On Coding Schemes for Straggler Mitigation in Distributed Computing

by

Muhammad Fetrat Qharabagh

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

Abstract

One of the main challenges in distributed computing are straggling servers i.e. servers that have a temporary or permanent delay in sending the result of the computations assigned to them. One of the proposed methods to address the straggler problem is to use forward error correction codes to encode the subtasks before distributing them among workers where the results of the lagging servers are treated as erasures. This method is referred to as coded distributed computing (CDC). In this thesis we study CDC in general and address two straggler related issues.

First we consider a master-worker setting to perform a matrix-vector multiplication. The main focus in prior CDC approaches for this problem is on minimising the computation time using a family of codes known as minimum distance separable (MDS) codes. The assumption is that the encoding and decoding times in the master are negligible. However, when the task is large or the number of workers is high, and the encoding/decoding can not be done offline, this time might no longer be small compared to the computation time. Hence, we introduce a new family of binary locally repairable codes (BLRCs) specifically designed for CDC. Being binary removes the costly multiplication operations in the encoding/decoding process and the locally repairable nature of the code further reduces the decoding complexity. Therefore, due to the lower encoding/decoding complexity in our codes, the overall time delay is lower than the conventional MDS scheme. Second, we consider a general master-worker setting where each worker is assigned with multiple subtasks.

Accordingly, we propose a CDC scheme based on the systematic MDS codes. Then we derive and calculate the analytical probabilities of finishing the coded distributed computation before a deadline in our scheme and obtain the corresponding optimal subtask loads and the code rate. The overall goal is to reduce the computation time compared to conventional schemes for multi-subtask-per-worker distributed computing.

Preface

The work in Chapter 3 are under review for possible publication in IEEE Transactions on Communications under the title “A Family of Binary Locally Rapairable Codes for Coded Distributed Computing”. Additionally, we intend to submit the contributions in Chapter 4 for possible publication in IEEE Communications Letters under the title “A New Coded MMC Distributed Computing Scheme: Analysis and Optimization”. The main contributor to both of the works is the author of this thesis.

To my family

for their love and never ending support

A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

– Leslie Lamport

Acknowledgements

First, I want to sincerely thank my supervisor Prof. Masoud Ardakani for his unwavering support and providing me with a vast amount of knowledge to complete my degree. I would like to also thank my sister for her valuable insights that helped improve this thesis.

Besides my supervisor, I would like to thank my thesis committee memebers Dr. Behrad Gholipour, Dr. Bruce Cockburn, and Dr. Hai Jiang for dedicating their time to this thesis.

Last but not the least, I want to express my deep gratitude for the love my family have for me and the support they provide to me. Without their efforts I could not continue my journey. Also, I want to thank all my friends for their kind support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In today's data-driven world, the demand for processing capacity is rising rapidly. The ever-increasing size of datasets and the emergence of new applications, like machine learning and big data analysis are key factors driving this exponential growth.

Machine learning has revolutionized numerous industries, including finance, healthcare, marketing, and manufacturing [52]. Its ability to uncover patterns, extract insights, and make accurate predictions relies heavily on processing power. Machine learning algorithms require substantial computational resources to process vast datasets, conduct complex calculations, and optimize models. As organizations increasingly embrace machine learning to enhance decision-making, automate processes, and improve customer experiences, the demand for processing capacity will only continue to surge.

Similarly, the advent of big data has ushered in a new era of information processing. With the proliferation of Internet-connected devices, social media platforms, and sensor technologies, organizations are generating colossal amounts of data on a daily basis. Big data analysis aims to extract valuable insights and actionable intelligence from these massive datasets. However, the processing of such large volumes of data necessitates significant computational power. Analyzing complex data structures, running advanced data analytics algorithms, and performing real-time processing on massive datasets require robust processing capacity.

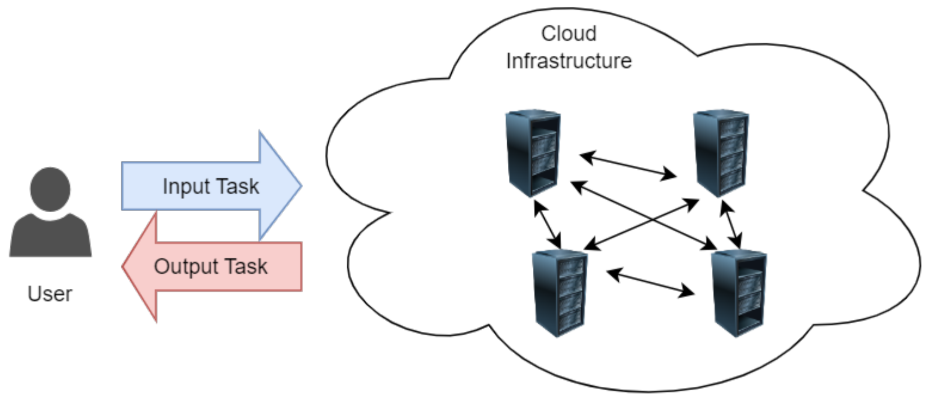


Figure 1.1: Cloud Computing

Handling vast amounts of data and computations using single processors presents significant challenges. First, single processors, whether CPUs or GPUs, have finite processing power and can only execute a limited number of instructions per unit of time. As data sizes increase, the time required to process them becomes impractical, resulting in slow and inefficient computations. Moreover, the memory capacity of single processors is limited, constraining the amount of data that can be stored and processed at any given time. Additionally, single processors lack fault tolerance and redundancy. In case of a hardware failure or system crash, all data and computations are at risk of being lost or corrupted.

As a result, large-scale computing has gained a lot of attention, and solutions like cloud computing are becoming increasingly popular. Cloud computing follows a basic procedure where users provision and access computing resources, managed by a cloud service provider, through the Internet. Initially, users request the desired resources, such as virtual machines, storage, or applications, which are then allocated from the provider's infrastructure. The allocated resources are virtualized, allowing for efficient utilization and scalability. Users can remotely access and utilize the provisioned resources, paying only for the actual usage on a pay-as-you-go basis. The cloud provider handles maintenance, security, and updates, ensuring high availability and reliability.

Cloud computing is considered an excellent solution for large-scale computing due to its scalability, flexibility, and cost-effectiveness. It enables users to effortlessly scale their resources to handle varying workloads and avoids the need for significant upfront investments in hardware and infrastructure. Additionally, it offers high availability, fault tolerance, and robust security measures, making it an attractive option for organizations seeking efficient and reliable computing solutions at scale.

On the other hand, cloud computing often utilizes distributed computing principles to achieve its goals. Cloud service providers rely on distributed computing architectures to manage and allocate resources across their vast networks of servers. They use distributed computing techniques to distribute workloads, balance the load, and ensure high availability and fault tolerance. In fact, it is by leveraging distributed computing that cloud providers can offer scalable and resilient infrastructure to their users. Hence, it is very important to develop efficient, reliable, and secure distributed computing methods.

1.2 Distributed Computing

As mentioned in the previous section, when the amount of computation is very large, or there is a tight deadline to finish a task, a single processor may not be able to handle it. In such cases, multiple processors may be exploited to finish the task. Two of the common approaches are parallel computing and distributed computing. In both approaches, a single problem is divided into smaller sub-problems, and each subproblem is assigned to a separate processing unit. In the end, the results are gathered and the main task is recovered. Although there is no clear dividing line between parallel and distributed computing, there are some distinctions between them.

In parallel computing, processors are usually located at a close distance from each other and have been designed to jointly execute computational tasks. In addition, the communication between processors is reliable and predictable. For distributed computing, on the other hand, processors may be far apart and inter-processor communication can become a significant bottleneck. In other words, communication delays may be unpredictable and the communication links may be unreliable. Moreover, the topology of the system may change during the computation due to failures or repairs in communication links and the removal or addition of processors.

In spite of the challenges, distributed computing offers several potential advantages. First, the nodes in a distributed computing system can be geographically dispersed and may vary in terms of hardware specifications, operating systems, and software configurations. It also offers scalability. As the workload or data size increases, additional nodes can be added to the system to handle the increased demand. Furthermore, distributed computing provides fault tolerance and resilience. In case of a hardware failure or system crash on a particular node, the distributed system can continue functioning by redistributing the workload to other available nodes.

There are two major models in distributed computing: Master/worker and peer-to-peer (P2P)[29]. In the peer-to-peer distributed computing model, all nodes in the network are considered equal and can act as both clients and servers. They collaborate and share resources directly with each other without a central coordination entity. However, in a master/worker model, a central master node delegates tasks to multiple worker nodes for parallel execution. After data and computational functions are provided to the helpers by the master, each worker node performs its assigned task and reports the results back to the master node. When workers finish their tasks, the master collects the processed tasks and finalizes the computation. Depending on the master/worker scheme deployed, workers can be assigned a single task or multiple smaller ones. They also may only communicate with the master or need to communicate with each other as well. Compared to the P2P model, this model is simple to implement and can be efficient for large-scale problems. However, it can be a performance bottleneck if the master node becomes overloaded. In Chapter 3 we will propose a scheme that addresses such a potential bottleneck.

1.3 Bottlenecks of Distributed Computing

Although distributed computing is an excellent solution for large-scale computing tasks several bottlenecks can limit the overall performance and efficiency of the system. Hence, identifying and addressing these bottlenecks is crucial for optimizing distributed computing systems.

One of the major bottlenecks is the communication bottleneck. Network latency, limited bandwidth, and congestion can impact the speed and efficiency of data transfer and communication between nodes. Excessive data shuffling between compute nodes, high network traffic or sub-optimal network configurations can exacerbate this bottleneck. Security bottleneck is another major concern in distributed computing where vulnerability to eavesdroppers and attackers threatens the overall performance and efficiency of the system.

There are other sources of failures as well but one of the most pressing issues in distributed computing is the straggling bottleneck. The workers that take significantly longer to complete their assigned computations compared to others are called stragglers. Stragglers slow down the overall execution time of the system and can create a bottleneck that affects the efficiency and scalability of distributed computing systems.

Several factors can cause the workers to straggle. For example, distributed computing systems often consist of nodes with varying computing capabilities and if some nodes have lower processing power or more limited resources, this could lead to the straggler phenomena. Similarly, an uneven distribution of data among nodes can lead to stragglers. Communication delays or congestion in the network can be another reason why the timely exchange of data and coordination between nodes is disrupted. Stragglers can also occur due to node failures or hardware issues.

The critical issue in the straggling problem is that even if the computation depends on the computation result of a single straggling worker, the entire process should wait for the result of that worker in order to finalize the overall computation. In fact, it was demonstrated empirically in [77] that this straggler effect can prolong the job execution time by as much as five times.

Therefore, stragglers can have significant implications on distributed computing systems. They can lead to resource underutilization. While some nodes are waiting for straggling tasks to complete, other resources remain idle. They can also hinder scalability. As the system grows in size or handles larger workloads, the presence of stragglers can limit the ability to efficiently scale and distribute tasks across nodes.

To address the problem of stragglers, various techniques can be employed, such as task duplication, speculative execution, load balancing algorithms, and fault tolerance mechanisms. These approaches aim to mitigate the impact of stragglers and improve the overall performance and reliability of distributed computing systems. Some of these techniques, such as task duplication, although they are able to create an acceptable robustness to stragglers, they can encounter issues like excessive overhead. An alternative choice that is able to maintain a balance between fault tolerance and overhead is coded distributed computing.

1.4 Coded Distributed Computing

In order to address major bottlenecks of distributed computing, different concepts from the coding theory have been leveraged [24], [36], [38], [39]. These ideas in the literature are often categorized as Coded Distributed Computing (CDC). In fact, CDC is a broad term that is used to refer to the distributed computing systems that at least in one stage of their computation, use coding theory to address their bandwidth bottlenecks, enhance their security, or provide robustness and fault tolerance in their application.

To address the bandwidth bottleneck, some of the proposed CDC approaches add extra local computations for more network bandwidth [38], [41]. For some design parameter r, the same subtasks are placed in r carefully chosen workers, injecting r times more local computations. The redundant computations in return create local information in the distributed nodes that provide the opportunity for coded multicast. This makes possible transmission of packets during the shuffle phase that are simultaneously useful for r

workers. In other words, such CDC schemes trade r times more computations for an r times gain in local bandwidth consumption.

Coding theory is also applied to help the distributed computing systems to meet the following constraints: (1) Privacy constraints such that sets of colluding workers cannot infer anything about the input dataset in the information-theoretic sense, and (2) security constraints that states the computation must be accomplished successfully even if some workers return purposefully erroneous results. In this regard, methods like the BGW scheme [5] are adopted that apply the Shamir secret sharing scheme [60] to generate coded data shares with security guarantees and compute the function on the coded shares.

Other than the mentioned applications of coding theory, another major problem for which coded distributed computing techniques have been considered is reducing the effect of the stragglers. The main idea is to divide the input data into smaller fractions. Then use forward error correction codes from coding theory to encode these fractions to obtain a larger number of them, introducing some redundancy. The encoded data fractions are then distributed among separate worker nodes. Due to the introduced redundancy during the coding process, now in case some of the workers become stragglers, the remaining workers still have enough information to recover the intended computation result. The amount of redundantly added computation or overhead in this approach is less than the cloning methods and, if selected carefully, even 5% extra helpers can mitigate the straggling effect and reduce the latency [1]. Moreover, this coded approach was shown to significantly outperform the state-of-the-art cloning approaches in straggler mitigation capability, and minimize the overall computation latency. In this thesis, our focus will be on the coded distributed computing methods that are proposed to mitigate the effect of the stragglers. From now on when we use CDC, we refer to this specific category. Later in Chapter 2 we will discuss the working procedure of the CDC in detail.

1.5 Thesis Overview

As discussed in the previous section, coded distributed computing can increase the reliability of the distributed systems. However, it increases the amount of required computation as it adds two pre-processing and post-processing stages. In the pre-processing step, the divided chunks of the data are encoded and in the post-processing, the result of non-straggling workers are decoded to extract the necessary information for the final result.

Note that the main idea in distributed computing is that each worker node spends its resources on the local fraction of the data that it receives. But the pre-processing and the post-processing need to be done on the entire input data or the collection of individual computation results due to the information-theoretic dependencies between them. As a result, these steps must be performed by a central node, i.e. the master node.

In some applications, the encoding of the input data and the decoding of the results are performed offline, meaning that the time required to perform these operations by the master is not a concern. As a result, the current CDC approaches ignore this time and only focus on improving the computation time. Nevertheless, in a large group of applications, where the encoding and the decoding cannot be done offline, the required time for these two stages not only is non-negligible but also can be a significant bottleneck in the overall execution time. Training of large-scale machine learning applications is in this category where due to iterative model updates, there is a constant need for the encoding and decoding of the model parameters and the results.

In this thesis, as our first contribution, we consider distributed matrix-vector multiplication and the potential encoding/decoding bottleneck. Unlike most of the previous work, in addition to the time spent on computation, we consider the time for encoding and decoding as well. In fact, in our work, the overall execution time is the sum of the time in three stages: encoding, computation, and decoding. Then we propose a new code and a new scheme to reduce the overall execution time.

In contrast to the majority of the codes used for CDC, our code is binary. Hence, it removes costly multiplication operations from the encoding and decoding stages. Furthermore, in our scheme, after the input data is divided into smaller parts, these fractions are put into disjoint groups. Then the scheme creates two sets of dependencies in the encoded data. The first set of dependencies is within each group. later in Chapter 3 we will call these local dependencies. The second set of dependencies is the information-theoretic relationship between the data fractions of different groups. Later we will refer to these as global dependencies. This structure further reduces the decoding time since a smaller number of dependent coded blocks of data would be required to recover the result of a specific straggling worker. As a result of our scheme, the master's workload for encoding and decoding drops significantly and is no longer a bottleneck. Additionally, the overall execution time is lower than the conventional scheme.

In our second contribution, we consider a CDC setting in which instead of large tasks, each worker receives multiple smaller ones. These smaller subtasks are executed in order and the results are reported to a master node as soon as any of them is completed. When the input data is divided so that each worker receives only a single task, designing the load of each task and the optimal code rate to minimize the computation time is straightforward. However, for multiple tasks, this is not an easy problem. For instance, one worker might be able to finish all of its subtasks while another worker still struggles to finish its first computation. To this end, we analytically model the multiple-task CDC and derive the optimal load of the subtasks and the code rate, accordingly.

The rest of this thesis is organized as follows: In Chapter 2 we provide some background material on coding theory, distributed computing, and CDC. In Chapter 3 we consider distributed matrix-vector multiplication and introduce our new CDC scheme to reduce the encoding/decoding time and, consequently, the overall execution time. In Chapter 4, we study a multiple-task CDC setting and derive the optimal task loads and code rate for it. Finally, in Chapter 5, we conclude our contributions and recommend future research directions.

Chapter 2

Background Material

2.1 Distributed Computing Systems

A distributed system refers to a collection of autonomous computers or nodes connected through a network that work together to achieve a common goal. In a distributed system, each node operates independently and has its own memory and processing power. The nodes communicate and coordinate with each other by passing messages or sharing data, allowing them to collaborate and perform tasks collectively. Recently distributed storage systems and distributed computing systems have gained a lot of attention [50], [61].

In this thesis, we only study distributed computing systems. In distributed computing systems the main goal is to minimize the computation time of a task. To achieve this goal, one needs to find a distributed computing scheme that suits the requirements of the problem the best. Some attributes of the DCS play an important role in finding the optimal scheme. These include but are not limited to the number of computing nodes, their memory, processing power, and communication links.

The key idea in distributed computing is to divide the main task into smaller subtasks, assigning them to multiple distributed processors, and aggregating the processed data to finalize the main computation task. Usually, a central node, which is called the master, receives the main computation task and executes it by employing other computing nodes in the network, called workers. The master collects the individual results and delivers the finalized computation. In most cases, the master is also responsible for doing the required pre-processing and post-processing of the data. This structure is usually known as the master-worker or master-helper model. Although there are distributed models with multiple masters [65] or without any master, such as peer-to-peer models [67], the common model in practice and the literature for distributed computing is the master-worker model due to its efficiency and low-complexity implementation.

2.1.1 System Model

One of the models that can be implemented in the form of a master-worker setting is MapReduce [12]. The MapReduce framework is designed for processing large-scale data sets in parallel across a distributed cluster of computers. It was originally introduced by Google and has become widely used for big data processing. MapReduce consists of three stages: Map, Shuffle, and Reduce.

