an alternative solution is possible, where the master computes the missing data locally. Note that, the master always has the input data and can construct the FFT outputs using partial DFT matrix multiplication in order to retrieve the missing symbols. This will put an additional load*

on the master. However, this extra load is not of major significance, considering that it is rare that the number of straggling nodes exceeds the considered overhead.

3.5.2 Complexity and cost comparison

In our complexity/cost comparison, since multiple objectives, such as optimizing the processing time, or the number of helper nodes, can be considered, we first define a cost function to capture these objectives. More specifically, we define the total cost as

$$C_{\text{Total}} = L\,(C_{h_1} + C_{h_2}) + \psi_t\, C_t, \tag{3.18}$$

where the first term represents the cost of offloading the computation to external resources, in which L is the number of helper nodes in each computation layer, and C_{h_1} and C_{h_2} are the loads (number of operations) per helper node in the first and second layer, respectively.

The second term in (3.18) represents the penalty associated with the latency of the process, in which C_t is the total latency, and ψ_t is the time penalty coefficient. This coefficient can be used to adjust the emphasis on time penalty versus the cost of offloading the computations. Note that, C_t can be written as follows,

$$C_t = C_m + C_{h_1} + C_{h_2}, \tag{3.19}$$

where C_m is the master's load. Also, since the computations in the middle stages are done in parallel, the time penalty of a single helper node must be considered.

The terms C_{h_1}, C_{h_2}, and C_m depend on several factors such as M, K, and the choice of the code. For the choice of the code, we consider the Raptor codes introduced in [68] which are based on the degree distribution

$$\begin{aligned}\Omega(\mathrm{x}) = {} & 0.0098\,\mathrm{x}^1 + 0.4590\,\mathrm{x}^2 + 0.2110\,\mathrm{x}^3 + 0.1134\,\mathrm{x}^4 \\ & + 0.1113\,\mathrm{x}^{10} + 0.0799\,\mathrm{x}^{11} + 0.0156\,\mathrm{x}^{40},\end{aligned}$$

introduced in [69]. There, the number of inactivations n_{ID} is studied for different lengths and overheads. The term n_{ID}^3 is known to be a dominant term in the decoding complexity in addition to $K \log_e(1/\varepsilon_c)$ for the back-substitution.

Table 3.1: The master's load is presented for $N = 2^{32}$, $\varepsilon_c = 0.10$, and a range of K. The numerical column is normalized to the first row, i.e., Self-FFT.

Method	Master's load			
	General	Numerical, normalized to Self-FFT		
		$K = 2^6$	$K = 2^8$	$K = 2^{10}$
Self-FFT	$1.5\, N \log N$	1	1	1
[12] MDS	$N\,(1.5 \log K + \log^2 K \log\log K)$	2.13	4.25	7.23
[12] Raptor	$N\,(1.5 \log K + 2\, \log_e 1/\varepsilon_c) + M\, n_{\text{ID}}^3$	0.35	0.41	0.44
Our scheme	$2\,N\, \log_e 1/\varepsilon_c + M\, n_{\text{ID}}^3$	0.17	0.16	0.13

First, we compare the load at the master for a number of approaches. This comparison is represented in Table 3.1 for $N = 2^{32}$, $\varepsilon_c = 0.10$, and a range of K. For MDS, the encoding load is ignored as it is negligible compared to the decoding load. It is clear that MDS is not a good choice for coding in this setup. To provide a more fair comparison with [12], we have applied Raptor coding to [12] and have reported the complexity results in Table 3.1. Even when both solutions use Raptor codes, the table shows that our work has significantly lower complexity than [12], mainly because it completely removes the FFT load from the master.

Next, we use the cost function defined earlier to study the effect of the added overhead. For the scheme proposed in this chapter,

$$C_{h_1} = 1.5\,(M \log M), \tag{3.20}$$

$$C_{h_2} = M\,(K \log_e(1/\varepsilon_c) + n_{\text{ID}}^3) + (\frac{M}{K})\,(K \log_e(1/\varepsilon_c)) + 1.5\,(\frac{M}{K})\,(K \log K). \tag{3.21}$$

The term in (3.20) corresponds to the M–FFT in each helper node. Also, in (3.21), the first term represents the decoding in the second layer, the second term represents the encoding, and the third term represents the K–FFT computations. Similarly,

$$C_m = M\,(K \log_e(1/\varepsilon_c)) + M\,(K \log_e(1/\varepsilon_c) + n_{\text{ID}}^3), \tag{3.22}$$

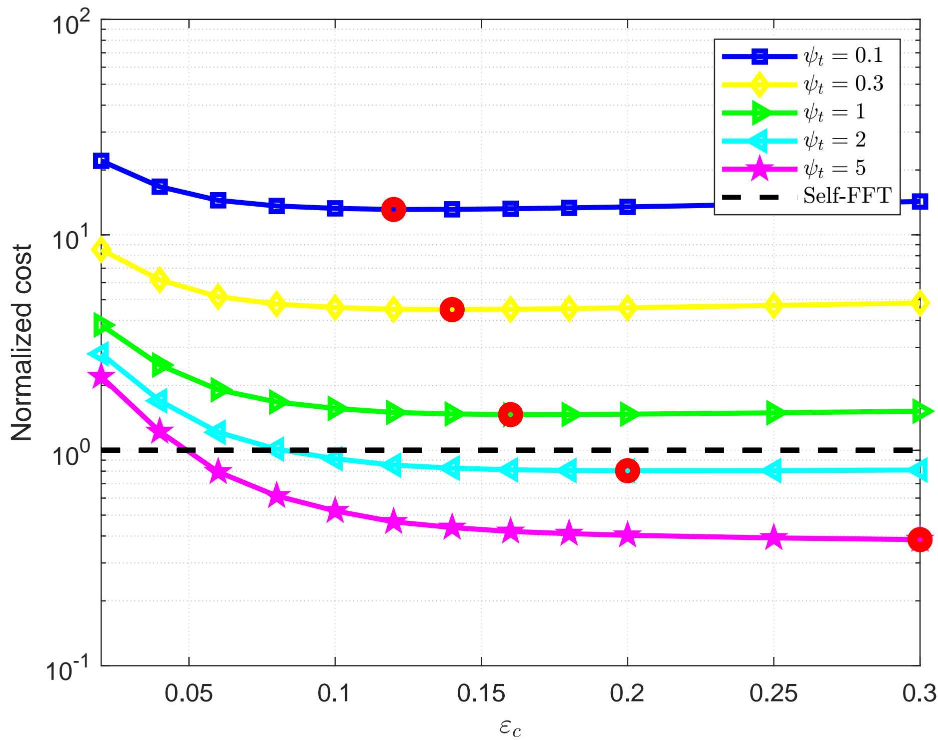

Figure 3.6: Normalized cost of the proposed algorithm for different values of ψ_t. The minimum point follows a trend: it occurs on higher overheads for higher ψ_t values.

where the first term corresponds to the initial encoding and the second term corresponds to the final decoding, both performed by the master. Note that increasing the number of helper nodes increases the first term of the (3.18) whereas it reduces the straggling time and the decoding complexity and hence, the second term. Therefore, there exists a trade off between L and C_t.

This trade off leads to finding the optimum number of helper nodes to achieve the minimum cost as shown in Fig. 3.6. These points determine the favorable ε_c and so, the optimum number of helper nodes. In this simulation, for $K = 1024$, $N = 2^{32}$ and $\varepsilon_s = 0.05$, the normalized cost of the proposed algorithm for a number of ψ_t were computed based on a range of ε_c. The cost of self-computing the FFT in the master, i.e.,

$$C = (1.5\, N \log N)\, \psi_t,$$

is considered as the base cost for the normalization. It can also be observed

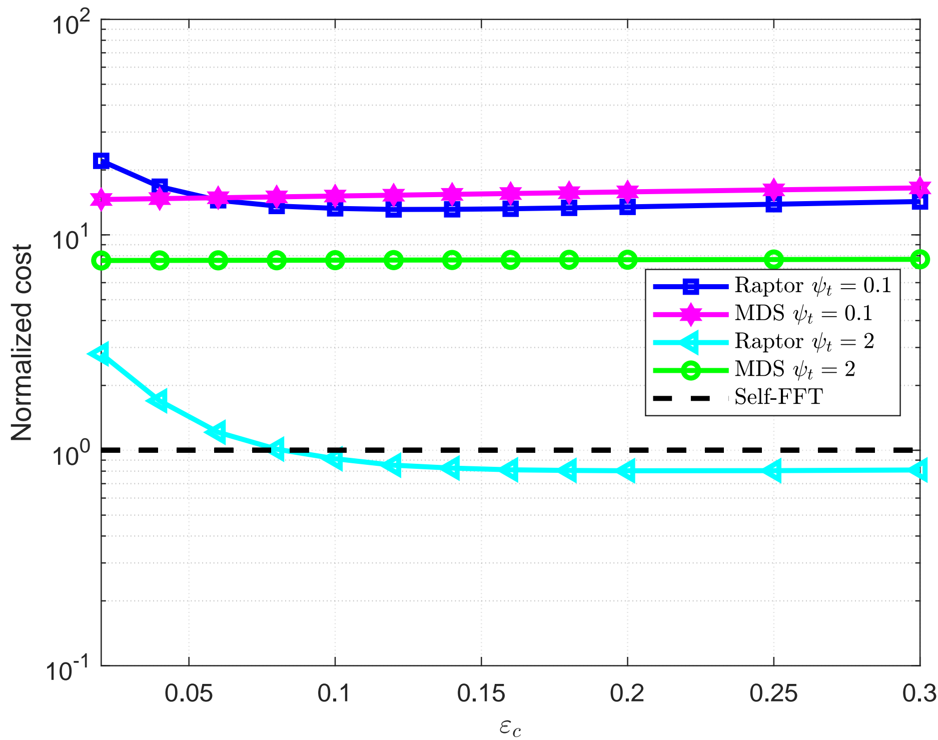

Figure 3.7: Comparison between the proposed algorithm with two computation layers and Raptor coding and single computation layer with MDS coding.

in Fig. 3.6 that the more expensive the helper nodes become, the higher the cost becomes.

In Fig. 3.7, the proposed approach is compared to the scheme using MDS code with one computation layer. For this scheme, we consider

$$\begin{aligned} C_h &= 1.5\,(M \log M), \\ C_t &= M\,(1.5\,K \log K + K\,\log^2 K\,\log\log K) + C_h, \end{aligned} \tag{3.23}$$

where only the dominant factors are considered for C_t. Although the load per helper node is smaller than our scheme, the time and the total complexity are higher for most regimes. The curves extend our previous comparison of master's load to the total cost and demonstrate the level of efficiency of our algorithm. Our complexity saving becomes more significant for higher ψ_t. The MDS-based solution is not affected much by the choice of ψ_t because its complexity is mainly due to decoding at the master. In our solution, however, the master's complexity is not necessarily the dominant factor, hence, changing

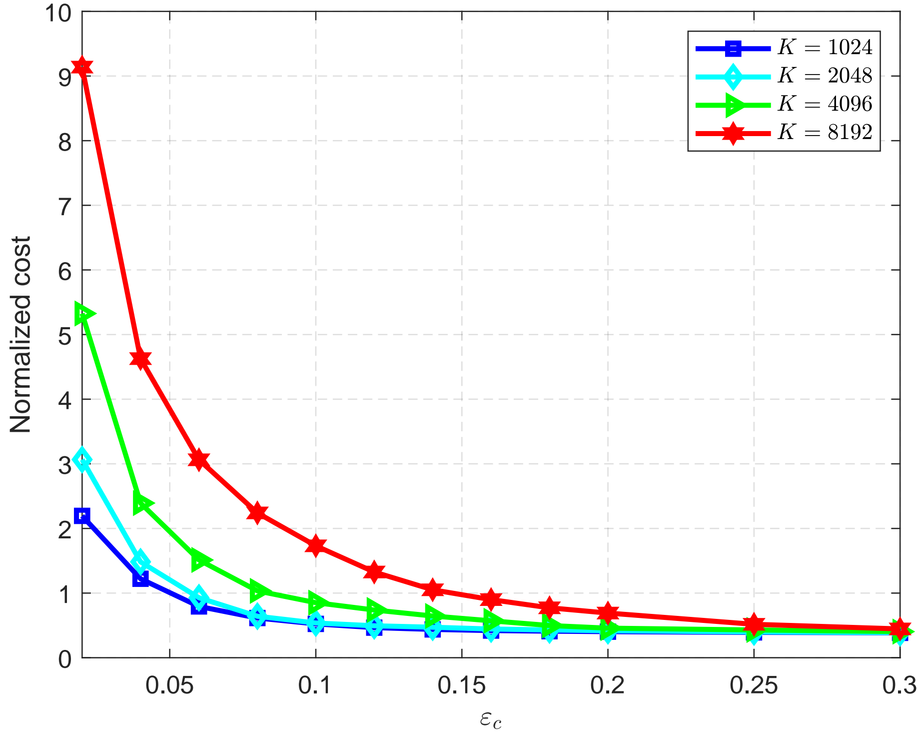

Figure 3.8: The proposed algorithm's cost for a number of code lengths for $N = 2^{32}$.

ψ_t can significantly affect the efficiency.

It should be noted that for some setups, the existing MDS-based approach outperforms our proposed scheme. One case as demonstrated in Fig. 3.7 is for small ψ_t and small overhead (ε_c), where the Raptor decoding becomes complex. However, in such setups, distributed schemes are worse than Self-FFT. The other case happens when K is small. This is because the superlinear factors in the MDS solution are negligible for short K. In contrast, Raptor codes do not perform well for short K as they need a larger overhead. As was mentioned earlier in this work, our proposed solution is targeted for setups with very large N that requires a large number of helper nodes, i.e., large K.

Furthermore, in Fig. 3.8, the impact of the code length is demonstrated using the data obtained from [68]. As expected, as K increases, the code complexity also increases, resulting in a higher total cost. The conclusion here is that one should use the highest size of the data M that can be handled by

a helper node since it results in a smaller K and less decoding cost. However, as the overhead increases, this impact is alleviated and the curves converge.

3.6 Conclusion

In this chapter, we focused on the challenges that distributed FFT computing faces when a large number of helper nodes are needed. In the proposed solution, we suggested two computing layers, where each layer was coded, independently. It was shown that the helper nodes can participate in the encoding/decoding reliably, leaving the master out of the Shuffle stage. Supported by the numerical results, we observed that for a large number of helper nodes, low-complexity coding such as Raptor codes are highly beneficial. As a result of the master-less Shuffle stage and linear-complexity coding, we achieved significantly lower complexity at the master.

Chapter 4

Distributed Decoding

4.1 Introduction

In this chapter, we consider the coded distributed matrix-vector multiplication problem, and similar to most existing literature, we mainly focus on Reed-Solomon codes. When the number of helpers is large and the original computation is massive, the decoding complexity that is left for the master could be very large. This load can even be comparable to the initial load of the original matrix multiplication task. Therefore, in order to minimize the total computation time, we need to be cognizant of the master decoding time and try to reduce the master's decoding load.

In this chapter, we utilize the network's heterogeneity in a novel way. In particular, we develop a multi-layer coding scheme to combat the straggling problem and also allow some helper nodes to be involved in the decoding process. As a result of this design, the encoding/decoding load of the master is significantly reduced. This creates a trade-off since the load of the helpers has increased in return. The challenge here is to ensure that the overall finish time is indeed minimized. This trade-off results in an optimization problem that is studied in Section 4.3.

After proposing a strategy for simplifying the optimization, we study different scenarios, optimize the solution for each case, and analyze the performance of the proposed scheme. Significant completion-time reductions (as high as more than 90%) are observed.

4.2 System model

Consider a heterogeneous network with helper nodes of v different types. For each type i, $1 \leq i \leq v$, assume there are n_i helper nodes with the straggling parameter μ_i. This means that when a task, requiring at least T seconds, is assigned to such a helper node, its runtime CDF is [9], [46],

$$P[t \leq \tau] = 1 - e^{-\mu_i(\frac{\tau}{T}-1)} \quad \text{for } \tau > T.$$

This represents the probability of the task being finished before time τ. Note that T is a function of the allocated load, and thus, it is constant here. The exponential distribution is suggested since it is the closest representation of the behavior of the servers in cloud systems. Without loss of generality, we assume $\mu_1 > \cdots > \mu_v$, meaning that type 1 is the most reliable helper type.