In the Map stage, the input dataset is divided into chunks by the master. These chunks of data are assigned to the available worker nodes. Workers also receive the map function which is applied to their assigned data to produce some intermediate results. These results are generated in parallel. In the shuffle stage, the results of the computations from the map stage are exchanged among the workers. Depending on the network, workers may have direct communication with each other, or the only way for their communication be through the master. In those cases, the master may engage in the shuffle phase and function as a relay.

Finally, in the reduce stage, the master provides each worker with the reduce function. Workers treat the results that they received in the shuffle stage as a new input and by applying the reduce function, produce the final outputs. The master is responsible for collecting the final results from the worker nodes. In Fig. 2.1 the master-worker model of the MapReduce scheme is depicted.

MapReduce provides a general architecture that can inspire the design of similar master-worker models to compute different large-scale problems. Consider a distributed computing system that consists of one master and k workers. The goal of the master is to compute a job $g(x)$, where x is the input.

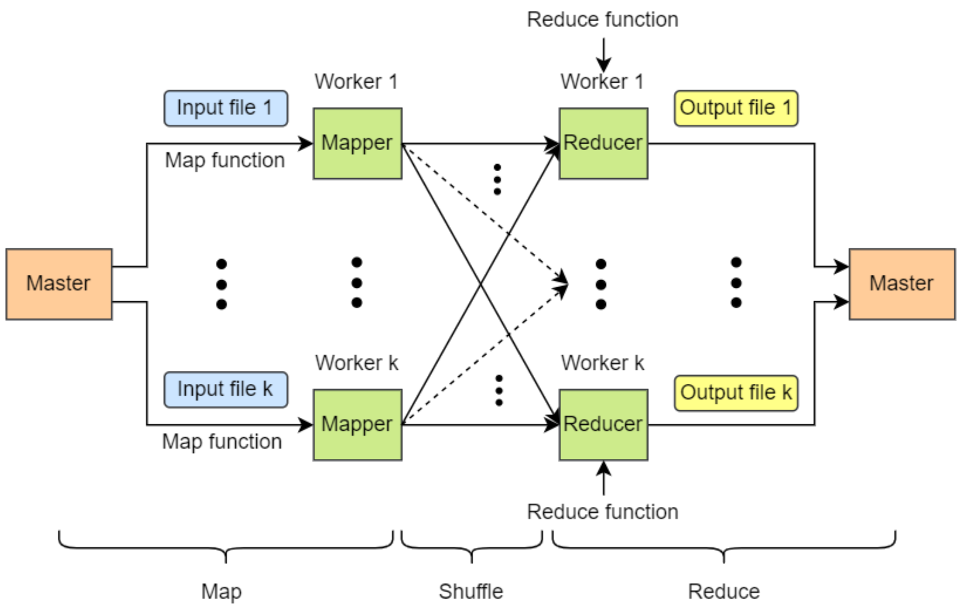


Figure 2.1: The MapReduce model

We assume that $g(x)$ can be decomposed into k tasks, i.e. $g_1(x), \ldots, g_k(x)$ and that the tasks are linear, i.e. $ag_i(x) + bg_j(x) = (ag_i + bg_j)(x)$. The master assigns each task to a worker and collects the result of each computation after they are completed. The master then maps the set of tasks $\{g_i(x)\}_1^k$ by $\phi(\cdot)$ to the main job $g(x) = \phi(g_1(x), \ldots, g_k(x))$.

One example is matrix-vector multiplication where we want to multiply matrix $\mathbf{A} \in \mathbb{R}^{m \times p}$ to vector $\mathbf{x} \in \mathbb{R}^p$. Hence, the intended computation or the job that is required from the master is $g(\mathbf{x}) = \mathbf{A}\mathbf{x}$, $g(\mathbf{x}) \in \mathbb{R}^m$. The master divides this job into k tasks and assigns each task to a worker node. The i-th task here is $g_i(\mathbf{x}) = \mathbf{A}_i\mathbf{x}$ where $g_i(\mathbf{x}) \in \mathbb{R}^{\frac{m}{k}}$ and $\mathbf{A}_i \in \mathbb{R}^{\frac{m}{k} \times p}$ is the i-th submatrix in a horizontally decomposed matrix $\mathbf{A}$. So the master needs to send submatrix $\mathbf{A}_i$ and vector $\mathbf{x}$ to the i-th worker. The i-th worker then starts computing $g_i(x)$. After all the tasks $\{g_i(x)\}_1^k$ are completed, the master must collect the results. In this example $\phi(\cdot)$ simply concatenates the results of the tasks $\{g_i(x)\}_1^k$ and obtains $g(\mathbf{x})$. The system model for matrix-vector multiplication is illustrated in Fig. 2.2.

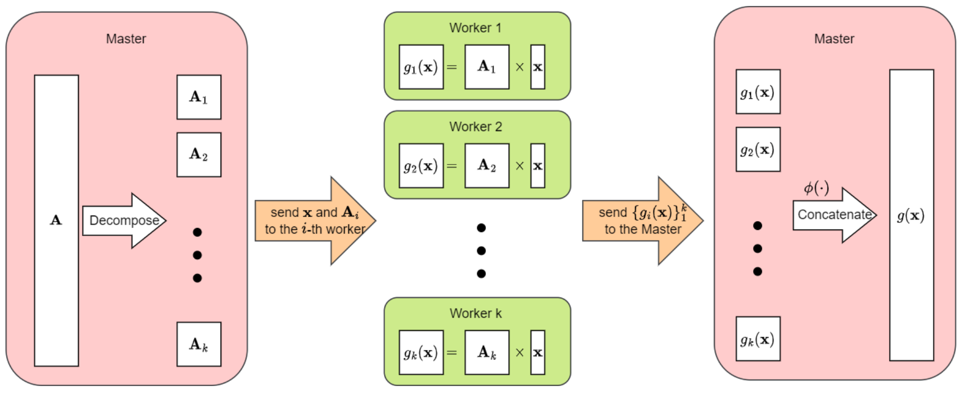

Master
A
Decompose
$\mathbf{A}_1$
$\mathbf{A}_2$
$\mathbf{A}_k$
send $\mathbf{x}$ and $\mathbf{A}_i$ to the i-th worker
Worker 1
$g_1(\mathbf{x}) = \mathbf{A}_1 \times \mathbf{x}$
Worker 2
$g_2(\mathbf{x}) = \mathbf{A}_2 \times \mathbf{x}$
Worker k
$g_k(\mathbf{x}) = \mathbf{A}_k \times \mathbf{x}$
send $\{g_i(\mathbf{x})\}_1^k$ to the Master
Master
$g_1(\mathbf{x})$
$g_2(\mathbf{x})$
$g_k(\mathbf{x})$
$\phi(\cdot)$
Concatenate
$g(\mathbf{x})$

Figure 2.2: Distributed computation of $g(\mathbf{x}) = \mathbf{A}\mathbf{x}$

2.1.2 Challenges

DCSs can face many challenges that might render them useless in practice. In fact, if not implemented carefully, decentralized computing which is considered a solution for computing a large-scale task in the shortest possible time, can turn into a problem itself.

As a necessity, distributed computing systems rely on efficient and reliable communication channels to exchange data and coordinate tasks among multiple interconnected nodes. However, communication links, especially in wireless networks, are not completely reliable. When a communication link fails, it disrupts the flow of information and prevents reporting the result of the nodes that have finished their tasks. Moreover, in some of the networks, the computing nodes are not fully committed. In many of the mobile and fog computing networks, nodes participate in the distributed computing but might leave the network or start higher priority tasks before finishing their computation and reporting their results [6], [14]. When a single or more computing nodes do not complete their tasks, if the divided tasks are disjoint and share no information in common, the main task will never be completed.

Even when the computing nodes are fully dedicated, there is no guarantee that all of them will complete their tasks in a timely manner or at all. In fact, it is common that in a distributed computing system a few nodes can be expected to have excessive delays in finishing their tasks due to various factors including hardware failure, ongoing processes in the memory, and software or algorithmic inefficiencies. These nodes are called stragglers. Straggler nodes impose negative consequences on the performance of distributed computing tasks and can cause extreme delays in the completion of the whole computing task. In other words, in the absence of the proper measures, the main task can not be completed unless every single subtask is finished first. This means the master and other computing nodes have to wait for the slowest computing node. This is in contrast to the original goal of distributed computing, which is completing a task in the shortest possible time. One might argue that the master can re-assign the unfinished tasks to other nodes that have finished their tasks. Nonetheless, this will at least double the computation time.

The key to solving the straggling problem is in reducing the dependency of the system on the result of a single node. One of the promising solutions is introducing some form of redundancy in the distributed tasks. The redundancy gives the system the ability to finish the computation even if some nodes fail. This implies that the subtasks in a distributed computing system should not be merely the divided parts of the main task, but should be related to each other. One of the simplest relations between the tasks is when each task is cloned multiple times [70]. Therefore, if one node straggles, the system can obtain the result from another non-straggling node assigned the same task. Nonetheless, the computational overhead of this method is very high despite its low complexity. Therefore, depending on the requirements and characteristics of the DCS, better alternatives are possible.

For instance, another technique to introduce redundancy is to create an information-theoretic relationship between the subtasks before distributing them. This way, the tasks are not exact replicas of each other but contain information about one another such that in the case of straggling or failure of a node, the system is able to obtain its result without having direct access to it. In this context, a common approach is considering the output of the straggling nodes as erasures. Treating the result of the straggling nodes as erased information forms an analogy between the straggling problem and bit erasure in faulty communication links. This inspires the use of the techniques in coding theory against the straggling problem in a similar manner.

In the following we will first briefly study the coding solutions that are used to create robustness against the bit erasure in communication links and then discuss how they can be leveraged to create fault tolerance in DCSs.

2.2 Forward Error Correction Codes

In communication channels, due to noise, interference, fading, or other channel impairments, some of the information bits might be erased or received with an error. One of the most widely adopted techniques to address this issue is channel coding. There are two primary types of channel coding schemes: error detection codes and error correction codes. Error detection codes, detect the presence of errors in the received data such that if an error is detected, the receiver can request re-transmission of the data. On the other hand, error correction codes not only detect errors but also have the ability to correct them. These codes introduce redundancy in such a way that the receiver can identify and correct a certain number of errors, improving the reliability of the communication.

In order to code a block of information with k bits, $n,\ n \geq k$ new bits are created from them. We note that the amount of information in the n newly generated bits is equal to the information that could be transmitted just by the original k bits. Hence, the $n-k$ new bits are redundant. This process is called coding with a (n,k) code where n is referred to as code length. Although the concept of coding was originally developed for communication channels, its applications are not limited to communications. Recently, error correction codes have been used in distributed storage systems and DCSs to protect data against node failures. Before discussing the details of using error correction codes for DCSs, we will review the relevant mathematical concepts in coding theory as well as a number of families of the codes that will be of use throughout this thesis.

2.2.1 Mathematical Background

Definition 2.1. *group: A group consists of a set G equipped with a map $\top : (G \times G) \mapsto G$ and $\top^{-1} : G \mapsto G$ and an identity element $e \in G$ so that the following axioms hold:*

1. $\forall (g_i, g_j) \in G \ (g_i \top g_j \in G)$.

2. $\forall (g_i, g_j, g_k) \in G \ (g_i \top g_j) \top g_k = g_i \top (g_j \top g_k)$.

3. $\forall g_i \in G \ g_i \top e = g_i$.

4. $\forall g_i \in G \ \exists g_i^{-1}$ *such that* $g_i \top g_i^{-1} = e$.

G is an Abelian or commutative group if an additional axiom is satisfied: $\forall (g_i, g_j) \in G \ g_i \top g_j = g_j \top g_i$

The group defined in Definition 2.1 is either an additive group when the operation ($\top$) is addition denoted as ($+$), or a multiplicative group when the operation is multiplication denoted as ($*$).

Definition 2.2. *Field: A set $\mathbb{F}$ with an addition operation "$+$" and a multiplication operation "$*$" represents a field, denoted $(\mathbb{F}, +, *)$, if*

1. $(\mathbb{F}, +)$ *is an Abelian group with additive identity* "0",

2. $(\mathbb{F} \backslash \{0\}, *)$ *is an Abelian group with multiplicative identity* "1",

3. *For all* $a, b, c \in \mathbb{F}$, $(a + b) * c = a * c + b * c$.

If the set $\mathbb{F}$ has a finite number of elements q, then it is called a finite or Galois field denoted as $GF(q)$ or $\mathbb{F}_q$ where q is the order of the field.

Remark 2.1. *If q is a prime number, the field GF(q) coincides with the ring of integer residues modulo q, also denoted by $\mathbb{Z}_q$.*

2.2.2 Block Codes

An (n, M) (block) code over a finite alphabet F is a nonempty subset $\mathcal{C}$ of size M of F^n. The parameter n is called the code length and M is the code size. The dimension or information length of the code is defined by $k = \log_{|F|} M$, and the rate of $\mathcal{C}$ is $R = k/n$. We will also call $\frac{n-k}{k}$ the overhead of the code.

Each element in a code is called a codeword. After knowing the characteristics of a code like code length and its dimension, it is important to quantify the difference between its codewords. To this end, we will use the following definitions.

Hamming distance

Let F be an alphabet. The Hamming distance between two codewords $\mathbf{x}, \mathbf{y} \in F^n$ is the number of coordinates on which $\mathbf{x}$ and $\mathbf{y}$ differ. We denote the Hamming distance by $\mathrm{d}(\mathbf{x}, \mathbf{y})$.

Hamming weight

Let F be an Abelian group. The Hamming weight of $\mathbf{e} \in F^n$ is the number of nonzero entries in e. We denote the Hamming weight by $\mathrm{w}(\mathbf{e})$. based on this definition, for every two words $\mathbf{x}, \mathbf{y} \in F^n$ we have: $\mathrm{d}(\mathbf{x}, \mathbf{y}) = \mathrm{w}(\mathbf{x} - \mathbf{y})$

Minimum distance

let $\mathcal{C}$ be an (n, M) block code over F. The minimum distance of $\mathcal{C}$ is the minimum Hamming distance between any two distinct codewords of $\mathcal{C}$; i.e, the minimum distance d is given by $d = \min\limits_{\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C} : \mathbf{c}_1 \neq \mathbf{c}_2} \mathrm{d}(\mathbf{c}_1, \mathbf{c}_2)$. An (n, M) code with minimum distance d is called an (n, M, d) code.

2.2.3 Linear Codes

Definition 2.3. *An (n, M, d) code $\mathcal{C}$ over a field $F =$GF(q) is called linear if $\mathcal{C}$ is a linear subspace of F^n over F; namely for every code words $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}$ and two scalars $a_1, a_2 \in F$ we have $a_1\mathbf{c}_1 + a_2\mathbf{c}_2 \in \mathcal{C}$*

The dimension of a linear (n, M, d) code $\mathcal{C}$ over F is the dimension of $\mathcal{C}$ as a linear subspace of F^n over F. If k is the dimension of $\mathcal{C}$, then it is called a linear $[n, k, d]$ code over F, and $n - k$ is the redundancy of the code.

We associate two matrices with a linear code: a generator matrix and a parity-check matrix. The generator matrix of a linear $[n, k, d]$ code over F, denoted by $\mathbf{G}$ is a $k \times n$ matrix whose rows form a basis of the code. Please note that the generator matrix of a linear code is usually not unique. The rank of $\mathbf{G}$ is equal to the dimension of $\mathcal{C}$, k. One can use the generator matrix to map an information word $\mathbf{u} = (u_1 u_2 \ldots u_k)$ to a codeword of $\mathcal{C}$ by $\mathbf{y} = \mathbf{uG}$ where $\mathbf{y} = (y_1 y_2 \ldots y_n) \in F^n$. Since $\text{rank}(\mathbf{G}) = k$, this mapping is one-to-one.

A parity-check matrix of $\mathcal{C}$ denoted by $\mathbf{H}$, in the most common case where rows are independent, is an $(n - k) \times n$ matrix over F such that for every $\mathbf{c} \in F^n$, $\mathbf{c} \in \mathcal{C} \iff \mathbf{Hc}^{\mathrm{T}} = \mathbf{0}$

Theorem 2.1. *Let $\mathbf{H}$ be a parity-check matrix of a linear code $\mathcal{C} \neq \{0\}$. The minimum distance of $\mathcal{C}$ is the largest integer d such that every set of $d - 1$ columns in $\mathbf{H}$ is linearly independent.*[1]

Binary linear codes

Binary linear codes are linear block codes where the codewords are generated by taking linear combinations (modulo-2 addition) of the information bits. In other words, a linear code defined over the field F =GF(2) is called a binary linear code. Binary linear codes have many advantages over non-binary counterparts. The fact that they operate on a two-symbol alphabet (0s and 1s) simplifies the encoding and decoding processes and reduces their complexity. Moreover, the use of binary symbols allows for straightforward circuit implementations, logical operations, and arithmetic computations.

Systematic linear block codes

A linear code is called systematic if it has a generator matrix in the form of $\mathbf{G} = [\mathbf{I}_k, \mathbf{P}] \in \mathbb{F}_q^{k \times n}$. Systematic linear codes are characterized by their

[1]For the proof, we refer the interested reader to [51]

ability to encode a message in a systematic form, where the original message appears as a contiguous portion of the encoded codeword, i.e. when coding the information word $\mathbf{u}$ with a systematic generator matrix, the code word takes the following form: $\mathbf{y} = (\mathbf{u}, \mathbf{uP})$. This feature makes their encoding and decoding less complex.

2.2.4 Maximum Distance Separable (MDS) Codes

Since every two distinct codewords in a code $\mathcal{C}$ with minimum distance d differ on at least d locations, $\mathcal{C}$ is able to recover up to $d-1$ erasures. This means that if a $[n, k, d]$ linear code is used to encode k information symbols into n coded symbols, any $n-d+1$ coded symbols would suffice to recover the original k information symbols.

One of the fundamental bounds in coding theory is the Singleton bound. The Singleton bound for an $[n, k, d]$ linear code states that: $d \leq n - k + 1$ [64]. Any linear code that attains the Singleton bound, namely it satisfies the equality $d = n - k + 1$, is a maximum-distance separable (MDS) code. This means if k information symbols have been coded with an MDS code and are sent through an erasure channel, any set of k coded symbols is enough to recover the original information symbols. Hence, MDS codes are optimal in that sense. Please note that the Singleton bound and MDS are general concepts for every block code. However, here we made our statements specific to linear codes.

Reed-Solomon codes

An important family of linear MDS codes is the family of Reed-Solomon codes. Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be distinct elements of $F =$GF(q). A [normalized generalized] Reed-Solomon code over F is a linear $[n, k, d]$ code with the parity-check matrix:

$$\begin{pmatrix} 1 & 1 & \dots & 1 \\ \alpha_1 & \alpha_2 & \dots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \dots & \alpha_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{n-k-1} & \alpha_2^{n-k-1} & \dots & \alpha_n^{n-k-1} \end{pmatrix}$$

This construction requires that the code length n be at most the field size q. One of the disadvantages of Reed-Solomon codes is that even if one erasure happens, k coded symbols need to be accessed to reconstruct the erased symbol. In the next subsection, we will see that this number can be reduced by using another linear family of codes.

2.2.5 Locally Repairable Codes (LRCs)

Locally repairable codes (LRCs) are a family of linear codes originally proposed for distributed storage systems. They are characterized by their ability to repair or recover from erasures or errors in a localized manner. In LRCs, the codeword is divided into multiple sub-blocks, and each sub-block is associated with a set of local parity symbols. When a symbol is erased or lost, the corresponding local parity symbols can be used to reconstruct the missing symbol. This localized repair property allows for efficient and targeted recovery operations.

Definition 2.4. *(symbol locality, code locality). Let $\mathcal{C}$ be a code over $\mathbb{F}_q$ and let us denote the the i-th coded symbol of $\mathcal{C}$ by y_i and its locality by $Loc(y_i)$. $Loc(y_i)$ is defined as the minimum number of other coded symbols required to reconstruct y_i. In mathematical terms, $Loc(y_i)$ is the size of the smallest set $\mathcal{I}_i \subset \{j\}_1^n \backslash \{i\}$ that satisfies $\sum_{l \in \mathcal{A}_i} \alpha_l y_l = 0$ for a fixed set of coefficients $\alpha_l \in \mathbb{F}_q$ where $\mathcal{A}_i = \mathcal{I}_i \cup \{i\}$. A code $\mathcal{C}$ is said to have locality r if $\max_{i \in \{i\}_1^n} \{Loc(y_i)\} \leq r$.*

A linear code $\mathcal{C}$ with code length n and dimension k that has locality r where $r \leq k$ is called a locally repairable code and is usually denoted as a $[n, k, r]$ code.

2.3 Coding in Distributed Computing

As discussed previously, one of the serious bottlenecks facing the DCSs is the straggling problem. One way to look into the straggling problem is to consider the information (computation results) in the straggling workers as erasures. Consequently, an appealing solution to the issue can be the solution suggested to protect the information in the erasure channels in telecommunication, namely introducing some redundancy through coding.

The key idea here is that after dividing the original task into a number of smaller subtasks and before distributing them among the worker nodes, the subtasks are encoded. Accordingly, it is the coded subtasks that will be distributed among the worker nodes for computation. This idea has been also used in distributed storage to protect the stored data against node failures and erasure, i.e. The data is encoded prior to being stored in the storage nodes. In case of node failure where a part of the stored data is lost, it is the redundancy introduced during the coding process and stored in other nodes that are used to recover the lost data.

In distributed storage, the output data is exactly in the form of input coded data. As a result the decoding algorithm that could be applied on the encoded input to obtain the original data, is valid for the output data as well. In contrast, after the encoded data is distributed among worker nodes in a distributed computing system, the output of each worker node is no longer the input subtask, but the result of its computation. However, if the computation on the input is linear, the corresponding decoding algorithm of the coding performed on the input data is still valid over the computation results. In other words, the same decoding algorithm can be applied to the output of worker nodes to obtain the final results in the form of the final result of an uncoded computation.

Let us go back to the master-worker distributed computing setting introduced in Subsection 2.1.1. Let us again assume that the original task $g(x)$ can be decomposed into k subtasks i.e. $g(x) = \phi(g_1(x), \ldots, g_k(x))$ where $\phi(\cdot)$ is a function that maps the subtasks to the original task. Also, we assume that the subtasks are linear, i.e. $ag_i(x) + bg_j(x) = (ag_i + bg_j)(x)$. In coded distributed computing, the master uses a linear $[n, k, d]$, $n > k$ code on the subtasks to encode them, $h = \mathcal{E}([g_1(x), \dots, g_k(x)])$, where $\mathcal{E}$ encodes the k original subtasks into n new ones, $h = [h_1, h_2, \dots, h_n]$. The master then distributes the new subtasks among n worker nodes.

The worker nodes start to compute their assigned subtasks. After a while, some of the worker nodes finish their subtasks and send the results to the master while other workers either still have not managed to finish the computation or face problems in communicating their results to the master. On the other hand, from our previous discussions on the error correction codes, we know that a $[n, k, d]$ code gives the ability to recover $d-1$ erasures. This means that now that the subtasks are coded, the master does not need to wait for the result of all the subtasks. In fact, the master has enough results to finalize the computation even if $d - 1$ worker nodes straggle or never report the result of their computation. In other words, the master waits only for the result of the fastest workers and when a sufficient number of them report their results, the master considers the rest as erased data. Hence, using codes in DCSs provides robustness against permanent failures, speeds up the process, and solves the straggling issue.

After receiving the results from a sufficient number of workers, $h_{\mathcal{S}} \subset \{h_1, h_2, \dots, h_n\}$, the master decodes the results according to the code used for the encoding of the subtasks in the beginning, $[g_1(x), \dots, g_k(x)] = \mathcal{D}(h_{\mathcal{S}})$. Then the master maps the decoded subtasks to the original task $g(x) = \phi(g_1(x), \dots, g_k(x))$ and finalizes the computation.

As an example, let us consider a coded distributed matrix-vector multiplication, $g(\mathbf{x}) = \mathbf{A}\mathbf{x}$, with an MDS code. Initially, the master will decompose the matrix-vector multiplication into k subtasks. For convenience, let us assume $k = 4$. To create four subtasks, the master will decompose $\mathbf{A}$ to four horizontal blocks. Consequently, the subtasks are $g_1(\mathbf{x}) = \mathbf{A}_1\mathbf{x}, \dots, g_4(\mathbf{x}) = \mathbf{A}_4\mathbf{x}$ where $\mathbf{A}_i$ is the i-th block of $\mathbf{A}$. Let us also assume that the code master will use to encode the subtasks is a [5,4,2] MDS code. Such an MDS code is able to recover only one erasure. To encode the subtasks the master will take the four

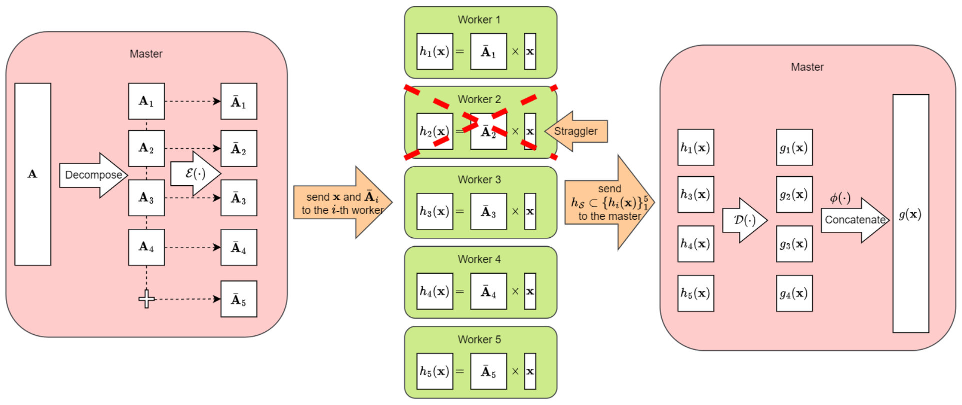


Figure 2.3: Coded Distributed computation of $g(\mathbf{x}) = \mathbf{A}\mathbf{x}$ with the assumption that worker 2 will straggle. In this figure $g_i(\mathbf{x}) = \mathbf{A}_i\mathbf{x}$ and $\mathcal{E}(\cdot)$ and $\mathcal{D}(\cdot)$ denote the encoding and the decoding operations.

decomposed blocks of $\mathbf{A}$ and do element-wise operations to have five encoded blocks. One simple example is element-wise addition. The master keeps the four blocks as they are and creates the fifth block by adding all the blocks element-wise, i.e. $\bar{\mathbf{A}}_1 = \mathbf{A}_1, \ldots, \bar{\mathbf{A}}_4 = \mathbf{A}_4$, and $\bar{\mathbf{A}}_5 = \mathbf{A}_1 + \mathbf{A}_2 + \mathbf{A}_3 + \mathbf{A}_3$. As a result, the new subtask that master will assign to the i-th worker is $h_i = \bar{\mathbf{A}}_i\mathbf{x}$.

The master has now to only wait for four workers to send their results since any combination of four subtasks out of five gives the master the information it needs to finalize the computation of $g(\mathbf{x}) = \mathbf{A}\mathbf{x}$. However, in four out of all five possible orders that the workers might finish their computation, the master needs to execute one more extra step, namely decoding of the received results. The only case that the master can skip this step is when the first workers to finish are the ones assigned with $\{h_i\}_1^4$. When the master recovers $\{g_i(\mathbf{x})\}_1^4$ either after the decoding step or directly, it concatenates them and finalizes the computation of the original task. Fig. 2.3 shows the procedure described above.

2.3.1 Challenges

The use of coding in distributed computing solves the straggler problem to a great extent. However, this comes with a cost. The fact is that coding the tasks introduces redundancy which in turn imposes extra computation on the DCS. On the other hand, the computation resources available to the system are limited and now a part of these resources will be occupied by redundancy. This means the system will divide the original task into larger portions compared to the uncoded case where the task is divided among all the available resources.

Furthermore, the coding introduces two extra pre-processing and post-processing steps, namely the encoding and the decoding of the tasks, that are not required in an uncoded scheme. Since the encoding and the decoding require access to all the tasks, they usually are carried out by a central node or the master. Nonetheless, the goal of distributed computing was to reduce the computational burden of a central processor. It is true that in some applications the encoding and the decoding can be done offline, i.e. in an independent time but there are many other applications that there is a constant need for the encoding and decoding of the new tasks in an online manner. Hence, it is possible that the encoding and decoding steps will become the new bottlenecks. We will address this issue in the next chapter.

2.3.2 Related Work

Many coding solutions have been proposed for different problems in distributed computing. These can be categorized based on many factors. The purpose of coding, for instance, can be mitigating the straggler problem, reducing communication load between the nodes, or enhancing security. The features of computing nodes and the distributed system, the type of code, and the type of computation are other factors to classify the coding solutions.

Mitigating the straggler problem with codes has been the subject of interest in many types of large-scale computation. Matrix multiplication is one of the most widely studied cases as matrix-vector and matrix-matrix multiplications are essential operations in numerous applications such as machine learning, deep learning, and ranking algorithms in search engines [7]. In matrix-vector multiplication, the matrix is divided into smaller blocks that are coded and distributed among the workers while oftentimes, the replicas of the vector are sent to the workers in the original form [15], [56]. Similarly, in the matrix-matrix multiplications, both matrices are divided into smaller blocks and are coded before distribution. The code is designed so that the minimum number of small coded multiplications need to be completed for the final result [17], [37], [75]. The Fast Fourier transform (FFT)[76] and the convolutional operation [16] are two other types of operations studied in this context.

Distributed systems can have very different dynamics and so do the coding schemes proposed for them. In the majority of the coded distributed schemes, the assumption is that in the distributed system all the workers have the same characteristics and have an equal likelihood of exhibiting straggling behavior. In such distributed systems, referred to as homogeneous, equally sized coded subtasks are distributed amongst workers. On the other hand, there are distributed systems, referred to as heterogeneous systems, where the worker nodes possess different computational capacities and memory. In heterogeneous systems, equal load allocation can significantly exacerbate the straggling problem. Hence, optimal load allocation in heterogeneous clusters has been proposed in [34] and [49] propose Heterogeneous Coded Matrix Multiplication (HCMM) algorithm for performing distributed matrix multiplication over heterogeneous clusters.

The type of code in a coded scheme can also depend on the requirements of the system. While some works develop their schemes for a general linear code, others specifically determine the type of code they use. Many of the coded schemes proposed to alleviate the straggling problem are based on MDS codes [33], [36], [37], [48]. MDS codes have an advantage over other codes in terms of their overhead to provide a specific minimum distance. Moreover, they are optimal in the sense that they require the minimum number of coded elements, precisely equal to the number of information elements prior to the coding for the restoration of the mentioned information elements. However, there are studies that adopt other codes especially when, in addition to the computation time, the encoding and the decoding times are also important. In this regard, [56] introduces a coded scheme based on Luby transform (LT) codes under inactivation decoding and [4] propose polar coded distributed matrix multiplication, both enjoying low encoding and decoding complexity.

As mentioned earlier, the applications of coding in distributed computing are not limited to straggler mitigation. For instance, [38], [40] repeat the intermediate computations to create coded multicasting opportunities to reduce communication load, establishing a fundamental trade-off between communication-computation. Later, a unified coding framework was suggested in [39] that combines the coded scheme of [38], [40] and the coded scheme of [36] that generates redundant intermediate computations to combat against straggling servers, giving the ability to stand at any point of a trade-off between the two extreme cases of minimizing either the load of communication or the latency of computation individually. Furthermore, coding solutions are also employed for the security and privacy of distributed computing. For example, [47], [74] practice Lagrange encoding to provide resiliency against stragglers, security against Byzantine (or malicious) workers, and (information-theoretic) privacy of the dataset amidst possible collusion of workers while [8] focus on information-theoretically secure distributed matrix multiplication with the goal of characterizing the minimum communication overhead.

In the rest of this thesis, we will focus on employing coded schemes for reducing the straggler effect. In Chapter 3 we introduce a coded scheme based on a new family of binary locally repairable codes that mitigate the straggler problem with a low encoding and decoding complexity and in Chapter 4 we aim to minimize the computation time of a distributed system in which workers are assigned multiple tasks.

Chapter 3

A Family of Binary Locally Repairable Codes for Coded Distributed Computing

3.1 Introduction

Recent advancements in technology have brought applications that require a massive amount of computation. Terabytes of data are processed by machine learning and data analysis tools. In many cases it is impractical for individual computers to handle such huge operations. As a result, cloud services and distributed computing, which distribute a massive task among many workers using the concept of parallelization, are gaining unprecedented popularity [11].

Achieving the theoretical gains of parallel computations requires many practical considerations. In particular, when a huge computation is distributed across multiple computing devices (workers), factors such as software/hardware failures, communication delays, and maintenance result in varying completion-time across different workers. This means nodes with the same task load and identical hardware configuration will not experience identical computation time. In such a setup, the slow workers are referred to as "stragglers". Stragglers can hinder the overall computation process. In other words, in order to complete the whole task, one needs to wait for the slowest workers to finish their part. This issue is widely referred to as the "straggler problem" in the literature. Fortunately, there are solutions to reduce the stragglers' effect.

One conventional approach to address this problem is repetition [1], [57],

[69]. In repetition, multiple instances of a task are created and distributed among different workers such that in the case of a straggling worker, the exact completed replica of its task could be obtained from another worker. However, due to memory constraints and unwanted recalculation of the tasks already completed, this approach is not efficient.

Recently, error correction codes have been proposed to address this issue [16], [21], [36], [37], [39], [56], [73]. The authors of [36] use an (n, k) maximum distance separable (MDS) code to create $n-k$ redundant tasks for a job broken into k subtasks. The n coded tasks are distributed among workers and the original task can be reduced as soon as any set of k tasks is received by the master.

The approach in [36] eliminates the need to wait to receive the exact replica of a missing task. In addition, it is optimal in the sense of recovery threshold, meaning that it requires the minimum number of received subtasks to complete a computation. Hence, it alleviates the straggler problem and reduces the computational delay. However, the encoding and decoding complexity of the MDS codes, especially multiplication in higher-order finite fields, can be a challenge. Google multiplies matrices of dimensions in the order of $10^{10} \times 10^{10}$ when finding the rank of pages in their search engines [28]. In such scenarios where the matrix is very large, the delay incurred by the encoding and the decoding may overshadow the advantage of using MDS codes for their minimum recovery threshold.

Coding is also used for serving other purposes in distributed computing. For instance, coding is used in [24], [38] to alleviate bandwidth bottlenecks, heterogeneous servers are considered in [13], [19], [32], and optimizing the throughput via coding is considered in [72]. Also [30] studies coded distributed optimization and [14], [45], [54] propose coded federated learning schemes.

Binary codes are a family of error-correction codes that enjoy low computational costs in the encoding and decoding phases due to the absence of multiplication operations. Hence, when coding complexity is a bottleneck, using binary codes can be an attractive option. [9] uses binary codes for coded distributed matrix multiplication with such an objective. In addition, further complexity reduction can be achieved using the concept of locality. Code locality is defined as the maximum number of coded symbols required to reconstruct any missing coded symbol [44]. Locally repairable codes (LRCs) have recently gained a lot of attention in distributed storage systems (DSS)[10], [22], [26], [31], [35], [46], [53], [66], [68], [71]. The reason for this interest is that for an LRC with a low locality, the number of required accesses to the storage nodes in order to restore a missing symbol is small. Likewise, in distributed computing, instead of needing the whole code structure, using LRCs allows for recovery from a missing subtask using the local structure. This can significantly reduce the decoding complexity.