For the matrix multiplication problem introduced in Section 4.1, the matrix $\mathbf{A}_{N \times P}$ is first divided into $K = \frac{N}{M}$ equal parts, $\mathbf{A}_1, \ldots, \mathbf{A}_K$. These new matrices are then encoded, using an (L, K) Reed-Solomon code, to generate coded matrices $\mathbf{B}_1, \ldots, \mathbf{B}_L$. Each matrix $\mathbf{B}_\ell$ is sent to a helper node along with the vector $\mathbf{x}$, so that $\mathbf{y}_\ell = \mathbf{B}_\ell \mathbf{x}$ is computed. Therefore, L helper devices are to be utilized where $L = L_1 + \ldots + L_v$ and $L_i \leq n_i$ represents the number of helpers of type i that are selected for this computation. This scheme is demonstrated in Fig. 4.1.

For the uncoded case, i.e., $L = K$, the time needed for matrix multiplication to be finished τ_{p} is then defined as

$$\tau_{\text{p}} = \max_{w} \tau_w. \tag{4.1}$$

The max operation above can be responsible for the straggling effect. As discussed, to resolve the straggling impact, coding must be applied. In the coded scenario, where $L > K$ helper nodes are used, we need not wait for all helpers to respond. The helpers' finish time is when the fastest K helpers out of L return their computation. At this point, the Reed-Solomon code can be decoded. In other words, the maximum multiplication time of the fastest K helper nodes defines the matrix multiplication time (τ_p). Regarding the

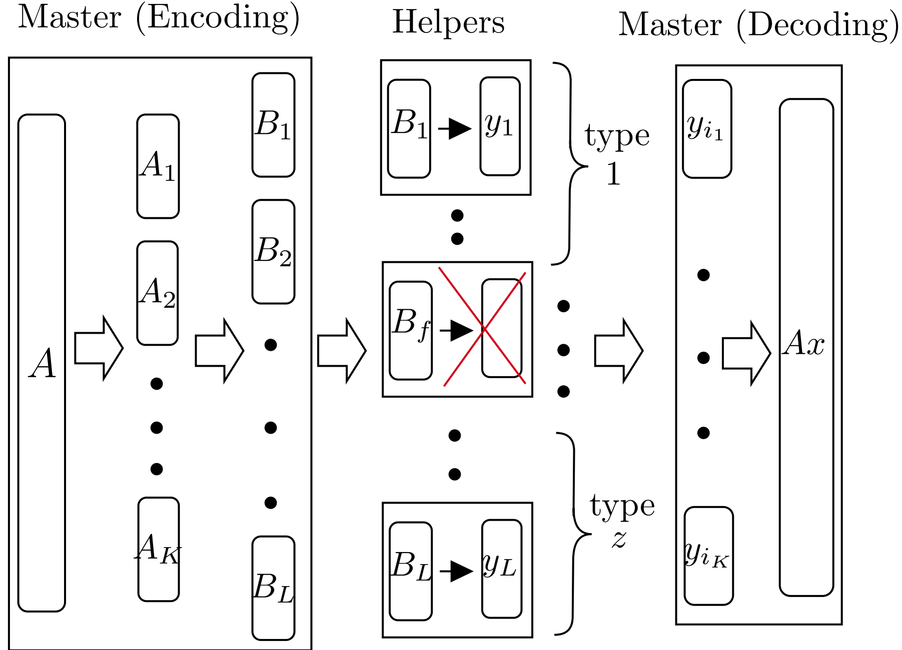

Figure 4.1: The coded matrix-vector multiplication distributed computing scheme based on MapReduce with no Shuffle stage

load allocation, a thorough study and optimization have been done in [46]. To minimize $\mathbb{E}[\tau_{\text{p}}]$, for both coded and uncoded cases, there is a trivial solution where

$$L_1 = n_1,\ L_2 = n_2,\ \ldots,\ L_z \leq n_z,\ L_{z+1} = 0,\ \ldots,\ L_v = 0. \tag{4.2}$$

The objective of a distributed computing system, however, is to minimize the total execution time τ_{ex}. The total execution time is a random variable whose exact value varies from one realization to another. Hence, for the structure introduced in Fig. 4.1, this parameter can be expressed as

$$\mathbb{E}[\tau_{\text{ex}}] = \mathbb{E}[\tau_{\text{p}}] + \tau_{\text{mas}}. \tag{4.3}$$

Here, τ_{mas} is the master's decoding time. This factor impacts τ_{ex} independently of the performance of the helpers.

Consider an example where the computation of $\mathbf{A}_{N\times P}\mathbf{x}_{P\times 1}$ is distributed among $L > K = \frac{N}{M}$ helpers. Each helper performs $\mathbf{y}_i = \mathbf{B}_i\mathbf{x}$ multiplication where $\mathbf{B}_i \in \mathbb{R}^{M\times P}$, and, $\mathbf{y}_i \in \mathbb{R}^{M\times 1}$. For a systematic (L, K) Reed-Solomon, the decoding complexity is $O(K(L-K))$, and for large K, employing polynomial codes [31] can reduce the complexity to $O(K\log^2 K \log\log K)$ [35]. Note

that the $\mathbf{y}_i$ vectors each consist of M symbols. It means that the decoding must be repeated M times, i.e., the actual decoding computation complexity is M times larger. Since decoding is traditionally the master's task, this leaves $M\,K \log^2 K \log\log K = N \log^2 K \log\log K$ computations for the master alone. This is while the load of each helper is MP.

When K is large, the master's load and thus, τ_{mas} might be much greater than τ_{p}. The decoding time for a number of scenarios can be the main bottleneck, and hence, the computing load allocation becomes less important. As an example, note that for a case where $M = 1000, K = 1000, P = 1000$, the load of each helper is $MP = 10^6$ while, the master's load is $MK \log^2 K \log\log K \simeq 3 \times 10^8$ and thus, τ_{mas} is much larger than τ_{p}. Therefore, any solution towards minimizing the average execution time $\mathbb{E}[\tau_{\text{ex}}]$ must consider τ_{mas}, meaning that we need approaches to reduce τ_{mas} by using the available resources in the network.

Imagine that there were completely reliable helper nodes available in the network. Then, they could have helped the master in the decoding stage, thus, reducing the decoding load at the master and, consequently, the overall time. Assume there are Q helper nodes available for decoding. This means that the time required for the decoding stage is reduced to smaller than $1/Q$ of its initial value. In the previous example, for $Q = 20$, the decoding complexity will be $M\frac{K}{Q} \log^2 \frac{K}{Q} \log\log \frac{K}{Q} \simeq 4 \times 10^6$, much smaller than 3×10^8. Since the decoders are entirely reliable, there is no concern about the straggling of the newly defined nodes, and thus, no additional layer of coding on these Q nodes is required. This scheme is demonstrated in Fig. 4.2.

In practical settings, however, there are no perfect helpers. Essentially, we have to pick up from the available helpers that have a non-zero chance of straggling. Hence, entrusting decoding to unreliable helpers can endanger the success probability of the whole computation. In short, there exist two choices, none of them attractive. One choice is to engage helpers in the decoding and lose the reliability of the computation. The other choice is to leave the decoding for the master to ensure reliability, but suffer a long execution time. In the next section, we propose a third choice that is both reliable and fast.

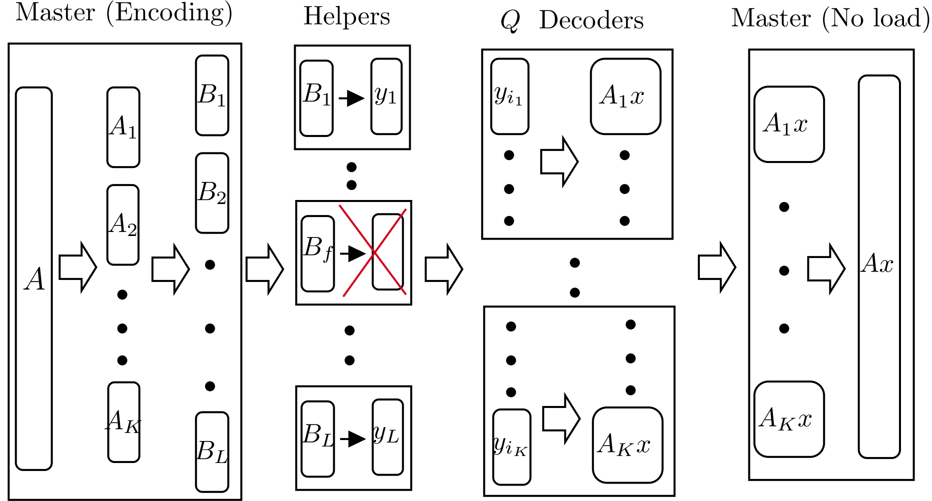

Figure 4.2: The coded matrix-vector multiplication distributed computing scheme with perfect decoding helpers