In this chapter, we propose the use of binary locally repairable codes (BLRCs) for a coded distributed computing scenario. BLRCs enjoy the advantages of both binary and locally repairable code families [23], [58], [59], [63]. Previously, [25], [27], [59] introduced binary locally repairable codes for DSS that are optimal with respect to Singleton-like bound[1]. This is because the optimization goal for DSS is reducing the storage overhead. However, since the optimization goal in coded distributed computing is the time delay of the process, being optimal with respect to singleton-like bound might not be the first priority in coded distributed computing. In fact, the encoding and decoding complexity often play a more important role in the overall time delay and one might sacrifice optimality with respect to Singleton-like bound to minimize encoding/decoding complexity. We propose a new family of BLRCs with such a design goal. In our coded distributed scheme using our proposed family of BLRCs, the master can easily handle the encoding process just by implementing a few XOR operations in the binary field. This applies to the recovery process of the main task as well. The locally repairable nature would reduce the decoding complexity even further. After a sufficient set of subtasks are collected, the master uses the local redundancies to recover any missing

[1]Singleton-like bound states that the minimum distance d of a (n, k, r) LRC is bounded as [22], [44]:

$$d \leq n - k - \lceil \frac{k}{r} \rceil + 2.$$

data blocks, and only in the cases that there are no sufficient local redundancies, it uses the global relationships between subtasks to recover the missing ones. Hence, in most cases, the high-complexity global structure is not used.

Our contributions in this chapter are as follows:

- To the best of our knowledge, we are the first to suggest using LRCs for distributed computing systems. By prioritizing the local redundancies for the recovery of missing subtasks, we reduce the decoding (recovery) complexity. As a result, the decoding time and hence the overall completion time is reduced.

- We introduce a new family of BLRCs specifically designed for coded distributed computing scenarios. This family of codes enjoys very low encoding and decoding complexity. The locality and the length of the introduced codes are flexible and can be adjusted based on the requirements of the distributed setting.

- For a code length of n and locality of r such that $(r+1)|n$, our code has $g = \frac{n}{r+1}$ local groups and a minimum distance of $d = 4$. We prove that among all BLRCs with this local structure, and the same minimum distance, our proposed BLRCs need the minimum number of XORs for the encoding process. Furthermore, given the number of XORs for encoding, we prove that our code has the best code rate among those with the same local structure and minimum distance.

- We also suggest a decoding scheme that guarantees that the recovery process will be completed with the minimum number of XORs. Experiments show that our scheme reduces the combined time delay of the encoding and decoding phases up to more than 99% in some cases compared to the MDS codes.

- It is known that MDS codes have the lowest recovery threshold and hence the optimal computation time. However, we show that in spite of a longer computation time, our BLRCs have a total time delay that is much lower than a system based on MDS codes. This is due to the significant time delay reduction in the encoding and decoding phases.

The rest of this chapter is organized as follows. In Section 3.2, we present our distributed computing system model. In Section 3.3, we introduce our family of BLRCs and analyze its features. We also prove two senses of optimality for our codes and discuss their minimum-complexity encoding and decoding procedures. In Section 3.4, we simulate different distributed computation scenarios and show the time delay performance of our codes. Finally, in Section 3.5 we conclude this chapter.

Notations: Matrices and vectors are denoted by upper and lower boldface letters. $\mathbb{F}_{2^l}$ shows a finite field with cardinality 2^l. Calligraphic upper case letters show a set and $|\mathcal{S}|$ denotes the cardinality of the set $\mathcal{S}$. Finally, $\{i\}_1^n$ represents the set of all integers from 1 to n.

3.2 System Model and Preliminaries

Matrix-vector multiplications are among the most fundamental operations in distributed computing systems. The linear nature of matrix-vector multiplication makes it a suitable candidate for coded distribution. In this section, we give details about the considered matrix-vector multiplication, our assumptions about the distributed setting, and the three main phases of the coded distributed computation.

3.2.1 Matrix-Vector Multiplication Formulation

We consider a matrix-vector multiplication problem where matrix $\mathbf{A} \in \mathbb{F}_{2^l}^{m\times p}$ is multiplied to N vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbb{F}_{2^l}^{p}$. Here $\mathbb{F}_{2^l}$ is an extended binary field. In many cases, $N = 1$, but we consider a more general setup where N is not necessarily 1. For instance, in gradient descent algorithms, in each forward pass, a new weight matrix $\mathbf{A}$ representing the parameters of the model is multiplied by many vectors representing features of the data points in a batch of size N.

We consider a scenario in which there is a master that, with the help of n nodes called workers, finishes the multiplication task $\mathbf{Y} = [\mathbf{A}\mathbf{x}_1, \mathbf{A}\mathbf{x}_2, \ldots, \mathbf{A}\mathbf{x}_N]$. We assume the computational capacity of the master is the same as each of the workers. The master partitions the rows of the matrix $\mathbf{A}$ into k submatrices each consisting of $\frac{m}{k}$ rows and p columns:

$$\mathbf{A} = [\mathbf{A}_1; \mathbf{A}_2; \ldots; \mathbf{A}_k].$$

By such a partitioning, the original task $\mathbf{Y}$ is the concatenation of subtasks $\mathbf{Y}_i$, $i \in \{i\}_1^k$:

$$\mathbf{Y} = [\mathbf{Y}_1; \mathbf{Y}_2; \ldots; \mathbf{Y}_k],$$

where $\mathbf{Y}_i = [\mathbf{A}_i\mathbf{x}_1, \mathbf{A}_i\mathbf{x}_2, \ldots, \mathbf{A}_i\mathbf{x}_N]$. The master uses an encoding matrix to linearly encode k blocks of $\mathbf{A}$ into n new submatrices:

$$\bar{\mathbf{A}} = [\bar{\mathbf{A}}_1; \bar{\mathbf{A}}_2; \ldots; \bar{\mathbf{A}}_n],$$

where $\bar{\mathbf{A}} \in \mathbb{F}_{2^l}^{n\frac{m}{k} \times p}$. We assume that vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ are known to all the workers. The master sends submatrix $\bar{\mathbf{A}}_i$ to the i-th worker where $i \in \{1, 2, \ldots, n\}$. Upon receiving the submatrix, the i-th worker starts to compute $\bar{\mathbf{Y}}_i = [\bar{\mathbf{A}}_i\mathbf{x}_1, \bar{\mathbf{A}}_i\mathbf{x}_2, \ldots, \bar{\mathbf{A}}_i\mathbf{x}_N]$ and returns the result after finishing the calculation of the products. The encoding matrix that the master uses is built based on an (n, k) linear code, where k and n are the numbers of information symbols and code length, respectively.

In this work, it is assumed that encoding cannot be done offline and hence its complexity cannot be ignored. There exist scenarios where matrix $\mathbf{A}$ does not change over time, hence it can be encoded once and offline. However, in many practical scenarios (e.g., gradient descent optimization of deep neural networks), $\mathbf{A}$ changes frequently and thus its encoding cannot be done offline. As an example, consider a regression problem in machine learning solved by a gradient descent algorithm. That is, given $\mathbf{x}$ and $\mathbf{y}$, we are interested to solve $\mathbf{A}\mathbf{x} = \mathbf{y}$ for $\mathbf{A}$. Let $\mathbf{A}_i$ be the estimated $\mathbf{A}$ in iteration i. In each iteration, we need to find $\hat{\mathbf{y}}_i = \mathbf{A}_i\mathbf{x}$ (a matrix-vector multiplication) in order to obtain the mean square error of our estimation. Since $\mathbf{A}_i$ depends on the loss function, it cannot be known beforehand, and hence, it cannot be pre-encoded. Therefore, the encoding time in such a problem cannot be ignored.

3.2.2 Probabilistic Model of Time Delay

The shifted exponential distribution has been widely used to model the computational delay among the workers [21], [36], [39], [56]. Following the literature, we assume that the time it takes a worker to complete its task is an independent random variable that follows a shifted exponential distribution denoted by T. Since our setting is homogeneous, i.e., all the workers have the same computational capacity, they all follow the same distribution with the cumulative distribution function (CDF):

$$F_T(t;\sigma) = \begin{cases} 1 - e^{-\left(\frac{t}{\sigma}-1\right)}, & \text{for } t \geq \sigma \\ 0, & \text{otherwise} \end{cases}, \tag{3.1}$$

where σ is a parameter for scaling the distribution based on the computational load of the tasks. Like [18], we assume that the time units required for completing an addition and a multiplication, denoted by σ_A and σ_M, are in $\mathcal{O}\left(\frac{l}{64}\right)$ and $\mathcal{O}(l \log_2 l)$ over $\mathbb{F}_{2^l}$, respectively. Please note that while we assume a binary code, the matrix-vector multiplications are performed in an extended binary field.

The shift in the shifted exponential distribution is the minimum time for completing a task. The tail of the distribution models unpredictable delays, i.e., the straggling behavior.

Let us denote the completion time of worker i, $i \in \{i\}_1^n$, by T_i. Random variables $T_1, \ldots, T_n$ are independent and identically distributed with CDF $F_T(t;\sigma)$. We denote the time delay for the i-th fastest worker by $T_{(i)}$. By the i-th order statistic analysis [3] one could simply verify that $T_{(i)}$ is a Gamma-distributed random variable with the expectation:

$$\mu(\sigma, n, i) \triangleq \mathbb{E}[T_{(i)}] = \sigma \left(1 + \sum_{j=n-i+1}^{n} \frac{1}{j}\right)$$

3.2.3 Distributed Computing Model

Similar to [2], [33], our scheme is based on a master and n workers, all with the same computational capacity. The master starts the process by splitting the matrix $\mathbf{A}$ into k blocks and encoding them into n new blocks with some redundancies introduced. Then it sends each of the coded blocks to the corresponding worker. The workers start the computation of subtasks and when they finish, immediately send the results back to the master. The moment the master receives enough results to recover the original computation, it halts the ongoing processes. We denote by $\mathcal{S} \subset \{i\}_1^n$ the set of indices of the workers that have produced sufficient coded results for the master to recover the main task. The master starts to decode these encoded blocks $\bar{\mathbf{Y}}_i$, $i \in \mathcal{S}$, to obtain all the information blocks $\mathbf{Y}_i$, $i \in \{i\}_1^k$.

While most existing work ignores the encoding and decoding time of the master, and models the waiting time entirely based on the computation time of the helpers, in this work, we take into consideration the time delays incurred by the master during the encoding and decoding of the subtasks. When m or p are large the encoding operations can be quite costly and ignoring the encoding time is not an accurate assumption, especially when the master has similar computation abilities as helpers. Likewise, when m, n, or N are large, the decoding time cannot be neglected.

Hence, the overall distributed computing process proceeds in three phases; *encoding, computation,* and *decoding.*

Encoding phase

In this phase, the master divides matrix $\mathbf{A}$ into k submatrices and uses an encoding matrix $\mathbf{E} \in \mathbb{F}_2^{n\frac{m}{k} \times m}$ to obtain the coded matrix $\bar{\mathbf{A}}$:

$$\bar{\mathbf{A}} = \mathbf{E}\mathbf{A}. \tag{3.2}$$

The encoding matrix is built based on the systematic generator matrix $\mathbf{G}$ of a linear code. Let $\mathbf{I}_{\frac{m}{k}}$ be the $\frac{m}{k} \times \frac{m}{k}$ identity matrix, $\mathbf{E}$ is calculated as:

$$\mathbf{E} = \mathbf{G}^T \otimes \mathbf{I}_{\frac{m}{k}}, \tag{3.3}$$

where $\otimes$ denotes the Kronecker product. In our scheme, we choose the generator matrix $\mathbf{G}$ to be systematic in order to reduce the encoding and decoding complexity. A systematic generator matrix for an (n, k) linear code is of the form $[\mathbf{I}_{k\times k}|\mathbf{P}_{k\times(n-k)}]$, as a result, the first k blocks in the matrix $\bar{\mathbf{A}}$ are exactly the same k blocks in the matrix $\mathbf{A}$. Hence, when the master receives any of the product sets $\{\bar{\mathbf{A}}_i\mathbf{x}_j : j = 1, \ldots, N\}$ from worker i, $i \in \{1, \ldots, k\}$, no decoding will be required. We call these blocks information blocks and the remaining $n - k$ blocks are referred to as redundant blocks. In an ideal case, where the first k blocks to be received by the master are the k information blocks, the process of distributed computing ends without decoding. Otherwise, some of the received redundant blocks will be used to retrieve the missing information blocks.

We model the time delay for completing the encoding phase for the master by considering two potential scenarios. In the first scenario, the master can suffer straggling behavior in the same way as the workers, hence the time delay is modeled by a shifted exponential distribution with a parameter σ_{encode}. In the second scenario, however, the master has a stable behavior and the time delay is a deterministic value scaled by the encoding load, i.e., the time delay is equal to σ_{encode}, or equivalently, the time shift in the shifted exponential distribution.

We will discuss σ_{encode} for our BLRC scheme in Section 3.3. Here, we focus on MDS codes. Here, σ_{encode} will be the weighted sum of all the addition and multiplication operations performed to calculate the product $\mathbf{EA}$ with σ_A and σ_B as the weights, respectively. Assuming a systematic generator matrix, $\mathbf{G}_{\text{MDS}}$, the first k blocks in $\bar{\mathbf{A}}$ do not require any operation and are directly obtained. The remaining $n-k$ blocks correspond to the product of last $(n-k)\frac{m}{k}$ rows of $\mathbf{E}$ and p columns of $\mathbf{A}$. Please note that each row in the last $(n-k)\frac{m}{k}$ rows of $\mathbf{E}$ is sparse and has only k non-zero elements. This operation hence requires $p\left((n-k)\frac{m}{k} \cdot k\right)$ multiplications and $p\left((n-k)\frac{m}{k} \cdot (k-1)\right)$ additions. Thus, σ_{encode} for an MDS code would be:

$$\sigma_{\text{encode, MDS}} = pm(n-k)\sigma_M + \frac{pm(k-1)(n-k)}{k}\sigma_A.$$

We later use these parameters in our simulations and show how our proposed code significantly reduces the time delay incurred by the encoding phase in the master.

Computation phase

At this stage each worker i receives its $\bar{\mathbf{A}}_i$ from the master and starts calculating $\bar{\mathbf{Y}}_i$. When a worker finishes all its subtasks, it immediately sends them back to the master. We model the time for completing the subtasks in each worker by a shifted exponential distribution with parameter $\sigma_{\text{computation}}$ being the weighted sum of the total number of additions and multiplications performed in the calculation of the products. We already know that $\bar{\mathbf{A}}_i \in \mathbb{F}_{2^l}^{\frac{m}{k}\times p}$ and $\mathbf{x}_j \in \mathbb{F}_{2^l}^{p}$, $j \in \{1,\ldots,N\}$, so in the worker i, $\frac{m}{k}$ rows of $\bar{\mathbf{A}}_i$ each with length p are multiplied to N vectors of the same length. Thus $\sigma_{\text{computation}}$ for each worker is calculated as follows:

$$\sigma_{\text{computation}} = N\left(\frac{m}{k}p\sigma_M + \frac{m}{k}(p-1)\sigma_A\right). \tag{3.4}$$

Parameter $\sigma_{\text{computation}}$ is calculated regardless of the coding scheme. The difference between an MDS code and a BLRC in this phase is in their corresponding minimum recovery thresholds. We define the minimum recovery threshold (MRT) as follows:

Definition 3.1. *The minimum recovery threshold in an (n,k) erasure code is the minimum number of coded blocks required to recover the original k information blocks.*

When MDS codes are used, any combination of the first k blocks received by the master is enough to recover all the original k information blocks. Hence, MRT for MDS codes is k. However, if a BLRC is used there is no guarantee that the first k received blocks are sufficient to recover the missing information blocks. In fact, by using the proposed BLRCs, we slightly sacrifice the time delay of the computation phase in return for a significant gain in the time delay of the encoding and decoding phases such that the overall time delay is less than that of the MDS case.

Decoding phase

In this phase, the master uses available coded results to recover the k information blocks. In the case of a systematic code, the master has received a number of original information blocks and some other redundancies. The master does not perform decoding for the information blocks it receives. On the other hand, the information blocks that are missing need to be recovered based on the redundant blocks. We will model the time delay of the decoding phase for the master, in the same manner we described for the encoding phase with two possible scenarios: steady behavior and the straggling behavior of the master. The proposed BLRC scheme in this work aims to reduce the complexity of the decoding phase by, firstly, using only XOR operation to create redundancies and, secondly, wisely dividing them into local and global groups. The decoding procedure, the computational complexity of the decoding, the number of XORs required, and the σ_{decode} for the proposed BLRC will be discussed further in Section 3.3. For MDS codes, the decoding process can be complex and costly especially when the number of vectors to be multiplied is high or the dimensions of matrix $\mathbf{A}$ are very large. This is in part because all the k information blocks are included in generating each of the redundant blocks. The other reason is the multiplication operations required in the decoding process. Depending on the MDS code used and the decoding algorithm, one can face different computational costs. We assume that the MDS code used is a Reed-Solomon (RS) code. To the best of our knowledge, the lowest complexity algorithm for decoding RS codes is FFT-based decoding with complexity in $\mathcal{O}\left(r \log r\right)$ [42]. We take the number of estimated addition and multiplications required for a given code of length n from empirical results in [56] to be $2 + 8.5n \log_2\left(0.867n\right)$ and $2 + n\log_2\left(4n\right)$, respectively. Thus we can calculate the σ parameter associated with the MDS decoding as:

$$\sigma_{\text{decode, MDS}} = N\frac{m}{k}\Big(2 + 8.5n \log_2\left(0.867n\right)\Big)\sigma_A + N\frac{m}{k}\Big(2 + n\log_2\left(4n\right)\Big)\sigma_M$$

3.3 Proposed BLRC for Distributed Computing

In this section, we propose a BLRC family with a minimum distance of four and locality r. We discuss the flexibility of our proposed code in terms of the locality and the code length. Then we analyze the complexity and computational cost of the encoding and decoding phases of a coded distributed scheme that uses our family of BLRC. Finally, we prove two senses of optimality for our proposed codes.

3.3.1 Linear Codes Preliminary

Before we go into the detailed construction procedure of our proposed BLRC, we lay out definitions of some terms that will be used frequently.

Definition 3.2. *The locality of an (n, k) erasure code, denoted r, is defined as the maximum number of coded symbols required to reconstruct any missing coded symbol.*

Definition 3.3. *BLRCs are a family of linear block erasure codes that operate on the binary field and have the locality $r < k$ where k is the number of information symbols.*

Definition 3.4. *For a linear erasure code $\mathcal{C}$, the minimum Hamming distance of any two arbitrary codewords is defined as its minimum distance. In other words, we have:*

$$d = \min \{d(\mathbf{u}, \mathbf{v}) : \mathbf{u}, \mathbf{v} \in \mathcal{C}, \mathbf{u} \neq \mathbf{v}\},$$

where $d(\cdot, \cdot)$ denotes the Hamming distance.

It is shown that in a linear code with minimum distance d, any $d-1$ erasures can be recovered [51]. If we denote the parity check matrix of an (n, k, d) linear code $\mathcal{C}$ with $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, the minimum distance of $\mathcal{C}$ is equal to the minimum number of columns of $\mathbf{H}$ that are linearly dependent[51].

Definition 3.5. *The Tanner graph of an (n, k) code is a bipartite graph $G(\mathcal{V}, \mathcal{U}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{U}$ are two disjoint sets of vertices connected by edges* *in $\mathcal{E}$. Here, $\mathcal{V}$ is the set of n variable nodes and $\mathcal{U}$ is the set of $n-k$ check nodes. The i-th node in $\mathcal{V}$ is adjacent to the j-th check node in $\mathcal{U}$ if and only if $\mathbf{H}_{ij}$ is nonzero.*

Based on the above definition, each $v_i \in \mathcal{V}$ represents a coded symbol and the nodes from $\mathcal{V}$ that are connected to a specific check node from $\mathcal{U}$ form a linearly dependent set. In the case of binary codes, the result of the XOR among all the variable nodes connected to a check node is equal to zero.

3.3.2 Construction of the Proposed BLRC

In this section, we introduce the procedure to build our proposed BLRC. Two of the important features for every linear erasure code are the minimum distance and code rate. The trade-off between the code rate and the minimum distance is an important factor to be considered in code design. Specifically, when the code is binary, a high minimum distance would require a great sacrifice in the code rate.

In distributed computing, coding is mainly used to combat stragglers [36], i.e. computing devices that finish their tasks with delay. Also, when a straggler becomes a “permanent straggler” (e.g., due to hardware failure), their task is failed forever and the code’s ability to restore their permanently lost computation becomes important. However, in most practical distributed systems, the probability of straggler behavior is low and permanent stragglers are very rare, meaning that a large minimum distance may not be required. For instance, 3-replication coding with a minimum distance of $d = 3$ is commonly used for its enough robustness against straggler behavior [26], [53]. As an example, consider a computation that takes a few minutes and is distributed among $n = 20$ computing nodes. It is very unlikely that more than three nodes experience hardware failures in such a scenario. Hence, we choose the minimum distance of our code to be $d = 4$. This way, reasonable robustness against stragglers is achieved.

As mentioned before, the other factor that is important is the locality of the code. To enforce a locality of r, we start building our code by creating and repeating a specific local structure of $r+1$ coded symbols. This local structure has been adopted by many previous works [59], [62], [66] and we will call any BLRC with this local structure a "well-structured" code.

Definition 3.6. *If we denote the length of a locally repairable code with n and its locality with r, and call $g = \frac{n}{r+1}$ the number of local groups, then a well-structured code is a code that its parity check matrix could be written as:*

$$\mathbf{H}_{\textit{well-structured}} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{v}_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{v}_g \\ \hdashline \multicolumn{4}{c}{\mathbf{H}_G} \end{bmatrix}$$

where $\mathbf{v}_i \in \mathbb{F}_q^{1\times(r+1)}$ for $i \in \{i\}_1^g$, $\mathbf{0}$ shows a vector of $r+1$ zeros, and $\mathbf{H}_G \in \mathbb{F}_q^{(n-k-g)\times n}$. Here $\mathbf{v}_i$ constructs the i-th local check node, creating the ability to locally recover one erasure. A well-structured code is depicted in Fig. 3.1.

Our proposed well-structured code, depicted in Fig. 3.1, guarantees the local recovery of one erasure. Now, to increase the recovery capacity to three erasures (i.e., $d = 4$), we devise a special global structure between members of different localities. Later, we will show that our choice of global structure guarantees the minimum number of XOR operations for the encoding and decoding among all the possible well-structured codes with a minimum distance $d = 4$. Hence, our design minimizes the coding complexity.

In our code, like any other well-structured code, $(r+1)|n$. This way, we can divide the variable nodes in the Tanner graph into $g = \frac{n}{r+1}$ groups. Each of the first $g-1$ groups contains $r+1$ nodes out of which r of them are information symbols. The last node, in each group, is the XOR of the other r information symbols. These $r+1$ nodes together are connected to a *local* check node, hence the XOR of all of them amounts to zero. This guarantees the locality of r for our code. The g-th group has also the same structure but unlike the previous groups, the first r variable nodes are not information symbols. In fact, the i-th, $i \in \{i\}_1^r$, node in the g-th group contains the XOR result of the i-th nodes from all the previous groups and together they are connected to a

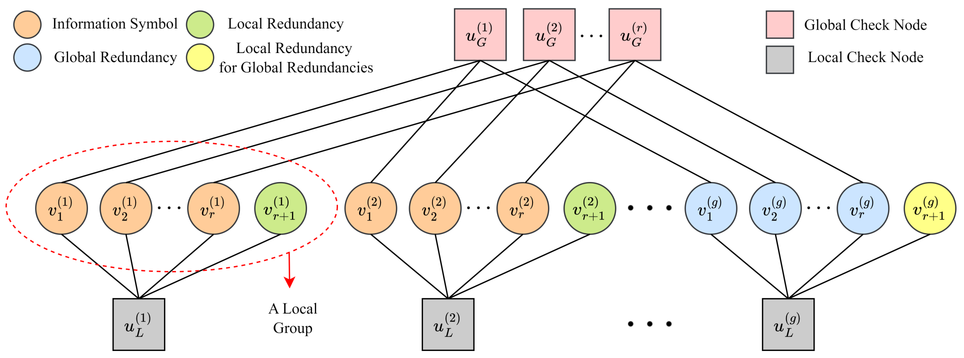


Figure 3.1: The Tanner graph of the proposed BLRC; local and global check nodes have been denoted by $u_L^{(\cdot)}$ and $u_G^{(\cdot)}$ respectively

check node that we call it a *global* check node. The $(r+1)$-th node in the g-th group like the previous groups is the XOR of the previous r nodes in the same group. One can see that the $(r+1)$-th node in the g-th group is in fact the XOR of all the information symbols in the graph too. Note that in our code, the choice of r is flexible, and the only constraint on n is that $(r+1)|n$. This results in

$$k = (g-1)r,$$

where $g = \frac{n}{r+1}$. Now that we have the Tanner graph of our proposed BLRC, building a parity check matrix based on it would be possible. Let us break the parity check matrix into two blocks as:

$$\mathbf{H}_{\text{BLRC}} = \begin{pmatrix} \mathbf{H}_L \\ \mathbf{H}_G \end{pmatrix} \in \mathbb{F}_2^{(n-k)\times n}, \tag{3.5}$$

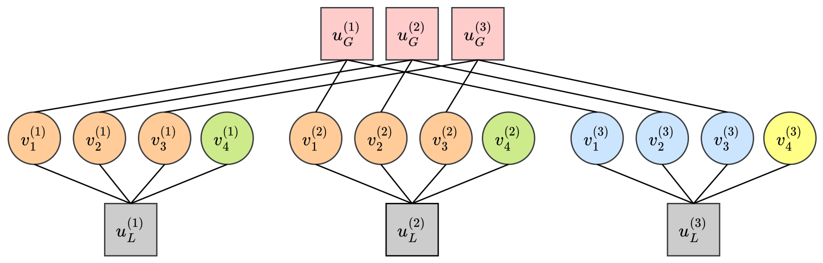
where $\mathbf{H}_L \in \mathbb{F}_2^{g\times n}$ and $\mathbf{H}_G \in \mathbb{F}_2^{(n-k-g)\times n}$ are associated with the local and global check nodes, respectively. Based on the structure of the local check nodes,

$$\mathbf{H}_L = \mathbf{I}_g \otimes \mathbf{1}_{1\times(r+1)} \in \mathbb{F}_2^{g\times n}, \tag{3.6}$$

where $\mathbf{1}$ denotes a matrix of all ones. On the other hand

$$\mathbf{H}_G = \mathbf{1}_{1\times g} \otimes [\mathbf{I}_r \ \mathbf{0}_{r\times 1}] \in \mathbb{F}_2^{(n-k-g)\times n}, \tag{3.7}$$

where $\mathbf{0}$ denotes a matrix of all zeroes. By expanding the building blocks of $\mathbf{H}_{\text{BLRC}}$, it will have the following form:



$u_G^{(1)}$
$u_G^{(2)}$
$u_G^{(3)}$
$v_1^{(1)}$ $v_2^{(1)}$ $v_3^{(1)}$ $v_4^{(1)}$
$v_1^{(2)}$ $v_2^{(2)}$ $v_3^{(2)}$ $v_4^{(2)}$
$v_1^{(3)}$ $v_2^{(3)}$ $v_3^{(3)}$ $v_4^{(3)}$
$u_L^{(1)}$
$u_L^{(2)}$
$u_L^{(3)}$

$$\mathbf{H} = \begin{array}{c} \left(\begin{array}{cccccccccccc} 1&1&1&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&1&1&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&1&1&1\\ 1&0&0&0&1&0&0&0&1&0&0&0\\ 0&1&0&0&0&1&0&0&0&1&0&0\\ 0&0&1&0&0&0&1&0&0&0&1&0 \end{array}\right) \end{array} \begin{array}{c} u_L^{(1)}\\ u_L^{(2)}\\ u_L^{(3)}\\ u_G^{(1)}\\ u_G^{(2)}\\ u_G^{(3)} \end{array} \quad \mathbf{G} = \left(\begin{array}{cccccc|cc|c|ccc} v_1^{(1)}&v_2^{(1)}&v_3^{(1)}&v_1^{(2)}&v_2^{(2)}&v_3^{(2)}&v_4^{(1)}&v_4^{(2)}&v_4^{(3)}&v_1^{(3)}&v_2^{(3)}&v_3^{(3)}\\ 1&0&0&0&0&0&1&0&1&1&0&0\\ 0&1&0&0&0&0&1&0&1&0&1&0\\ 0&0&1&0&0&0&1&0&1&0&0&1\\ 0&0&0&1&0&0&0&1&1&1&0&0\\ 0&0&0&0&1&0&0&1&1&0&1&0\\ 0&0&0&0&0&1&0&1&1&0&0&1 \end{array}\right)$$

Figure 3.2: The Tanner graph of the proposed $(n, k, r) = (12, 6, 3)$ BLRC code

$$\mathbf{H}_{BLRC} = \left(\begin{array}{cccc} \overbrace{\begin{matrix} 1 & \dots & 1 & 1\\ 0 & \dots & 0 & 0\\ \vdots & \ddots & \vdots & \vdots\\ 0 & \dots & 0 & 0 \end{matrix}}^{1^{st}\text{Local Group}} & \overbrace{\begin{matrix} 0 & \dots & 0 & 0\\ 1 & \dots & 1 & 1\\ \vdots & \ddots & \vdots & \vdots\\ 0 & \dots & 0 & 0 \end{matrix}}^{2^{nd}\text{Local Group}} & \dots & \overbrace{\begin{matrix} 0 & \dots & 0 & 0\\ 0 & \dots & 0 & 0\\ \vdots & \ddots & \vdots & \vdots\\ 1 & \dots & 1 & 1 \end{matrix}}^{g^{th}\text{Local Group}} \\ \begin{matrix} 1 & \dots & 0 & 0\\ \vdots & \ddots & \vdots & \vdots\\ 0 & \dots & 1 & 0 \end{matrix} & \begin{matrix} 1 & \dots & 0 & 0\\ \vdots & \ddots & \vdots & \vdots\\ 0 & \dots & 1 & 0 \end{matrix} & & \begin{matrix} 1 & \dots & 0 & 0\\ \vdots & \ddots & \vdots & \vdots\\ 0 & \dots & 1 & 0 \end{matrix} \end{array}\right) \begin{array}{l} \left.\vphantom{\begin{matrix}1\\1\\1\\1\end{matrix}}\right\} \mathbf{H}_L \\ \left.\vphantom{\begin{matrix}1\\1\\1\end{matrix}}\right\} \mathbf{H}_G \end{array} \tag{3.8}$$

Example 3.1. *As an example, let us build an $(n, k, r) = (12, 6, 3)$. The relation between the* **H** *matrix and the corresponding Tanner graph of the proposed (12,6,3) BLRC is illustrated in Fig. 3.2. Based on the figure, the parity check matrix of the code is:*

$$\mathbf{H} = \left(\begin{array}{ccc} \overbrace{\begin{matrix} 1&1&1&1\\ 0&0&0&0\\ 0&0&0&0 \end{matrix}}^{1^{st}\,Local\ Group} & \overbrace{\begin{matrix} 0&0&0&0\\ 1&1&1&1\\ 0&0&0&0 \end{matrix}}^{2^{end}\,Local\ Group} & \overbrace{\begin{matrix} 0&0&0&0\\ 0&0&0&0\\ 1&1&1&1 \end{matrix}}^{3^{rd}\,Local\ Group} \\ \begin{matrix} 1&0&0&0\\ 0&1&0&0\\ 0&0&1&0 \end{matrix} & \begin{matrix} 1&0&0&0\\ 0&1&0&0\\ 0&0&1&0 \end{matrix} & \begin{matrix} 1&0&0&0\\ 0&1&0&0\\ 0&0&1&0 \end{matrix} \end{array}\right).$$

Proposition 3.1. *The minimum distance (d) of the proposed BLRC family is $d = 4$.*

Proof. Let us show each column in $\mathbf{H}_{\text{BLRC}}$ with $\mathbf{h}_i^{(j)}$ where $i,\ i \in \{i\}_1^{r+1}$, shows the position of the column in the local group and $j,\ j \in \{j\}_1^{g}$, shows the local group that the column belongs to. Based on the structure in (3.8) we notice that each column has two parts; one part is in $\mathbf{H}_L$ and the other part is in $\mathbf{H}_G$. Let us denote the part of $\mathbf{h}_i^{(j)}$ in $\mathbf{H}_L$ with $\mathbf{h}_{i,L}^{(j)}$ and the part in $\mathbf{H}_G$ with $\mathbf{h}_{i,G}^{(j)}$. One can immediately see that:

(1) $\mathbf{h}_{i,G}^{(j)} \oplus \mathbf{h}_{l,G}^{(j)} \neq 0, \forall i \neq l,$

(2) $\mathbf{h}_{i,L}^{(j)} \oplus \mathbf{h}_{i,L}^{(l)} \neq 0, \forall j \neq l.$

As discussed in Subsection 3.3.1, in the parity check matrix of a linear code with minimum distance $d = 4$, every $d - 1 = 3$ or a smaller number of columns must be linearly independent. To prove this, first, we note that there is no column in $\mathbf{H}_{\text{BLRC}}$ that is a vector of all zeros. Let us start with any linear combination of two columns. $\mathbf{H}_G$ in (3.8) is composed of g blocks of matrix $[\mathbf{I}_r \ \mathbf{0}_{r\times 1}]$ and each local group has one such matrix. If two columns belong to the same local group, since no two columns of $[\mathbf{I}_r \ \mathbf{0}_{r\times 1}]$ are identical, the result of the combination will not be zero (please see (1) above). If the two columns are from different localities, since in H_L different local groups have different nonzero rows, the result again will not be zero (please see (2) above). For the combination of three columns, there are three possibilities; First, the three columns belong to the same local group. In this case, due to the fact that the block $[\mathbf{I}_r \ \mathbf{0}_{r\times 1}]$ in each local group is composed of an identity matrix and a zero vector, the combination of no three columns will be zero. The second case happens when two columns are from one local group and the other column is from a different group. In this case, we know that each local group in H_L has a different nonzero row, hence the result is nonzero. The last case is when three columns are each from a different local group, in this case again, because of the same reason mentioned for the second case, i.e. different nonzero rows for each local group, the combination will not be zero. On the other hand, if we consider the first two columns of one local group and the first two columns of any other group, the combination of the four columns is zero ($\mathbf{h}_1^{(i)} \oplus \mathbf{h}_2^{(i)} \oplus \mathbf{h}_1^{(j)} \oplus \mathbf{h}_2^{(j)} = 0 \ , \forall i, j \in \{l\}_1^g, i \neq j$). This proves that the minimum distance for our proposed BLRC is $d = 4$. □

Remark 3.1. *An (n, k, d) linear code with locality r is said to be d-optimal if there exists no $(n, k, d+1)$ code with the same locality[27]. By this definition, our proposed BLRCs are d-optimal for $k \in \{k\}_1^{10}$. This can be easily verified using the online table [55]. For larger k, depending on the locality r, there are many other cases in which our codes are d-optimal (e.g., $n = 20, k = 12, r = 3$ or $r = 4$).*

Remark 3.2. *In the design of LRCs for distributed* storage *systems, satisfying the Singleton-like bound with equality is an important objective. This is because being optimal with respect to the Singleton-like bound results in the minimum storage overhead. However, satisfying this bound might not be a design objective for coded distributed* computing*. In distributed computing, reducing the time delay of the overall process is usually the main objective. As such reducing the encoding and decoding complexity of the codes might be a more important goal.*

3.3.3 Encoding with the Proposed BLRC

As discussed, to reduce the computational complexity, we use a systematic generator matrix. We use the Tanner graph From Fig. 3.1 to obtain the generator matrix. Let us separate the variable nodes in the Tanner graph of Fig. 3.1 into two groups; information symbols and redundancies. We build matrix $\mathbf{G}$ with k rows and $k+n$ columns. Each row represents an information symbol and each column shows the relation of the corresponding variable node with the information symbols. If $\mathbf{G}_{ij} = 1$, it means the i-th information symbol is present in the XOR operation that results in the j-th variable node. Hence, one can obtain $\mathbf{G}$ as:

$$\mathbf{G}_{\text{BLRC, systematic}} = [\mathbf{I}_k \; \mathbf{G}_L \; \mathbf{G}_G], \tag{3.9}$$

where $\mathbf{G}_L = [\mathbf{I}_{g-1} \; \mathbf{1}_{(g-1)\times 1}] \otimes \mathbf{1}_{r\times 1}$ and $\mathbf{G}_G = \mathbf{1}_{(g-1)\times 1} \otimes \mathbf{I}_r$ build the columns for local and global redundancies respectively.