4.3 A fast and reliable distributed decoding solution

4.3.1 Main idea

Let us first discuss the big picture of our proposed solution. The main idea is to involve some helpers in the decoding process. Now, in order to make the decoding performed by the helpers a reliable process, we also add another layer of coding for these helpers. It is not clear, though, if such a solution helps the overall performance. Certainly, the problem with the unreliability is mitigated, and the solution will be fully reliable. However, in this approach, the master still needs to decode the output of the decoding helpers. Moreover, two decoding completion time must be considered in the total execution time. In other words, the expected execution time is now

$$\mathbb{E}[\tau_{\text{ex}}] = \mathbb{E}[\tau_{\text{p}}] + \mathbb{E}[\tau_{\text{dec}}] + \tau_{\text{mas}}, \tag{4.4}$$

where $\mathbb{E}[\tau_{\text{dec}}]$ is the time needed for decoding helpers to finish their task. Note that the master now needs only to decode a short code. This is because the code length at the master is now reduced from K in the conventional solution to Q in the suggested approach, and Q is much smaller than K. Therefore, there exists a trade-off between the added time τ_{dec} due to an extra decoding

layer and the reduced τ_{mas}.

In this chapter, we investigate the suggested two-layer coded system and study parameters such as the number of helpers to participate in the decoding, and the length of the new coding layer, in order to minimize the total execution time. We will see that this proposed scheme can significantly outperform existing solutions.

4.3.2 Proposed distributed decoding solution

Consider a structure with two layers [1] of helper nodes: (i) processing nodes that compute matrix multiplication, and (ii) decoding nodes that perform the first stage of decoding. Also, assume there are n_i helpers of type $i, 1 \leq i \leq v$ available. Based on the optimization problem that will be discussed later, we first assign L_p nodes to the processing layer and L_d nodes to the decoding layer.

Assuming there are $K = \frac{N}{M}$ matrix multiplication tasks in total, these tasks are grouped into Q batches and an $(L_q, \frac{K}{Q})$ code is applied to each batch. Later, these Q batches are also encoded by an (L_d, Q) code. This second layer of coding adds redundancy and hence, reliability to each batch, as well. Therefore, $L_d L_q$ coded tasks are created and sent to L_p processing helpers in the first layer. As a result, $L_q = \frac{L_p}{L_d}$. This scheme is demonstrated in Fig. 4.3.

Once a sufficient number of helper nodes respond for the decoding to be initiated, the first layer nodes send their output to the designated decoding helpers. These helpers decode the $(L_q, \frac{K}{Q})$ code such that the decoding helper $r, 1 \leq r \leq L_d$ decodes the batch r outputs. Now, since these nodes are also subject to straggling, the (L_d, Q) code ensures the master that by even receiving any Q decoded batches from the decoding helpers, it can decode the second layer of coding and recover the uncoded computed blocks of data.

[1] A structure with more than two layers either has multiple processing layers which is not advantageous for a matrix multiplication problem or has more than one layer of decoders. When helpers are not perfect, we cannot avoid decoding at the master. As we will see, even with one layer of helpers, we can reduce the master complexity to values comparable with the complexity of helper nodes. Hence, multiple decoding layers cannot offer any extra advantage.

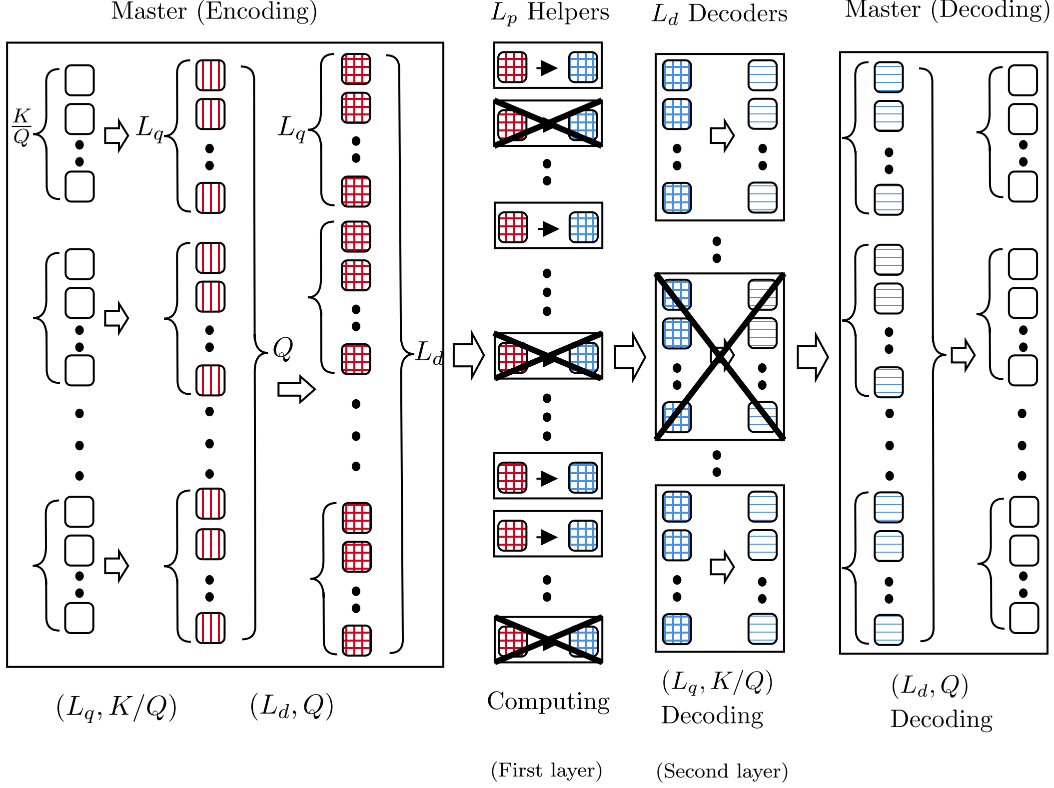

Figure 4.3: The two-layer coding structure to add reliability for both processing and decoding layer

Example 1 *With* $K = M = 1000$*,* $Q = 20$*,* $L_q = 55$*,* $L_d = 21$*, there will be* $L_p = 1155$ *processing helpers for multiplication tasks and* 21 *helpers for the first stage of decoding. The decoding helpers are decoding a (55, 50) code. The master will be required to decode a (21,20) code. Such a code can tolerate one straggling node among the* 21 *decoding nodes.*

In this structure, there are many parameters that must be chosen. For example, L_p, L_d, Q, and more. These parameters directly affect the system performance in terms of its processing time. For example, a large Q means a high decoding complexity at the master. Also, if L_p is too close to K we may experience a longer tail in the processing stage. These parameters have to be chosen for maximizing performance in view of the available helpers and their quality.

The ultimate goal is to select the parameters in order to minimize $\mathbb{E}[\tau_{\text{ex}}]$ out

of all possible setups. In this problem, some limitations exist. The parameters N and P (which define the computation size) are given, moreover, the pool of available helpers $\mathcal{H}$ is limited. Therefore, the total number of helpers

$$|\mathcal{H}| = \sum_i n_i = L_p + L_d$$

is constant. So, although L_d is a degree of freedom, L_p becomes determined once L_d is selected.