Example 3.2. *Based on Fig. 3.2, the systematic generator matrix of the*

BLRC code in Example 3.1 is as follows:

$$\mathbf{G} = \left(\begin{array}{cccccc|ccc|ccc} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right).$$

By substituting the systematic generator matrix of the proposed BLRC into (3.3), we obtain the corresponding encoding matrix, $\mathbf{E}$. We, then use $\mathbf{E}$ in (3.2) to finish the encoding process.

To obtain the σ parameter associated with the computational cost of the encoding, we should count the number of all XOR operations between the elements of the original k blocks of the matrix $\mathbf{A}$. Each block in $\mathbf{A}$ has $\frac{m}{k}$ rows with a length of p. This means the total number of binary additions for operating an XOR between two blocks of the matrix $\mathbf{A}$ is $p\frac{m}{k}$. Based on the Tanner graph, the total number of XOR operations to obtain all the local and global redundant variable nodes is calculated as follows:

$$C_{\text{XOR encode}} = \underbrace{g(r-1)}_{\text{Local Redundancy}} + \underbrace{r(g-2)}_{\text{Global Redundancy}}, \tag{3.10}$$

so the total number of binary additions is $p\frac{m}{k}C_{\text{XOR encode}}$ and consequently the parameter σ becomes:

$$\sigma_{\text{encode, BLRC}} = \left(p\frac{m}{k}C_{\text{XOR encode}}\right)\sigma_A. \tag{3.11}$$

Here, (3.11), quantifies the relation between the code parameters and the complexity of encoding. In the next propositions, we use this relation to show how our code has the minimum encoding complexity among all well-structured BLRCs with the same parameters and that for this minimum complexity, the code rate of our proposed code is optimal.

Proposition 3.2. *Among all well-structured BLRCs with the same length n, locality r, and the minimum distance $d = 4$, our proposed BLRC requires the minimum number of* XOR*s to generate.*

Proof. Any well-structured BLRC code has the same structure as $\mathbf{H}_L$ of (3.8) in their parity check matrix $\mathbf{H}$. Let us now focus on $\mathbf{H}_G$. Reducing the number of ones in $\mathbf{H}_G$, directly reduces the encoding complexity. Our code has exactly r ones in each section of $\mathbf{H}_G$ below a local group (please see (3.8) for a local group). We claim that at least r ones are needed in each section of $\mathbf{H}_G$ under a local group. To see, this, note that with less than r ones, regardless of the number of rows in $\mathbf{H}_G$, the pigeon-hole principle states that at least two columns of $\mathbf{H}_G$ (and hence two columns of $\mathbf{H}$) are the same. This contradicts the assumption that $d = 4$. □

Theorem 3.1. *Among all the well-structured BLRCs with the same length n, locality r, minimum distance $d = 4$, and $C_{\mathrm{XOR}\,encode} = g(r-1) + r(g-2)$, our proposed BLRC has the highest code rate.*

Proof. Again, any well-structured BLRC code has the same structure as $\mathbf{H}_L$ of (3.8) in their parity check matrix $\mathbf{H}$. Hence, we only focus on $\mathbf{H}_G$. Since the code has the same $C_{\mathrm{XOR\,encode}}$ as our code, the number of ones in $\mathbf{H}_G$ of this code is the same as our code. All that we need to show is that the number of rows in $\mathbf{H}_G$, denoted by s here, cannot be less than r. Let us consider the local group with the minimum number of ones in its $\mathbf{H}_G$ rows. This minimum in the best-case scenario is r when all ones are distributed uniformly among the local groups. Now, we have r ones to be placed in a block of $r+1$ columns and s rows. To avoid two equal columns (which would contradict $d = 4$), we need to place exactly a single one in every column and leave one column as all zeros. Now If $s < r$, there will be at least two equal columns. Hence, s is at least equal to that of our code, i.e., $s = r$. □

3.3.4 Decoding the Proposed BLRC

The main motivation for using BLRCs, in this work, was to reduce the coding complexity. Hence, in this section, we put our focus on developing a decoding process that requires a small number of XORs. The challenge is that, as previously mentioned in Subsection 3.2.3, the MRT of the proposed BLRC is not deterministic. In other words, unlike MDS codes where any k coded symbols are enough to recover the k original information symbols, in our BLRC depending on the set of coded symbols received by the master, MRT varies between k and $n-3$ received symbols.

The decoding process has two separate stages. In the first stage, a blueprint for the optimal decoding process is constructed. This construction is done on a prototype matrix. This is to avoid operating on large matrices and vectors of the actual distributed problem. When the decoding blueprint is ready, the master can operate on actual large matrices and vectors, knowing that the steps require the minimum number of XORs.

Phase one of the decoding process is therefore as follows. After receiving the k-th coded block at the master, we check if the received set is decodable. If not, the master keeps receiving new coded blocks until the received set becomes decodable. This decodable set corresponds to a submatrix of $\mathbf{G}$ that is full rank. Let us call it $\mathbf{G}_{\mathcal{S}}$. Since there may be linearly dependent columns in this submatrix, we remove any global redundancy that does not affect the decodability of the set. When $g > r+1$ (i.e., the degree of global check nodes is larger than that of local check nodes), this choice ensures that we end up with a decodable set, whose corresponding submatrix of $\mathbf{G}$ has the minimum possible number of ones. This submatrix is the prototype that will be used to form the decoding blueprint. In the following, we show how to construct a decoding blueprint with the minimum possible number of XORs from this sub-matrix.

Let us call the set that the master keeps as the final decodable symbols, a necessary set. We denote this set by $\mathcal{K}$ and its corresponding submatrix of $\mathbf{G}$ by $\mathbf{G}_{\mathcal{K}} \in \mathbb{F}_2^{k \times |\mathcal{K}|}$. To construct the decoding blueprint, we start by picking up the columns of $\mathbf{G}_{\mathcal{K}}$ in the order of their corresponding Hamming weights, i.e. in the order of their number of ones, w. It is clear that the support[2] of the column with the largest w is not a subset of the support of any other column. So we take the next column with the second highest w. Without loss of generality, let us assume this is the i-th column of $\mathbf{G}_{\mathcal{K}}$ shown by $\mathbf{g}_i$.

[2]We define support of a vector $\mathbf{g}$ of length k as $\text{Supp}(\mathbf{g}) = \{i \in \{i\}_1^k | g_i \neq 0\}$, where g_i is the i-th element of the $\mathbf{g}$.

Now we check every other column in $\mathbf{G}_{\mathcal{K}}$ to find another column, $\mathbf{g}_j$, that $\mathbf{g}_i$'s support is a subset of the latter's. If such $\mathbf{g}_j$ exists, we replace $\mathbf{g}_j$ with $\mathbf{g}_j \oplus \mathbf{g}_i$ and start the process all over again. We know that each column in $\mathbf{G}_{\mathcal{K}}$ initially represents a coded symbol and the ones in each column correspond to the information symbols used to form this coded symbol. So replacing $\mathbf{g}_j$ with $\mathbf{g}_j \oplus \mathbf{g}_i$ is equivalent to canceling out the information symbols already available in $\mathbf{g}_i$. On the other hand, if we are not able to find $\mathbf{g}_j$ such that $\text{Supp}(\mathbf{g}_i) \subset \text{Supp}(\mathbf{g}_j)$, we move on to the column with the third highest w and so on. We repeat the same procedure on $\mathbf{G}_{\mathcal{K}}$ in an iterative manner until all the redundant symbols are decomposed into their building information symbols, i.e., $\mathbf{G}_{\mathcal{K}}$ has a single one in each of its columns. A look-up table of all the XOR operations performed above is the decoding blueprint that the master must follow to finish the decoding. The complexity of obtaining this blueprint (i.e., running Algorithm 1) is ignored in this work because it involves vectors and matrices that are significantly smaller than the actual data vectors and matrices. Please also note that the rank of $\mathbf{G}_{\mathcal{K}}$ in the binary field is k. As can be deducted from the above discussion, since our algorithm starts from the column with the highest Hamming weight and goes down, it cancels out the maximum possible number of overlapping information symbols in other columns at each step.

The decoding algorithm is presented below where $\oplus$ shows the binary addition (equivalent to the XOR) of two columns:

Algorithm 1 Decoding for the Proposed BLRC

```
1:  Result: obtaining the number of XORs in the decoding process, recovering $k$ information symbols
2:  Initialization: XOR_counter=0
3:  Input: $\mathbf{G}_{\mathcal{K}}$
4:  while change occurs in $\mathbf{G}_{\mathcal{K}}$ do
5:      for $w = k, k-1, \ldots, 1$ do
6:          for $i = 1, 2, \ldots, |\mathcal{K}|$ do
7:              if Hamming weight $\mathbf{g}_i = w$ then
8:                  for $j = 1, 2, \ldots, |\mathcal{K}|, j \neq i$ do
9:                      if Supp($\mathbf{g}_i$) $\subset$ Supp($\mathbf{g}_j$) then
10:                         $\mathbf{g}_j \leftarrow \mathbf{g}_j \oplus \mathbf{g}_i$
11:                         XOR_counter $\leftarrow$ XOR_counter + 1
12:                         Break all the loops and start the outmost loop again
13:                     end if
14:                 end for
15:             end if
16:         end for
17:     end for
18: end while
```

Proposition 3.3. *Given a necessary set $\mathcal{K}$, Algorithm 1 recovers all the missing information symbols with the minimum number of XOR operations*

Proof. $\mathbf{G}_{\mathcal{K}}$ in Algorithm 1 is composed of zeros and ones. The goal of the algorithm is to convert this matrix into a matrix whose columns have at most a single one. Since our algorithm removes the maximum number of ones per XOR, it requires the minimum number of XORs. □

Let us clarify Algorithm 1 with an example.

Example 3.3. *Consider the matrix* $\mathbf{A}$ *divided into tree blocks as below:*

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \mathbf{A}_3 \end{bmatrix}.$$

Let us use generator matrix $\mathbf{G}$ *defined below to encode these blocks:*

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

Doing so, we will obtain the coded matrix $\bar{\mathbf{A}}$ *as below:*

$$\bar{\mathbf{A}} = \begin{bmatrix} \boxed{\bar{\mathbf{A}}_1 = \mathbf{A}_1} \\ \boxed{\bar{\mathbf{A}}_2 = \mathbf{A}_1 \oplus \mathbf{A}_2} \\ \boxed{\bar{\mathbf{A}}_3 = \mathbf{A}_1 \oplus \mathbf{A}_2 \oplus \mathbf{A}_3} \\ \boxed{\bar{\mathbf{A}}_4 = \mathbf{A}_1 \oplus \mathbf{A}_3} \end{bmatrix}.$$

For the purpose of simplicity, we avoid the multiplication of the coded matrix with target vectors. Now consider a case where only the first three coded blocks have arrived at the master. This means the master will have access to the first three columns of $\mathbf{G}$*. To recover* $\mathbf{A}_1$*,* $\mathbf{A}_2$*, and* $\mathbf{A}_3$*, the master uses Algorithm 1 to make a decoding blueprint. Since* $k = 3$*, the master will look for a column of G with a Hamming weight (from now on just weight) of* $\omega = 3$ *which is the third column,* $\mathbf{g}_3$ *(lines 5, 6, and 7 in Algorithm 1). It is obvious that the support of* $\mathbf{g}_3$ *is not the subset of any other column (line 9). So the master will look for a column with* $\omega = 2$*, i.e.* $\mathbf{g}_2$ *(lines 5, 6, and 7). Now it will check if there is a column that the support of* $\mathbf{g}_2$ *is a subset of the support of that column. In this example, Supp*$(\mathbf{g}_2) \subset$ *Supp*$(\mathbf{g}_3)$ *(line 9). The master then will replace* $\mathbf{g}_3$ *with* $\mathbf{g}_3 \oplus \mathbf{g}_2$*. Hence the updated* $\mathbf{G}$ *will be:*

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

and by applying this step of the decoding blueprint on $\bar{\mathbf{A}}$*, we get the updated* $\bar{\mathbf{A}}$*:*

$$\bar{\mathbf{A}} = \begin{bmatrix} \boxed{\bar{\mathbf{A}}_1 = \mathbf{A}_1} \\ \boxed{\bar{\mathbf{A}}_2 = \mathbf{A}_1 \oplus \mathbf{A}_2} \\ \boxed{\mathbf{A}_3} \end{bmatrix}.$$

Because a change happened in $\mathbf{G}$ *the master starts the algorithm from line 5, setting* $\omega = 3$*. After similar steps, the algorithm discovers that Supp*$(\mathbf{g}_1) \subset$ *Supp*$(\mathbf{g}_2)$ *and replaces* $\mathbf{g}_2$ *with* $\mathbf{g}_2 \oplus \mathbf{g}_1$*. This step will end the algorithm as the updated* $\mathbf{G}$ *will be:*

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

which is the identity matrix. Following the blueprint operation, $\bar{\mathbf{A}}$ *will also* *change to:*

$$\bar{\mathbf{A}} = \begin{bmatrix} \bar{\mathbf{A}}_1 = \mathbf{A}_1 \\ \mathbf{A}_2 \\ \mathbf{A}_3 \end{bmatrix},$$

which concludes the decoding.

Based on the above algorithm, it becomes clear that depending on the received symbols, the number of XORs required to recover the original k information symbols is different. Hence, to find the σ parameter associated with the decoding process we need to find the average number of XOR operations. The simulation to find the average number of XORs will be discussed in Subsection 3.4.3. Let us assume that the average number of XOR operations for a decoding scenario is $C_{\text{XOR decode}}$. Since each received coded block in the matrix-vector multiplication has $\frac{m}{k}$ rows and there are N vectors to be multiplied the σ parameter is calculated as:

$$\sigma_{\text{decode, BLRC}} = \left(N\frac{m}{k}C_{\text{XOR decode}}\right)\sigma_A. \tag{3.12}$$

Now that we have all the σ parameters associated with three phases of encoding, computation, and decoding for both MDS and BLRC schemes, we can obtain the average time delay of these three phases for each scheme and compare them. The total time delay of the distributed scheme is the sum of the delays in different phases.

3.4 Experiments and Numerical Results

3.4.1 Modelling Task Completion

As was discussed in Subsection 3.2.2, we use shifted exponential distribution to model the time delay for completing a task and denote it by the random variable T. The shifted exponential distribution is scaled with a parameter σ which is proportional to the computational cost of the task. So far, we have calculated σ for encoding, decoding, and each of the partial multiplication tasks that workers perform in MDS and BLRC frameworks. In our system model, the encoding and decoding tasks are done by the master. As we mentioned before, we consider two possible behaviors for the master. In the case that its performance is stable, the time delay incurred by the encoding and decoding is a constant value equal to σ_{encode} and σ_{decode} respectively. Otherwise, the master is assumed to show straggling behavior. This means to find the average time delay τ in each of these phases, one needs to find $\mathbb{E}[T_{\text{encoding}}]$ and $\mathbb{E}[T_{\text{decoding}}]$ where random variables T_{encoding} and T_{decoding} have a CDF as (3.1) with parameters σ_{encode} and σ_{decode} respectively. We will obtain this average time delay by the Monte Carlo Method.

In the computation phase, n workers with the same computational capacity start computing tasks with σ parameter equal to (3.4), simultaneously. On the other hand, we know that in an MDS framework, as soon as any k subtasks are completed, the computation process is over and the original task can be recovered. Hence, if we denote the time delay of completing subtask $i \in \{i\}_1^n$ by T_i, where T_i follows the CDF in (3.1), the average time delay of the whole computation phase would be the expected value of $T_{(k)}$ that represents the k-th order statistic of T_i. Hence,

$$\tau_{\text{computaion}} = \mathbb{E}[T_{(k)}] = \sigma_{\text{computation}} \left(1 + \sum_{j=n-k+1}^{n} \frac{1}{j} \right). \tag{3.13}$$

For the numerical results, we generate n random variables all following the shifted exponential distribution to represent the time delays of the n workers. Then we find the time delay corresponding to the k-th fastest worker. We repeat the above procedure until we have a reliable average of all the instances as an approximation to the average time delay of the computation phase $\tau_{\text{computation}}$. In the BLRC framework, however, the MRT is not k and is not even a fixed number as it depends on the received set. Since there are numerous possibilities in the order of receiving the subtasks, we find the average time delay of the computation phase for the BLRC scheme through numerical simulations. Like the case for MDS, we create a scenario in which we assume each of the n workers is assigned a subtask whose completion time is a generated random variable following a shifted exponential distribution. As a result, the master receives subtasks from the workers in an order determined by these random variables. The master stops the process immediately after it has a decodable set by sending a message to all workers. The completion time of the

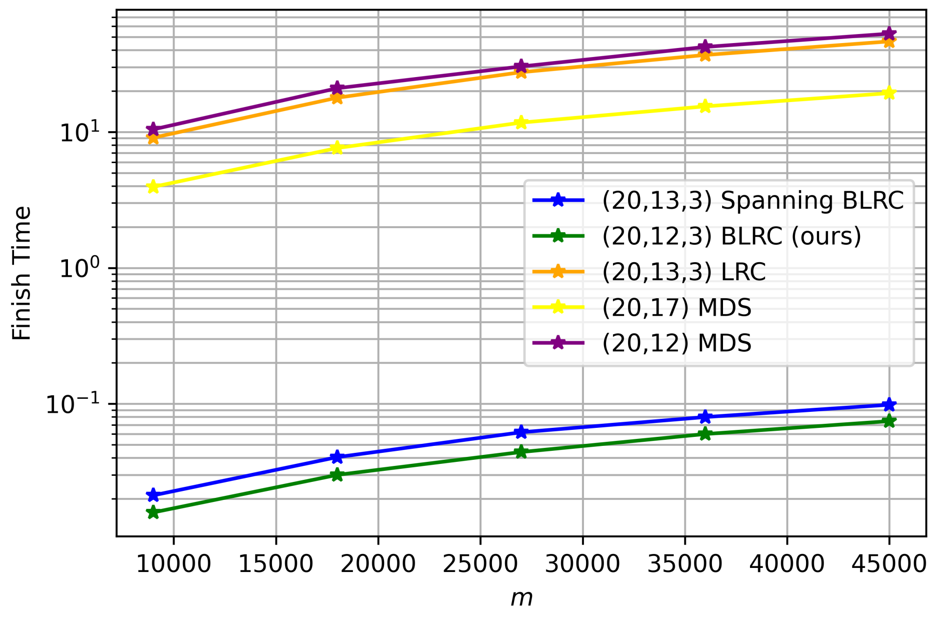


Figure 3.3: Comparison of the encoding time (in sec.) between (20,13,3) Spanning BLRC[59], our BLRC, (20,13,3) LRC[66], and two MDS codes

last subtask received is recorded as the computation time of this realization. We repeat this for many realizations until we have a reliable average for the $\tau_{\text{computation}}$.

3.4.2 Encoding Cost for the Proposed BLRC

In order to show the performance of our proposed BLRC for the encoding time delay and compare it with some of the LRCs and BLRCs that are optimal with respect to Singleton-like bound, the encoding time for a matrix, $\mathbf{A} \in \mathbb{F}_{2^l}^{m\times p}$, for $p = 900$ and a varying range of m is measured and depicted in Fig. 3.3. In this figure, the (20,17) MDS code has the same $d = 4$ as our code and the (20,12) MDS has a similar code rate to our code. It is clearly seen that our proposed BLRC has the lowest encoding time among all.

3.4.3 Decoding Cost for the Proposed BLRC

In Subsection 3.3.4 we described the decoding procedure for the proposed BLRC. We also mentioned that based on the set of received symbols the number of required XOR operations may vary. As a result, we should use the

Table 3.1: Decoding operations for an MDS code and the proposed BLRCs

Code	Additions/XORs	Multiplications
(20,12,3) BLRC	194.803	-
(20,12,4) BLRC	194.701	-
(20,12) MDS	4484	783
(20,17) MDS	3165	552

average number of required XORs in (3.12). To find the average number of XORs required we set up a scenario in which a set of coded symbols arrive at the master in a randomly generated order. Assuming the decoding process described in Section 3.3.4, we find the average number of XORs by the Monte Carlo method. Table 3.1 contains a comparison between the number of operations in the decoding of multiplying an example matrix $\mathbf{A} \in \mathbb{F}_{2^l}^{204\times 204}$ with only one vector where the matrix is coded with the proposed BLRCs and two MDS codes, one with the same code rate and the other with the same d.

The simulation results in Table 3.1 show that using the proposed BLRCs instead of MDS codes, not only has removed the costly multiplication operations but also has reduced the large number of additions required by MDS codes to just a few XORs. In the next subsection, we will see the effect of such a significant reduction in the number of operations on the time delay of the encoding and decoding phases.

3.4.4 Simulation Results for the Overall Process

In this subsection, we provide the time delay incurred in each of the three phases of a coded distributed computing problem that is solved based on the proposed BLRC scheme. For comparison, we also provide the same delay information for a scheme that uses an MDS code. Finally, we will compare the overall time delay of the two schemes.

We consider the multiplication of matrix $\mathbf{A}$ with $m = 30600$ rows and $p = 2040$ columns to $N = 200$ vectors of length p. Here m and p are chosen to be a multiple of 12 and 17 in order to make direct comparisons with MDS codes of the same length and with the same minimum distance or code rate possible. More specifically, we use two BLRC schemes with $k = 12$ and localities $r = 3$

and $r = 4$. This means in BLRC schemes, we initially divide matrix $\mathbf{A}$ into 12 blocks. The code length in both cases is equal to 20. We compare our $(n, k) = (20, 12)$ BLRCs with an MDS code that has the same length and rate. In other words, it is a linear $(20, 12)$ MDS code. We also compare our BLRCs with a (20,17) MDS code that has the same length and minimum distance as ours but a higher code rate. The minimum distance, d, of an MDS code is equal to $n - k + 1$. As a result, for an MDS code with $n = 20$ to have $d = 4$ the k should be equal to 17 which means for this case the $\mathbf{A}$ is initially divided into 17 blocks.

As discussed before, two phases namely, the encoding and decoding, that are done by the master are simulated in two ways. First, with the assumption that the master has a straggling behavior similar to each of the workers, and second, with the assumption that the master has a steady behavior. The results of our simulations are presented in Table 3.2 and Table 3.3 for a master with a probability of straggling behavior and for a non-straggling master, respectively.

From the results in Table 3.2 we can see that although the time to complete the computation phase for the (20,12) MDS is lower than the (20,17) MDS code, the second code has generally a better performance with a lower overall time delay. Comparing the BLRCs to the (20,17) MDS, the time to complete the encoding phase has dropped from 29.308 s to only a fraction of a second for BLRCs resulting in more than 99.6% reduction in the encoding time delay for either of the BLRCs. The same reduction, even in greater magnitude happens in the decoding phase where using our proposed BLRCs has reduced the time delay from 7.438s for the (20,17) MDS code to only 0.007s corresponding to a 99.9% decrease. However, in the computation phase, the simulation results for the BLRCs show approximately a 10% and 8% increase in the time delay compared to the (20,12) and (20,17) MDS codes, respectively. This is the sacrifice we make by not using a code with the minimum recovery threshold such as an MDS code. The simulation results also show that the significant reductions in the time delay of the encoding and decoding phases, not only completely compensate for the small increase in the time delay of the computation phase, but

Table 3.2: Time delay (in sec.) of the BLRC and MDS schemes in different phases under the assumption of a straggling master

Code	Encoding	Computation	Decoding	Overall
(20,12,3) BLRC	0.106	171.452	0.007	171.566
(20,12,4) BLRC	0.107	171.259	0.007	171.373
(20,12) MDS	77.468	155.488	10.554	242.995
(20,17) MDS	29.308	157.685	7.438	194.057

Table 3.3: Time delay (in sec.) of the BLRC and MDS schemes in different phases for a non-straggling master

Code	Encoding	Computation	Decoding	Overall
(20,12,3) BLRC	0.055	171.452	0.003	171.510
(20,12,4) BLRC	0.055	171.259	0.003	171.317
(20,12) MDS	40.237	155.488	5.463	201.188
(20,17) MDS	15.091	157.685	3.856	176.632

cause a notable 13.10% reduction in the overall time delay of the distributed computation task compared to the best-performing MDS code. In addition, results in Table 3.3 for a non-straggling master, point to an approximately 3% reduction in the overall time delay of the process using BLRCs instead of the (20,17) MDS code. This shows regardless of the master's behavior, the proposed BLRC scheme outperforms the MDS counterparts.

In order to better understand the importance of the BLRC scheme in reducing the delay of the encoding and decoding phases we simulate the matrix-vector multiplication for different numbers of vectors (N) that are being multiplied by the matrix $\mathbf{A}$. In Fig. 3.4, the overall time delays, τ_{overall}, of MDS and BLRC schemes have been demonstrated for matrix-vector multiplication. The codes and assumptions are the same as TABLE 3.2. Because of their similar performance, only one BLRC (the BLRC code with $r = 3$) is compared to MDS codes. The matrix $\mathbf{A}$ used for the performance evaluation has $m = 10200$ rows and $p = 1020$ columns. We change the number of vectors from $N = 1$ to $N = 1000$. We can see that for the entire range of N the overall time delay of the BLRC scheme is below its MDS counterparts. However, for smaller N, the difference is much larger. For example for multiplying a single vector ($N = 1$), using the proposed BLRC reduces the overall time delay by 96.81%

compared to the (20,17) MDS code. For moderate N, the performance of the BLRC scheme is still much better than the MDS. For $N = 64$ vectors which is a number in the range commonly used for batch size in the gradient descent algorithm with extensive applications in machine learning and optimization in general, the overall time delay of the BLRC code is more than 35% lower than the best-performing MDS code. It means a distributed optimization process using BLRC is completed in only two third of the time it takes for an MDS scheme. For larger N the difference in the time delay shrinks and the gap between the curves closes for extremely large values of N. The reason for this behavior is that, unlike $\sigma_{\text{computation}}$ and σ_{decode}, σ_{encode} in BLRC and MDS schemes is independent of N. Hence, when N is small, the dominant term in the overall time delay of the MDS schemes is the delay for encoding. The same should have happened to the BLRC scheme but since the encoding delay has already been reduced by orders of magnitude, it does not dominate the overall time delay. As N increases, the time delay in two other phases becomes comparable with the encoding phase of the MDS schemes. For large values of N, the dominant term in the time delay belongs to the computation phase and the curves approach each other. Yet, the lower delay in the decoding phase of the BLRC compared to MDS schemes keeps the overall delay of the former below the latter.

To better demonstrate the effect of using a BLRC scheme on reducing the master's workload, in Fig. 3.5, we present the combined time delay of the encoding and the decoding phases for different N. The figure clearly shows that through the entire range of N, the BLRC scheme has a significantly lower time delay compared to the MDS schemes. In fact, for different N, a master that uses a BLRC code can finish its task in approximately 0.3% of the time a master using the best MDS code will finish. In most of the applications, since the master is not on the cloud, reducing its workload is of particular importance, hence using our proposed BLRC will be extremely advantageous over MDS codes.

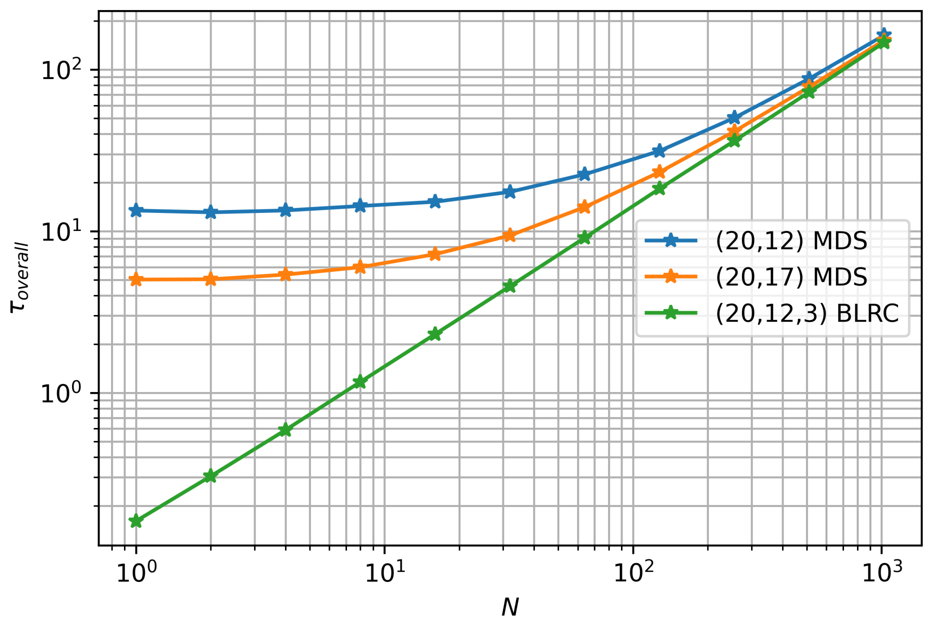


Figure 3.4: The overall time delay (in sec.) of the BLRC and MDS schemes for different N.

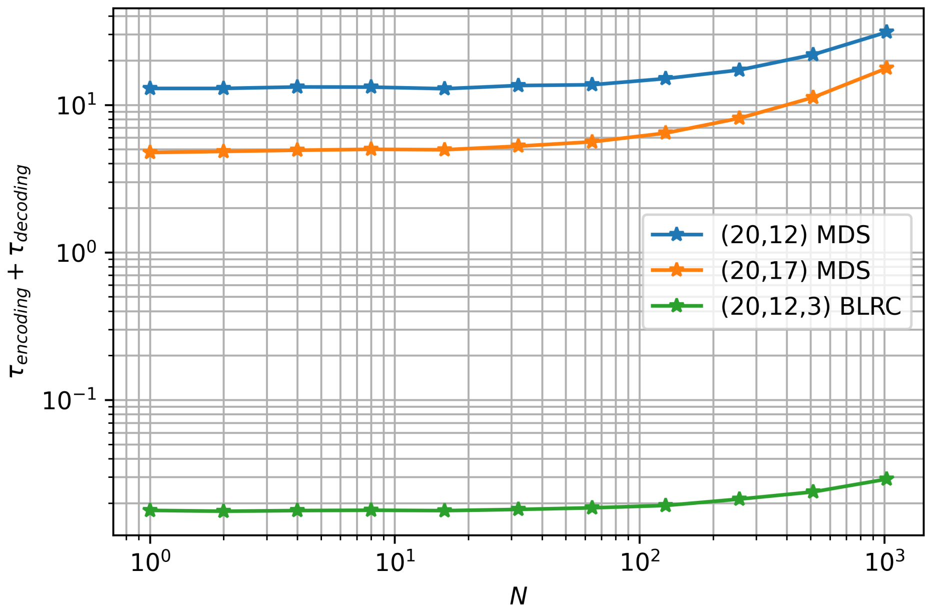


Figure 3.5: The combined time delay (in sec.) of the encoding and decoding phases (the master's time delay) in the BLRC and MDS schemes for different N.

3.5 Conclusion

In this work, for the first time, we proposed the use of BLRC codes for coded distributed computing systems. We introduced a new family of BLRC codes specifically designed for this purpose. By significantly reducing the encoding and decoding complexity, our proposed BLRCs reduce the overall time delay compared to solutions based on widely adopted MDS codes. Unlike MDS codes, our BLRC has two types of redundancies to recover the missing blocks: Local redundancies and global redundancies. This structure, in addition to the binary nature of the code that eliminates a need for multiplications, significantly reduces the encoding/decoding complexity.

We showed that among all BLRCs with a widely adopted local structure (called well-structured BLRCs in this work) and the same minimum distance, our proposed codes require the minimum number of XOR operations for the encoding. We also proved that it had the highest code rate among well-structured codes with the same encoding complexity. In addition, we suggested an efficient decoding algorithm for the proposed codes that guarantees the minimum number of XOR operations for the decoding process. Through extensive experiments, we showed the success of the proposed BLRCs in significantly reducing the master's workload during the encoding and decoding, as well as a lower overall time delay compared to MDS codes.

Chapter 4

Coding for a Multi-task Distributed Computing System

4.1 Introduction

The straggler problem in distributed computing has motivated solutions that encode the tasks before distributing them among the worker nodes, giving the system the ability to recover the results of the straggling workers [36]. A common model for distributed computing systems is the master-worker model. In the coding schemes proposed for the master-worker model, the master divides the main task into smaller subtasks and encodes them, increasing the number of total subtasks due to the introduced redundancy. In the conventional schemes, the master allocates a single subtask to each of the workers, meaning that the number of encoded subtasks should not exceed the number of workers. However, this type of task allocation makes the system prone to over-computation and under-utilization.

For instance, if a worker finishes its task much earlier than the others, it remains idle till the master finalizes the computation, or if all the workers finish their subtasks at almost the same time, many redundant computations will be carried out without actually being needed to finalize the computation. Another drawback of the conventional schemes is the limited straggler tolerance. Error correction codes can only recover a limited number of erasures. Hence, in a distributed computing system where the subtasks are encoded with error correction codes, if the number of stragglers for any reason exceeds the number of erasures that the code can recover, the master will not be able to complete the main task. To address the above-mentioned issues, multi-message communication (MMC) model has been proposed [2], [20], [33], [43].

In the MMC model for distributed computing, each worker receives more than one subtask and is able to send the master multiple messages. More specifically, the master assigns a series of subtasks to each worker and the workers execute their subtasks in the order they are received one at a time. Moreover, the workers do not wait for the completion of all their subtasks to report the results. Instead, they send the result of computing each subtask as soon as it is finished. Various schemes have been proposed for the MMC model and it has been deployed in many large-scale distributed computing problems [2], [20], [33], [43]. As an example, for a distributed matrix multiplication, [33] divides the main task into a number of subtasks larger than the number of workers, encodes them with a maximum distance separable (MDS) code, and assigns multiple subtasks to each worker. Since the subtasks are much smaller than the single-task case, even the stragglers manage to compute some of these subtasks, hence, being exploited in the overall process.

Although the MMC model reduces the computation time compared to the single-task case, the larger number of subtasks in this model increases the encoding and decoding time. The scheme in [2] considers this issue in distributed matrix multiplication and proposes a scheme based on systematic MDS codes. Similar to [33], they divide the main task into a number of subtasks larger than the number of workers. However, due to the use of systematic MDS codes, the encoding time is improved. More importantly though, when the coded subtasks are assigned to the workers, the information subtasks are prioritized over the redundant subtasks. Considering the fact that tasks are executed in sequential order, this form of subtask assignment will make sure that among the completed subtasks a larger number are information subtasks. Completed information subtasks do not need to be decoded and the master only decodes a few redundant subtasks, hence, compared to a random subtask assignment, the decoding time is greatly reduced as well.

The scheme in [2] is promising but it does not answer some important questions about the subtask assignment that are crucial to minimizing the computation time. The computational load of the subtasks, the number of information subtasks, and the total number of coded subtasks that are assigned to each worker are factors that play an important role in the computation time of a task. However, [2] assumes a fixed number of information subtasks and entirely focuses on the order of assigning them to the workers. In fact, the optimal value for quantities such as the number of information subtasks, and the total number of coded subtasks strongly depends on the characteristics of the distributed nodes such as their computational power as well as the computational complexity of the subtasks.

In this work, we adopt the MMC model and propose a new task assignment scheme based on systematic MDS codes. In our scheme, each worker receives a total number of coded subtasks equal to the number of information subtasks, where these subtasks are executed in order. Our scheme avoids any presupposition on the number of information subtasks. Instead, we model the computation time in each worker and obtain the probability of completing the overall computation before a deadline. Then we determine the number of information subtasks and consequently the computation load of each coded subtask as well as the total number of coded subtasks assigned to each worker to maximize the probability of meeting a specific deadline or to minimize the expected completion time. Although our model assumes that the total number of subtasks assigned to each worker is equal to the number of information subtasks, our probabilistic analysis also gives us the ability to calculate the probability of completing a subtasks of a specific priority in each of the workers before the main computation is finalized. In cases that the encoding time is a bottleneck one can accept a Small sacrifice in computation time and avoid creating and assigning redundant subtasks to the priorities that have a less completion probability. Hence, our analysis also allows for adjusting the trade-off between the number of redundant subtasks required for optimal performance and the encoding time. Since in our scheme, the optimum value of the number of information subtasks can be more than, equal to, or less than the number of available workers, i.e. each worker may receive more than a single, a single, or no information subtasks, we name our proposed scheme hybrid coded distributed computing (HCDC).