The other degree of freedom is the selection of helpers based on their quality for different tasks. One set of helpers denoted by $\mathcal{S}_p$, are the helpers in the processing layer and the other one is $\mathcal{S}_d$, the set of helpers performing decoding. Note that $\{\mathcal{S}_p, \mathcal{S}_d\}$ is a partition of $\mathcal{H}$ and $|\mathcal{S}_p| = L_p$, and $|\mathcal{S}_d| = L_d$. Please note that if $\mathcal{S}_d$ is decided, so is $\mathcal{S}_p$, and therefore, L_d and L_p. In other words, the only design parameters are the code sizes ($\frac{K}{Q}$ and Q) and the choice of helpers involved in decoding ($\mathcal{S}_d$). In short, our goal is to minimize the execution time over valid choices of K, Q and $\mathcal{S}_d$:

$$\min_{K,Q,\mathcal{S}_d} \mathbb{E}[\tau_{\text{ex}}] = \mathbb{E}[\tau_{\text{p}}] + \mathbb{E}[\tau_{\text{dec}}] + \tau_{\text{mas}}. \tag{4.5}$$

Moreover, Our two codes are of size $(|\mathcal{S}_d|, Q)$, $(\frac{|\mathcal{S}_p|}{|\mathcal{S}_d|}, \frac{K}{Q})$. Thus, assuming predefined overheads $\varepsilon_p, \varepsilon_d$, the choice of Q and K are also forced by $|\mathcal{S}_d|$ and $|\mathcal{S}_p|$ as follows,

$$\begin{aligned} |\mathcal{S}_d| &= Q\,(1+\varepsilon_d), \\ |\mathcal{S}_p| &= K\,\frac{|\mathcal{S}_d|}{Q}\,(1+\varepsilon_p). \end{aligned} \tag{4.6}$$

In distributed computing, the coding overhead is typically chosen to be between 5% to 10% because this level of overhead resolves the straggling effect [23]. In other words, $\frac{L_p}{K}$ and $\frac{L_d}{Q}$ that represent the codes' overheads can be preselected. This associates K and Q with $L_d = |\mathcal{S}_d|$, and thus, our optimization problem will simplify to

$$\min_{\mathcal{S}_d} \mathbb{E}[\tau_{\text{ex}}] = \mathbb{E}[\tau_{\text{p}}] + \mathbb{E}[\tau_{\text{dec}}] + \tau_{\text{mas}}. \tag{4.7}$$

To solve this optimization, we must find the best subset $\mathcal{S}_d$. This optimization problem is not very complex since $\mathcal{H}$ typically consists of only a few

different types of helpers. Let us represent the network of available helpers by

$$\mathcal{H} = [(n_1, \mu_1),\, (n_2, \mu_2),\, \ldots,\, (n_i, \mu_i),\, \ldots,\, (n_p, \mu_p)].$$

This means that there are n_i helpers from type i with the straggling parameter of μ_i. Therefore, there are exactly $\prod_i (n_i + 1)$ unique subsets. This is the search size for finding the minimum $\mathbb{E}[\tau_{\text{ex}}]$. However, in the following discussion, we offer a strategy that significantly reduces the complexity of this optimization.

4.3.3 Greedy decoding helper allocation

A good helper allocation is one that utilizes the heterogeneity in order to reduce the runtime. The greedy decoding helper allocation (GDHA) strategy suggests that the most reliable helpers must be assigned to the decoding layer and the rest to be assigned to the processing layer. By obeying this strategy, the search for the best $\mathcal{S}_d$ is simplified to search for the best size of $\mathcal{S}_d$. As soon as the size is known, $\mathcal{S}_d$ will be determined from the GDHA strategy.

The idea behind this strategy is that the high-quality helpers have a more positive impact on execution time when in the decoding layer rather than in the processing layer. This is because the low-quality helpers that are used at the processing layer highly influence the processing time $\mathbb{E}[\tau_{\text{p}}]$, and a few high-quality helpers cannot improve $\mathbb{E}[\tau_{\text{p}}]$. On the other hand, as we will see, the number of helpers needed for the decoding layer is typically small, and therefore, in many cases, it is possible to perform the decoding using only the high-quality helpers, meaning that the decoding process is significantly impacted by this greedy allocation. The above qualitative discussion is presented in a quantitative way in the following. In practical settings, the type of helpers with the least quality is the majority. Therefore, it is reasonable to assume that $n_v > L_p - K$. This will result in

$$\frac{n_v}{L_d} > \frac{L_p - K}{L_d} = L_q - \frac{K}{L_d} > L_q - \frac{K}{Q}. \tag{4.8}$$

This means that the number of type v helpers in most batches is greater than the redundant nodes, i.e., $L_q - \frac{K}{Q}$. Therefore, the completion of the processing computation is limited to the low-quality helpers and thus, $\mathbb{E}[\tau_{\text{p}}]$ is very close

to $\mathbb{E}[\tau_v]$. Therefore, assigning the better helpers to $\mathcal{S}_p$ has a negligible improvement on $\mathbb{E}[\tau_{\text{ex}}]$. On the other hand, $\mathbb{E}[\tau_{\text{dec}}]$ is only dependant on the quality of L_d helpers. As we will see, the optimum $|\mathcal{S}_d|$ is usually small enough that can exclude most of the low-quality helpers. Therefore, assigning the best helpers to this set can actually reduce $\mathbb{E}[\tau_{\text{dec}}]$, and thus, $\mathbb{E}[\tau_{\text{ex}}]$.

This algorithm might not achieve the optimal solutions when the processing load for the helpers in the first layer is significantly higher than the decoding helper's load. Then, even a small improvement in $\mathbb{E}[\tau_{\text{p}}]$ would be favorable as $\mathbb{E}[\tau_{\text{p}}] \gg \mathbb{E}[\tau_{\text{dec}}]$. Please note that $\mathbb{E}[\tau_{\text{p}}] \gg \mathbb{E}[\tau_{\text{dec}}]$ indicates that the processing time of the helpers is too long, meaning that we do not have enough helpers in the system. In other words, the conditions under which GDHA approach deviates from the optimal solution are impractical cases.

Remark 10 *For a given $|\mathcal{S}_d|$, GDHA suggests the best partitioning as $\mathcal{S}_d = [(n_1, \mu_1), \ldots, (L_u, \mu_u)]$ and $\mathcal{S}_p = [(n_u - L_u, \mu_u), \ldots, (n_v, \mu_v)]$, where u is found based on*

$$\sum_{i=1}^{u-1} n_i < |\mathcal{S}_d| \leq \sum_{i=1}^{u} n_i. \tag{4.9}$$

Remark 10 suggests that for the best performance, after grouping helpers into two groups $\mathcal{S}_p$ and $\mathcal{S}_d$, for any helper in $\mathcal{S}_p$ with μ_i and any helper in $\mathcal{S}_d$ with μ_j, we can be sure that $\mu_i \leq \mu_j$.

Using Remark 10, our search for optimal $\mathcal{S}_d$ will reduce to the size of $|\mathcal{H}|$. In reality, the search size is even much smaller than that. This is because the length of the first code is $\frac{K}{Q} \gg 1$. Thus, we can also say $\frac{L_P}{L_d} \gg 1$, which means that $|\mathcal{S}_d| \ll |\mathcal{S}_p|$ and thus, $|\mathcal{S}_d| \ll |\mathcal{H}|$. This means, the search for the optimal $|\mathcal{S}_d|$ can start from zero and will end soon. Note that when $|\mathcal{S}_d| = 0$ it means the two-layer solution is not optimal and we are back to the one-layer traditional solution. In other words, our search includes the existing one-layer solution as a special case.

At the heart of this optimization, there is a trade-off. On the one hand, the smaller Q is, the decoding complexity at the master is reduced, but the decoding complexity at the decoding helpers intensifies. In contrast, larger Q

means more reliability and less complexity in the decoding layer, resulting in smaller $\mathbb{E}[\tau_{\text{dec}}]$. In conclusion, and as supported by simulation results in the next section, the extremes in the choice of L_d are not typically optimal.

4.4 Numerical results

In this section, we will consider several system setups. One objective is to verify the optimality of the solution provided by Remark 10. The second goal is to find the optimum $|\mathcal{S}_d|$ in each study case and analyze the performance of the system. Furthermore, we will observe that how changing some parameters affects the optimum solution and its performance.

We consider the allocation of helpers to the decoding and processing layers, $\mathcal{S}_d$ and $\mathcal{S}_p$, based on both Remark 10, and searching all the possible partitionings. Assuming a preselected code overhead, K and Q can be found based on (4.6). From these selections, the load of each processing helper, i.e., $\frac{NP}{K}$ is calculated. Similarly, the decoding load of the decoding helpers and the master's load are calculated based on the Reed-Solomon decoding complexity stated in Section 4.2.

For our first setup, assume the size of matrix multiplication is given as $P = 10^3$ and $N = 10^6$. Also, assume $\mathcal{H} = [(10, 9), (40, 4), (1000, 1)]$. By changing $|\mathcal{S}_d|$, we have studied the performance of the system. As demonstrated in Fig. 4.4, the optimal $|\mathcal{S}_d| = 13$ is the one that achieves the minimum point on the $\mathbb{E}[\tau_{\text{ex}}]$ curve. We observe that almost always the partitioning provided by Remark 10 aligns with the optimum partitioning found by a complete search. The only cases that GDHA is not optimal are when the size of $\mathcal{S}_d$ is much larger than the optimal $|\mathcal{S}_d|$, which is clearly not an attractive regime. Even in these cases, the gap between the complete search and GDHA is negligible.

Note that if $|\mathcal{S}_d|$ is too small, the decoding helpers will have a high computation load, and $\mathbb{E}[\tau_{\text{dec}}]$ would be very large. Consequently, the single-layer model is not an attractive solution in this setup. Moreover, as discussed in Section 4.3.3 and supported by this figure, we can see that with a large number of decoding helpers $|\mathcal{S}_d|$, the master's completion time becomes too large,

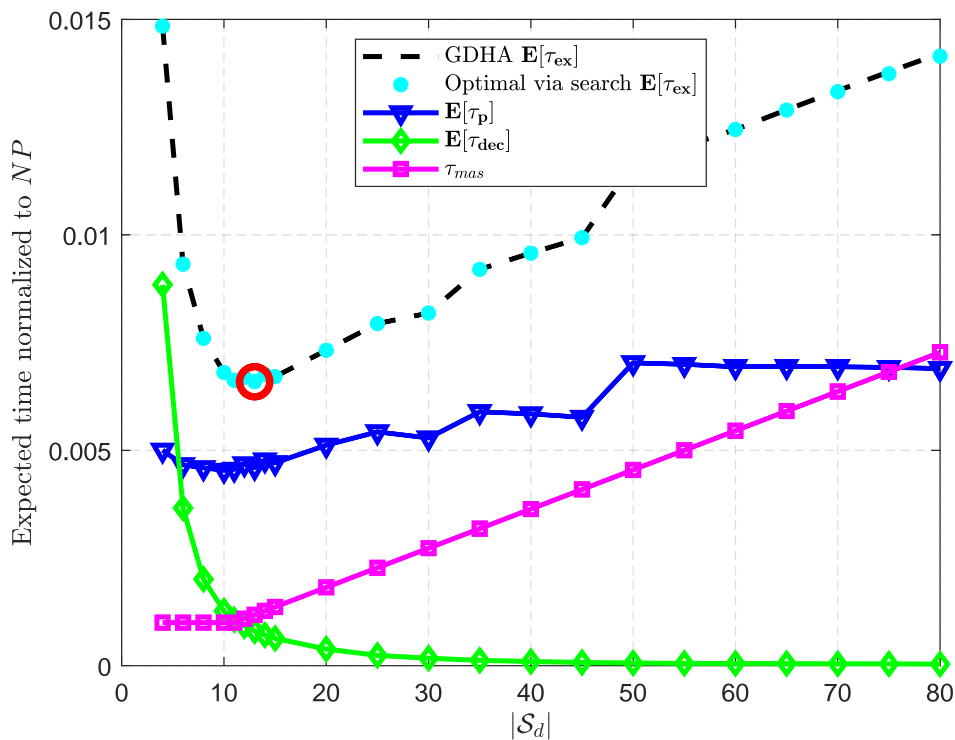

Figure 4.4: The behavior of $\mathbb{E}[\tau_x]$ and its components

resulting in a much longer total execution time. While more decoding helpers means shorter decoding time by the decoding helpers, the second code to be decoded by the master increases in size, making the master decoding the bottleneck. Hence, the extreme cases for $|\mathcal{S}_d|$ are not optimal and the existing trade-off defines the optimal $|\mathcal{S}_d|$ somewhere in the middle. The expected time of each stage of the distributed computing is also shown in Fig 4.4. The trade-off between τ_{mas} and $\mathbb{E}[\tau_{\text{dec}}]$ can be easily seen.

In Fig. 4.5, we have compared our proposed model with a traditional single-layer scheme. We can see the runtime reduction as a result of the proposed multi-layer decoding scheme. In this figure, we still assume $\mathcal{H} = [(10, 9), (40, 4), (1000, 1)]$, but we allow algorithms not to employ all the available helpers. For example, when $|\mathcal{H}| = 600$, we mean $\mathcal{H} = [(10, 9), (40, 4), (550, 1)]$. An interesting observation about the single-layer solution is that for a large number of helper nodes, the runtime increases with the number of employed helpers. When a smaller number of helpers are used, although the compu-

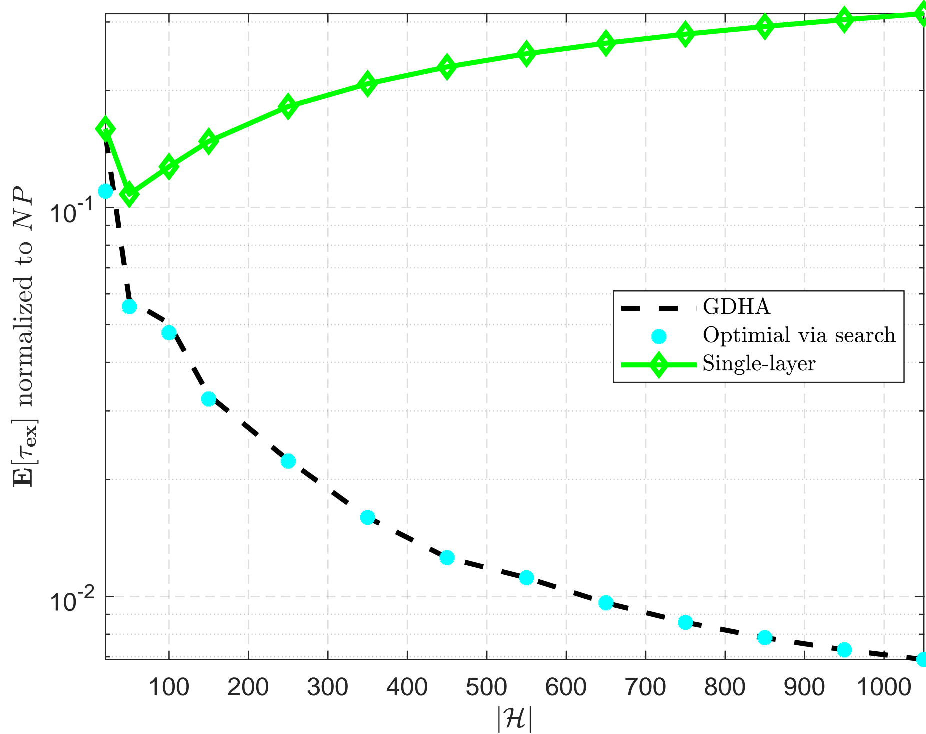

Figure 4.5: A comparison between our proposed multi-layer method and single-layer method when a different number of helpers from the network are utilized

tational load of each helper is increased, the decoding load of the master is decreased because the master is using a shorter code and thus experiences a reduced decoding complexity. This means the best choice of the single-layer solution is to not use all the available helper nodes. Despite this, the conclusion of this figure is that the optimal solution based on GDHA (achieved when using all 1050 helpers) is much better than the optimal single-layer solution (achieved when using 50 helpers) [2].

We have also studied the effect of $|\mathcal{S}_d|$ on a number of methods in Fig. 4.6. These methods are GDHA, reverse GDHA, and random allocation. Reverse GDHA allocates the most reliable helpers to the processing layer. Also, for comparison, the result of the single-layer solution is reported, where, as discussed in Fig. 4.5, not all the helpers are employed to achieve the best runtime

[2]Note that our scheme's curve will also reach a minimum point. That is because by having a lot of helpers, the sizes of codes increase again and become problematic. For the setup in Fig. 4.5, the minimum point is at around 2000 helpers.

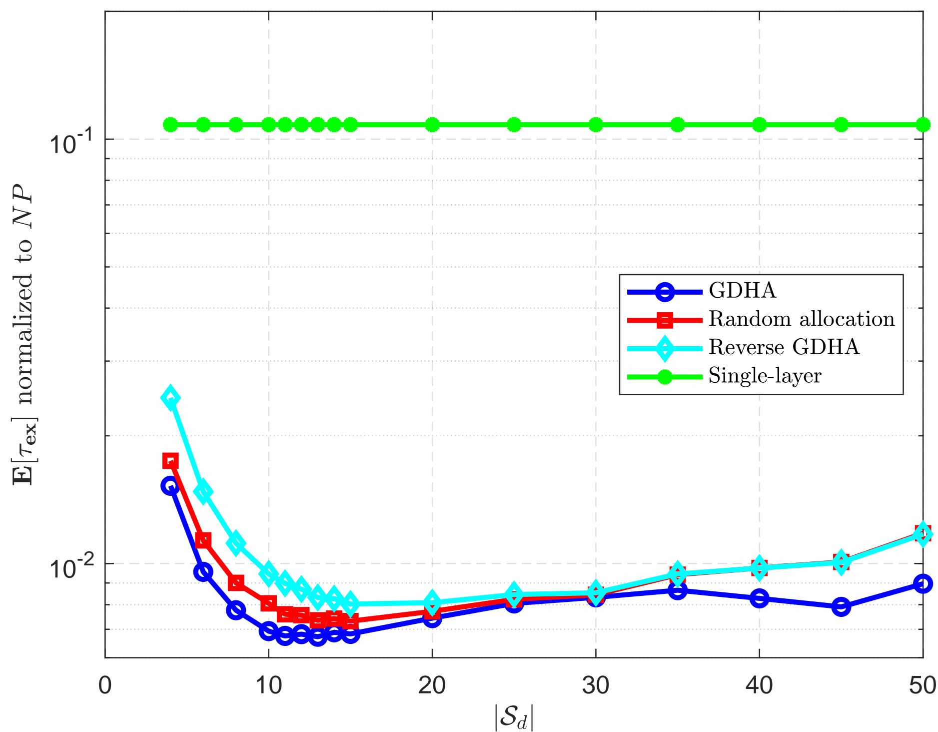

Figure 4.6: The behavior of different distributed computing allocations based on $|\mathcal{S}_d|$

possible. It can be seen that GDHA allocation can improve the minimum $\mathbb{E}[\tau_{\text{ex}}]$ by almost 16 times compared to the single-layer solution and by 10% compared to the random two-layer allocation.

In Fig. 4.4, the values for all three stages of distributed computing were in the same order. However, in cases where the number of helpers is insufficient, i.e., $\mathbb{E}[\tau_{\text{p}}]$ is dominant, or the cases in which the decoding complexity is the major bottleneck, the choice of $|\mathcal{S}_d|$ optimization and the behavior of $\mathbb{E}[\tau_{\text{ex}}]$ may differ. In Fig. 4.7, three scenarios are compared. We can see that the optimal $|\mathcal{S}_d|$ can be very different for the same network based on the original computation size. We can see that when both N and P increase, the optimum $|\mathcal{S}_d|$ increases, as well.

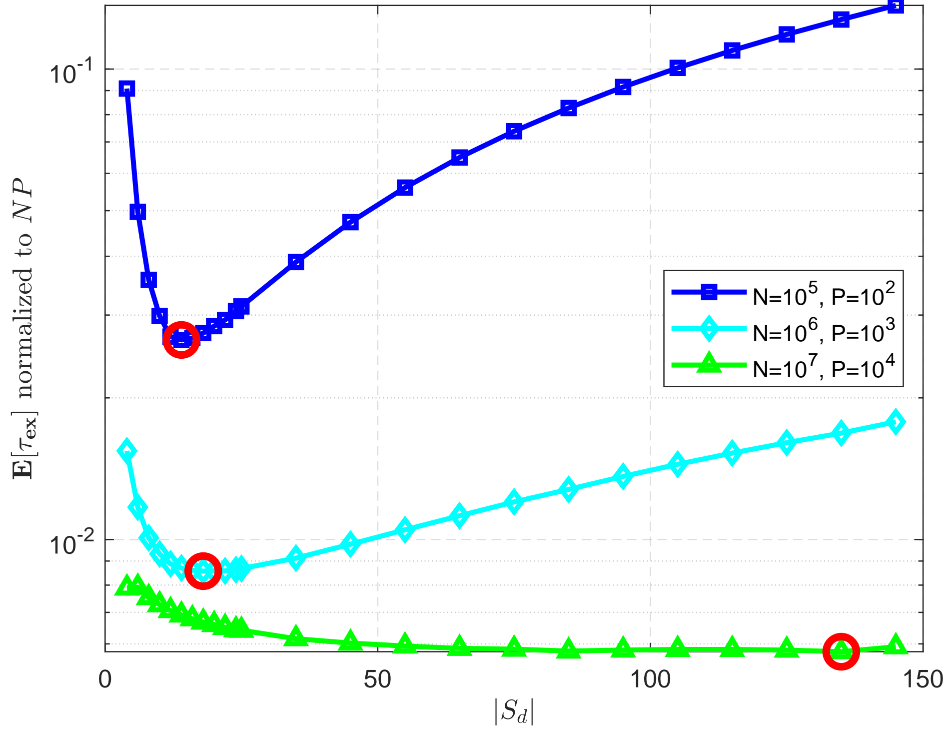

Figure 4.7: The effect of different sizes of computations on the optimum $|\mathcal{S}_d|$

4.5 Conclusion and future work

In this chapter, we focused on the high decoding complexity of the master node in coded distributed computing schemes. We proposed a multi-layer coding structure which allows the helpers to engage in decoding, thus, reducing the master's decoding load. Then the question of how to allocate helpers to processing or decoding was discussed. A simple helper allocation strategy was proposed and shown to be the optimal allocation. The numerical results supported the analytical discussions.

The proposed multi-layer coded solution for distributed computing problems can be very promising. MapReduce schemes with multiple Shuffle stages [41] can also benefit from this method in order to phase out the master's engagement in the Shuffle stages. Moreover, scheduling methods for heterogeneous distributed computing problems carry many opportunities for further research [46]. In this work, the processing load was distributed uniformly, and

it was shown that high-quality helpers are better to be assigned to the decoding task. It can be investigated whether non-uniform load allocations might offer other advantages.

Chapter 5

Conclusions and future work

5.1 Summary of contributions and results

In this work, we considered large-scale coded distributed computing systems. We investigated the effect of the added computational load to the master, due to coding. It was shown that for many practical setups, this newly introduced load was extensively high. Therefore, for large-scale coded scenarios, an alternative solution was needed.

The main idea of this thesis was to explore possibilities that are introduced by inserting an additional layer of helpers to the structure of the system. The goal was to have some helpers directly assist the master with its remaining computational load and decoding load. Such a multi-layer solution, however, induced some new challenges, challenges that become more severe as the size of the network increases.

To have a multi-layer coded solution, each layer of the computation must be coded itself. Applying multiple layers of coding to a system with master-free shuffling, however, did not have a straightforward solution. This thesis proposed solutions for these multi-layer coded distributed computing problems.

In Chapter 3, particularly, we formed our multi-layer structure based on FFT structure. In order to have a fully fault-tolerant design, a coding design at the Shuffle stage was proposed which fitted the FFT computation. Moreover, by benefitting from low-complexity codes, we further reduced the decoding load and showed that the overall system performance and cost-saving were extremely improved.

In Chapter 4, we focused on a coding structure in which two layers of coding were applied over each other. Thus, we were able to introduce some decoding helpers that carried the main load of decoding complexity and left a very small load for the master. The key challenge was to design the decoding layer also reliable. Considering a heterogeneous network, we proposed a strategy that achieved the minimum run-time and thus, the best performance.

In both studies, considering a number of scenarios and system setups, we performed various numerical analysis. The numerical results supported our algorithms, and it was shown that for practical setups, they outperform the existing solutions.

5.2 Future research directions

In our setups, we assumed that the size of helpers' input data is more or less the same. In heterogeneous networks, the helpers' memory capacity may also vary from node to node. Therefore, allocating different data sizes to helpers in the same layer can open up many opportunities. For instance, as discussed in Section 4.5, one could investigate whether non-uniform load allocation schedulings could result in a better performance in a heterogeneous network. Optimizing the solution for such a scenario, however, accompanies many challenges.

Moreover, including the helpers' cost could also add another factor of complexity to the optimization problem. In that case, employing all the helpers available might not offer the best solution. In both this and the previous scenario (i.e., varying memory capacity across nodes), because of the added degrees of freedom, finding the optimum design requires a massive search through all the possibilities. Finding a general solution even under some practical assumptions would be incredibly attractive.

Another possible future work is to explore the conjunction of two proposed coding structures. The necessary shuffling in distributed FFT does not let the multi-layer encoding proposed in Chapter 4 to be directly applied. Thus, coding in the Shuffle stage was required. However, there are distributed computing

systems that require multiple layers of computations, too, even though their data shuffling is not necessarily the same as FFT (e.g., not all the nodes share data with each other). In those applications, a combination of the proposed coding methods could be developed that achieves a better performance.

Finally, because of our multi-layer design, employing fountain codes as rateless codes was not applicable. Designing Raptor codes with a fixed rate is another interesting area of work. Although currently the need for a fixed-rate low-complexity code does not go beyond distributed computing problems, they may be proved helpful in other coded systems, in the future.

References

[1] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in *Proceedings of COMPSTAT’2010*, Springer, 2010, pp. 177–186.

[2] Y. Yan and L. Huang, “Large-scale image processing research cloud,” *Cloud Computing*, pp. 88–93, 2014.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets.,” *HotCloud*, vol. 10, no. 10, p. 95, 2010.

[4] 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.

[5] B. Varghese and R. Buyya, “Next generation cloud computing: New trends and research directions,” *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.

[6] B. P. Rimal, E. Choi, and I. Lumb, “A taxonomy and survey of cloud computing systems,” in *2009 Fifth International Joint Conference on INC, IMS and IDC*, IEEE, 2009, pp. 44–51.

[7] Y. Yu, P. K. Gunda, and M. Isard, “Distributed aggregation for data-parallel computing: Interfaces and implementations,” in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 247–260.

[8] A. Norton and A. J. Silberger, “Parallelization and performance analysis of the Cooley-Tukey FFT algorithm for shared-memory architectures,” *IEEE Transactions on Computers*, vol. 36, no. 5, pp. 581–591, 1987.

[9] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, “Speeding up distributed machine learning using codes,” *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2017.

[10] 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)*, IEEE, 2018, pp. 857–866.

[11] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *2016 IEEE Globecom Workshops (GC Wkshps)*, IEEE, 2016, pp. 1–6.

[12] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded Fourier transform," in *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2017, pp. 494–501.

[13] 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*, 2016, pp. 2100–2108.

[14] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[15] J. Dean, "Experiences with MapReduce, an abstraction for large-scale computation," *Proceedings of 15th International Conference on Parallel Architectures and Compilation Techniques*, vol. 6, p. 1, 2006.

[16] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce," in *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2015, pp. 964–971.

[17] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments.," in *Osdi*, vol. 8, 2008, p. 7.

[18] F. Li, J. Chen, and Z. Wang, "Wireless MapReduce distributed computing," *IEEE Transactions on Information Theory*, vol. 65, no. 10, pp. 6101–6114, 2019.

[19] M. Ji and R.-R. Chen, "Fundamental limits of wireless distributed computing networks," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, IEEE, 2018, pp. 2600–2608.

[20] 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*, vol. 64, no. 1, pp. 109–128, 2017.

[21] A. Reisizadeh and R. Pedarsani, "Latency analysis of coded computation schemes over wireless networks," in *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2017, pp. 1256–1263.

[22] S. Dhakal, S. Prakash, Y. Yona, S. Talwar, and N. Himayat, "Coded computing for distributed machine learning in wireless edge network," in *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, IEEE, 2019, pp. 1–6.

[23] 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)*, 2013, pp. 185–198.

[24] S. Kiani, N. Ferdinand, and S. C. Draper, "Exploitation of stragglers in coded computation," in *2018 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2018, pp. 1988–1992.

[25] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[26] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, IEEE, 2010, pp. 1–10.

[27] Y. Sun, J. Zhao, S. Zhou, and D. Gündüz, "Heterogeneous coded computation across heterogeneous workers," *arXiv preprint arXiv:1904.07490*, 2019.

[28] 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*, ACM, 2012, pp. 13–16.

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

[30] R. Singleton, "Maximum distance q-nary codes," *IEEE Transactions on Information Theory*, vol. 10, no. 2, pp. 116–118, 1964.

[31] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems*, 2017, pp. 4403–4413.

[32] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *2017 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2017, pp. 2418–2422.

[33] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the society for industrial and applied mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[34] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Transactions on Information Theory*, 2019.

[35] K. S. Kedlaya and C. Umans, "Fast polynomial factorization and modular composition," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1767–1802, 2011.

[36] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Distributed fountain codes for networked storage," in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, IEEE, vol. 5, 2006, pp. V–V.

[37] A. Shokrollahi and M. Luby, "Raptor codes," *Foundations and trends® in communications and information theory*, vol. 6, no. 3–4, pp. 213–322, 2011.

[38] M. Luby, "LT codes," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science, Proceedings.*, IEEE, 2002, pp. 271–280.

[39] F. Bensaali, A. Amira, and A. Bouridane, "Accelerating matrix product on reconfigurable hardware for image processing applications," *IEE proceedings-Circuits, Devices and Systems*, vol. 152, no. 3, pp. 236–246, 2005.

[40] A. Severinson, A. G. i Amat, and E. Rosnes, "Block-diagonal and LT codes for distributed computing with straggling servers," *IEEE Transactions on Communications*, vol. 67, no. 3, pp. 1739–1753, 2018.

[41] H. Jeong, T. M. Low, and P. Grover, "Coded FFT and its communication overhead," *arXiv preprint arXiv:1805.09891*, 2018.

[42] ——, "Masterless coded computing: A fully-distributed coded FFT algorithm," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2018, pp. 887–894.

[43] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *International Conference on Machine Learning*, 2017, pp. 3368–3376.

[44] N. Ferdinand, B. Gharachorloo, and S. C. Draper, "Anytime exploitation of stragglers in synchronous stochastic gradient descent," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, IEEE, 2017, pp. 141–146.

[45] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "A scalable framework for wireless distributed computing," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2643–2654, 2017.

[46] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Transactions on Information Theory*, 2019.

[47] 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.

[48] B. Bartan and M. Pilanci, "Polar coded distributed matrix multiplication," *arXiv preprint arXiv:1901.06811*, 2019.

[49] N. Ferdinand and S. C. Draper, "Hierarchical coded computation," in *2018 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2018, pp. 1620–1624.

[50] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," *arXiv preprint arXiv:1802.03475*, 2018.

[51] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," *arXiv preprint arXiv:1806.00939*, 2018.

[52] W.-T. Chang and R. Tandon, "On the capacity of secure distributed matrix multiplication," in *2018 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2018, pp. 1–6.

[53] N. Raviv and D. A. Karpuk, "Private polynomial computation from lagrange encoding," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 553–563, 2019.

[54] K. R. Rao, D. N. Kim, and J. J. Hwang, *Fast Fourier transform-algorithms and applications*. Springer Science & Business Media, 2011.

[55] A. C. Gilbert, P. Indyk, M. Iwen, and L. Schmidt, "Recent developments in the sparse Fourier transform: A compressed Fourier transform for big data," *IEEE Signal Processing Magazine*, vol. 31, no. 5, pp. 91–100, 2014.

[56] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[57] A. Sandryhaila and J. M. Moura, "Big data analysis with signal processing on graphs," *IEEE Signal Processing Magazine*, vol. 31, no. 5, pp. 80–90, 2014.

[58] J. W. Wong, C. Durante, and H. M. Cartwright, "Application of fast Fourier transform cross-correlation for the alignment of large chromatographic and spectral datasets," *Analytical Chemistry*, vol. 77, no. 17, pp. 5655–5661, 2005.

[59] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013, pp. 69–84.

[60] V. Jalaparti, P. Bodik, S. Kandula, I. Menache, M. Rybalkin, and C. Yan, "Speeding up distributed request-response workflows," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 219–230, 2013.

[61] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," in *Proceedings of the 2017 Symposium on Cloud Computing*, ACM, 2017, pp. 445–451.

[62] A. Gupta and V. Kumar, "The scalability of FFT on parallel computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 8, pp. 922–932, 1993.

[63] M. Pippig, "PFFT: An extension of FFTW to massively parallel architectures," *SIAM Journal on Scientific Computing*, vol. 35, no. 3, pp. C213–C236, 2013.

[64] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, p. 63, 2015.

[65] A. Shokrollahi, "Raptor codes," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. SI, pp. 2551–2567, 2006.

[66] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder, "RaptorQ forward error correction scheme for object delivery," Tech. Rep., 2011.

[67] F. Lázaro, G. Liva, and G. Bauch, "Inactivation decoding of LT and Raptor codes: Analysis and code design," *IEEE Transactions on Communications*, vol. 65, no. 10, pp. 4114–4127, 2017.

[68] F. L. Blasco, G. Liva, and G. Bauch, "Enhancing the LT component of Raptor codes for inactivation decoding," in *SCC 2015; 10th International ITG Conference on Systems, Communications and Coding*, VDE, 2015, pp. 1–6.

[69] "Technical specification group services and system aspects; multimedia broadcast/multicast service; protocols and codecs," 3GPP, TS 26.346, Jun. 2012, V11.1.0.