The contributions of this work can be summarized as follows:

- We propose a new coded distributed computing scheme based on systematic MDS codes, called HCDC, for the MMC model that addresses the over-computation and under-utilization problems that exist in conventional schemes.
- We model the computation time in the workers with multiple tasks and use this model to obtain the probability of completing the overall computation process before a deadline in the proposed HCDC scheme.
- Based on this probabilistic model, we derive the optimal number of information subtasks as well as the computation load of the coded subtasks that maximize the probability of finishing the overall computation before a deadline or minimize the expected time of the overall process.
- We provide a probabilistic method to systematically adjust the trade-off between the number of redundant subtasks assigned to each worker and the encoding time such that the performance is closer than a specified target to the optimal performance in a probabilistic sense.

4.2 System Model

Consider a master-worker distributed computing system with N workers. Similar to [20], we consider a general distributed computation problem, where the goal of the system is to compute a task $g(x)$ and x is the input. It is assumed that $g(x)$ is divisible into k subtasks $g(x) = \phi(g_1(x), g_2(x), \ldots, g_k(x))$ where $\phi(\cdot)$ simply maps the subtasks into the final result, i.e., $g(x)$. We also assume that the subtasks are linear, i.e. $ag_i(x) + bg_j(x) = (ag_i + bg_j)(x)$. One example can be matrix-vector multiplication $g(\mathbf{x}) = \mathbf{A}\mathbf{x}$ with the i-th task being $g_i(\mathbf{x}) = \mathbf{A}_i\mathbf{x}$, where $\mathbf{A}_i$ is the i-th submatrix in the horizontally decomposed $\mathbf{A}$. In this example $\phi(\cdot)$ simply concatenates the result of the tasks $\{g_i(\mathbf{x})\}_1^k$ to obtain $g(\mathbf{x})$.

In a coded distributed computing scheme, the master divides the main task into a set of subtasks $\{g_i(x)\}_1^k$, from now on called information subtasks and denoted by $g_i,\ i \in \{i\}_1^k$ for simplicity, and encodes them into a new set of n, $n > k$, coded subtasks to compute them with the help of the workers. In this work, we only consider systematic MDS codes. MDS codes have two nice features that make them an appropriate choice when the focus is on minimizing the computation time. Firstly, they have the minimum overhead, meaning that to achieve a specific minimum distance d, an MDS code requires the smallest possible number of coded subtasks, i.e. $n = k + d + 1$. Secondly, they have the minimum recovery threshold, meaning that receiving any k coded subtasks is enough to recover the information subtasks, hence finishing the original computation. At the same time, being systematic will help to keep the encoding and decoding time low.

In summary, the master uses a systematic $(n, k),\ k < n$, MDS code to create n coded subtasks:

$$\mathbf{h} = \mathcal{E}(\mathbf{g}),$$

where $\mathbf{g} = [g_1, g_2, \ldots, g_k]$ shows the information subtasks, $\mathbf{h} = [h_1, h_2, \ldots, h_n]$ shows the coded subtasks and $\mathcal{E}$ denotes the encoding operation. Also, since the code is systematic $h_i = g_i$, for $1 \leq i \leq k$. The other $n - k$ subtasks are composed of the linear combination of the information subtasks and if they are among the first k subtasks that are completed, the master needs to perform a decoding step to finalize the computation. These subtasks are introduced as redundancy to account for the straggling workers, hence, from now on we will refer to them as redundant subtasks.

There are two models for task assignments to workers, described below:

(i) Single-task model: In conventional coded schemes, the master assigns each worker a single coded subtask, hence $n = N$. The workers start their computation upon receiving their subtasks and immediately report the result when they are finished. The master can start recovering the main task as soon as it receives any set of the k coded subtasks. Fig. 4.1 shows the distribution of the subtasks among the workers in a single-subtask model.

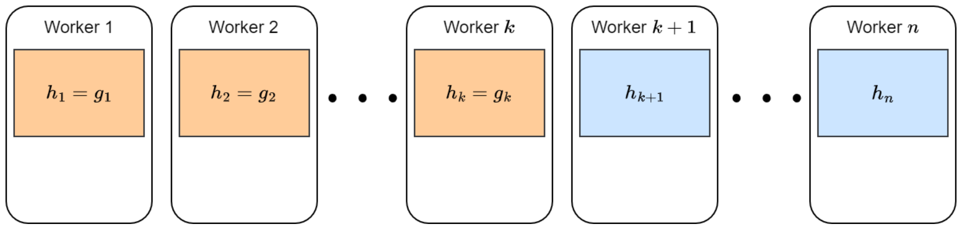


Figure 4.1: Distribution of subtasks in a single-subtask model. In this figure, the red color is used for information subtasks and the blue color for redundant subtasks

(ii) MMC model: The main difference between the MMC model and the single-task model is that in the MMC model, the workers are not limited to a single subtask. Let us assume the master wants to assign p tasks to each of the workers. Similar to the single-task model, the master divides the main task into k subtasks and uses a (n, k) systematic MDS code. However, here $n = pN$, where N is the number of workers. When $p = 1$ the MMC model simplifies to the single-task model.

The master assigns p coded subtasks to each worker in a specific order. Each worker computes its subtasks in the order given to it. In addition, after completing each subtask, the worker sends the result to the master and does not wait for all of the subtasks to finish. The benefit of MMC, of course, is that some workers may not even need to finish all of their assigned subtasks. Moreover, even some straggling workers may finish a number of their subtasks and contribute to the overall computation. In the single-task model, a worker was either completely useful or completely useless.

In the MMC model, similar to the single-task model, the arrival of any k results, enables the master to start the decoding process and finalize the computation. If the code is systematic, it is important that the information subtasks are assigned as the first subtasks to the workers. This way, a larger number of the k arrived results are information subtasks, hence, the decoding complexity is reduced.

As previously mentioned, the MMC model is advantageous over the single-task model because of solving the over-computation and under-utilization problems. This is especially important when there is a probability of persistent straggling behaviour. However, previous task assignment schemes proposed for the MMC model presume k and consequently the load of the coded subtask which makes their task assignment schemes sub-optimal in terms of the computation time. Hence, there is a need for a novel task-assignment scheme for the MMC model.

4.3 HCDC scheme

HCDC is a coded scheme designed for the MMC model. While in existing coded MMC approaches, p and k are assumed as given numbers, in HCDC we find the optimal values of p and k to meet various senses of optimality.

To work our way through this process, let us start by assuming $p = k$, i.e., k coded subtasks to be assigned to each worker. Later in Subsection 4.4.5 we will discuss that if the encoding time is a bottleneck it is possible to trade the number of the coded subtasks assigned to each worker for a lower encoding time, i.e accept some sacrifice in the computation time by assigning $p < k$ coded subtasks to each worker to have a better encoding time. The motivation for taking $p = k$ at this stage is that by assigning k subtasks to each worker it is guaranteed that even if all the workers except one of them permanently fail and do not complete any of their subtasks, the main computation still can be completed. Please note that assuming $p > k$ (assigning more than k subtasks to each worker) is not helpful because as soon as any worker completes its k-th subtask, the master can finalize the main computation.

In summary, the code employed in HCDC scheme is a (n, k) systematic MDS code where $n = k \times N$ and k is a variable obtained by optimizing for the minimum expected computation time or for the maximum probability of completion before a given deadline. Please note that k is the parameter that determines the computation load of the coded subtasks. More specifically, if we assume that the computation load of the main task, $\mathbf{g}$, is L and it is divided to k equally sized linear subtask, the computation load of the subtasks is $\frac{L}{k}$. The encoding does not change the computation load of the subtasks, hence, all the coded subtasks will also have a computation load of $\frac{L}{k}$. Thus k is one

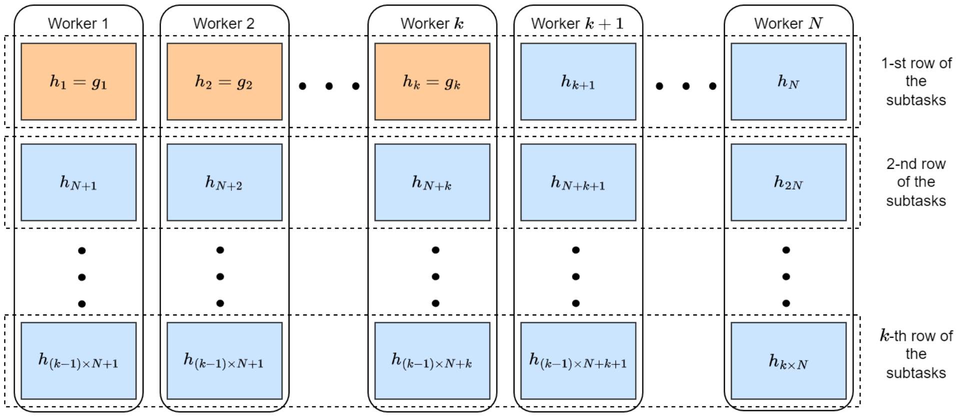


Figure 4.2: Distribution of subtasks in an HCDC scheme with the assumption that $1 < k < N$. In this figure, the red color is used for information subtasks and the blue color for redundant subtasks

of the factors that affect the finishing time of each subtask. Consequently, choosing k plays an important role in the overall computation time.

The master starts by assigning the first information subtask to the first worker, the second to the second worker, and so forth. If $k > N$, after assigning one information subtask to each of the workers, the master assigns the remaining information subtask in a similar way starting from the first worker again. When the master runs out of information subtasks, it continues the assignment with redundant subtasks. In the HCDC scheme, since k is determined by optimizing for the expected computation time or the probability of completion before a deadline, it is not necessarily a multiple of the number of workers N. As a result, it is possible that in one case, some workers have one information subtask but the others do not have any while there might be another case where all the workers have at least one information subtask, but some of them are assigned with two. Hence, we have chosen the name hybrid coded distributed computing.

Figure 4.2 illustrates the HCDC scheme. In this figure, the subtasks that are assigned with the same priority across the workers have been separated with a dashed rectangle. We will refer to all the subtasks assigned as the i-th priority across the workers as subtasks of the i-th row. please note that row here is just an abstract term used for convenience in reference.

4.4 Finishing Time Analysis

In this section, we lay out the probabilistic analysis of the finishing time in the HCDC scheme.

4.4.1 Finishing Time Distribution of a Single Task

In the literature, the finishing time of a task by a worker is usually modeled by a shifted exponential distribution[36], [39], [56]. If there is a homogeneous set of workers each with a task of the same computational load as others, the common approach is to model it with independent identical distributions (iid) that have the following cumulative distribution function (CDF):

$$\Pr(\tau \leq t) = F_\tau(t;\mu,L) = \begin{cases} 1 - e^{-\frac{1}{\mu}(t-L)}, & \text{for } t \geq L \\ 0, & \text{otherwise} \end{cases}, \tag{4.1}$$

where τ is the random variable denoting the finishing time of a single task. In (4.1) L is the computation load, thus it is the minimum time required to complete the task and μ reflects the average of all the other delays in addition to the minimum completion time of the task. In other words, L controls the shift in the shifted exponential distribution and μ controls the tail.

4.4.2 Finishing Time of A Number of Tasks by a Worker

In our setting described for the HCDC scheme in Section 4.3 we assigned more than one subtask to each worker. As such, we are interested in the distribution of the completion time for a specific number of subtasks by a worker. Let us derive the probability that s subtasks are finished before time t in a worker. We denote the completion time of the subtasks by $\{\tau_i\}_{i=1}^{s}$. Each worker executes its tasks one after another in a chronological order. This means that the probability of finishing s subtasks is equivalent to:

$$\Pr(\tau_{s\,\text{subtasks}} \leq t) = \Pr(\sum_{i=1}^{s} \tau_i \leq t)$$

Calculating the above probability would be a daunting task. Hence, we will use properties of the moment generating functions (MGF). MGF of a shifted exponential random variable with a CDF as stated in (4.1) is equal to:

$$M_\tau(u) = \mathrm{E}\left[e^{u\tau}\right] = e^{Lu}\left(1-\mu u\right)^{-1},\ u \leq \frac{1}{\mu}. \tag{4.2}$$

If some random variables are independent, the MGF of their summation is equal to the multiplication of their individual MGFs:

$$\begin{aligned} M_{\sum_{i=1}^{s}\tau_i}(u) &= \prod_{i=0}^{s} M_{\tau_i}(u) = \prod_{i=0}^{s} e^{Lu}\left(1-\mu u\right)^{-1} \\ &= e^{sLu}\left(1-\mu u\right)^{-s},\ u \leq \frac{1}{\mu} \end{aligned}. \tag{4.3}$$

Once we obtain the MGF of the distribution of the finishing time of s subtasks in a worker we immediately notice that it has the general form of the MGF of a shifted Gamma distribution. After some mathematical calculations, we obtain its probability distribution function as follows:

$$f_{\tau_{s\,\text{subtasks}}}(t; s, \mu, L) = \begin{cases} \frac{(t-sL)^{s-1} e^{-\frac{1}{\mu}(t-sL)}}{\mu^s \Gamma(s)}, & \text{for } t \geq L \\ 0, & \text{otherwise} \end{cases},$$

and as a result the probability of finishing s subtasks in a worker by time t is equal to the CDF value of the above distribution at that point:

$$\begin{aligned} \Pr(\tau_{s\,\text{subtasks}} \leq t) &= F_{\tau_{s\,\text{subtasks}}}(t; s, \mu, L) \\ &= \begin{cases} \frac{\gamma\left(s, \frac{t-sL}{\mu}\right)}{\Gamma(s)}, & \text{for } t \geq sL \\ 0, & \text{otherwise} \end{cases}, \end{aligned} \tag{4.4}$$

where the $\gamma(\cdot)$ and the $\Gamma(\cdot)$ are the the lower incomplete gamma and gamma functions respectively.

4.4.3 Finishing Time Distribution of the Main Task

Since an (n, k) MDS code is used, if k out of n subtasks are finished, the master will be able to recover the main task. Hence, We are interested in calculating the probability that at least k subtasks are finished before time t. For this purpose, we will first calculate the probability that the main task will not be finished before t, also known as the probability of failure after t.

Theorem 4.1. *Let τ_{main} denote the finishing time of the main task. The probability that τ_{main} is larger than t, is calculated as:*

$$Pr(\tau_{main} > t) = \sum_{m_1=1}^{N} \sum_{m_2=0}^{m_1} \cdots \sum_{m_{k-1}=0}^{m_{k-2}} \prod_{s=0}^{k-1} \binom{m_s}{m_{s+1}} \big(F_s(t) - F_{s+1}(t)\big)^{m_s - m_{s+1}}, \quad (4.5)$$

where $m_i, i \in \{i\}_1^k$ denote the total number of finished subtasks in the i-th row of the workers, $\sum_{i=1}^{k-1} m_i < k$ while $0 \le m_{k-1} \le m_{k-2} \le \ldots \le m_1 \le N$, and $F_s(t)$ is the CDF obtained in (4.4) with L being the main load divided by k. Also by convention, $m_0 = N$, $m_k = 0$, and $F_0(t) = 1$.

Proof. We know that there are k rows of subtasks in the workers based on the order that they are executed and completed. Let us denote with j the number of all the subtasks that are completed before time t. Overall, if k subtasks are finished, the main task is completed. Hence we are interested in the probability of all the cases that $j < k$. The completed subtasks can be from any worker but at any worker, only those subtasks can be completed that the subtasks on their upper rows have finished earlier. Let m_1 be the total number of subtasks in the first row completed before time t. It is clear that $1 \le m_1 \le \min\{j, N\}$. The probability that $m_0 - m_1$ subtasks from the first row are not completed before time t is equal to:

$$\binom{m_0}{m_1} \big(F_0(t) - F_1(t)\big)^{m_0 - m_1},$$

where $m_0 = N$ and $F_0 = 1$. Let us show with $P_1(t)$ the probability that m_1 workers that have completed their first task will be able to complete at least $j - m_1$ subtasks before time t. Based on the multiplication rule, the probability of completing j subtasks in total would be:

$$\Pr(\tau_{\text{main}} > t) = \binom{m_0}{m_1} \big(F_0(t) - F_1(t)\big)^{m_0 - m_1} \times P_1(t). \quad (4.6)$$

We know that $P_1(t)$ is related to the subtasks of the second and further down rows of the m_1 workers that have already completed their first subtask. Let us assume that only m_2, $1 \le m_2 \le m_1$, out of m_1 workers also finish their second subtask. The probability that $m_1 - m_2$ out of m_1 workers are not able to finish their second subtask after finishing the first one is:

$$\binom{m_1}{m_2}\big(F_1(t) - F_2(t)\big)^{m_1-m_2}.$$

Hence, based on the multiplication rule, $P_1(t)$ can be written as:

$$P_1(t) = \binom{m_1}{m_2}\big(F_1(t) - F_2(t)\big)^{m_1-m_2} \times P_2(t), \tag{4.7}$$

where $P_2(t)$ is the probability that $j - (m_1 + m_1)$ subtasks from the third row or below are completed before time t by the m_2 workers that have already completed their second subtasks. Substituting (4.7) into (4.6), we obtain:

$$\begin{aligned}\Pr(\tau_{\text{main}} > t) = \\ &\binom{m_0}{m_1}\big(F_0(t) - F_1(t)\big)^{m_0-m_1}\times \\ &\qquad\binom{m_1}{m_2}\big(F_1(t) - F_2(t)\big)^{m_1-m_2} \times P_2(t)\end{aligned} \tag{4.8}$$

Let us assume that by receiving some subtask belonging to the r-th row, the total number of completed subtasks is j and the process ends. Following the same approach, we break $P_2(t)$ recursively until we arrive at the row r, $1 \leq r \leq k$, of the subtasks. For the r-th row we have $\sum_{i=1}^{r} m_i = j$ and $m_i = 0,\ i > r$. By continuing the recursion up to this row we will have:

$$\prod_{s=0}^{r}\binom{m_s}{m_{s+1}}\big(F_s(t) - F_{s+1}(t)\big)^{m_s-m_{s+1}} \times P_r(t), \tag{4.9}$$

where $P_r(t) = 1$. To simplify the formulation of (4.9), we can rewrite it assuming the recursion is continued until $r = k-1$. For this, we note that since $m_i = 0,\ i > r$ and by convention $\binom{x}{0} = 1$ and $x^0 = 1$, (4.9) can be rewritten as:

$$\prod_{s=0}^{k-1}\binom{m_s}{m_{s+1}}\big(F_s(t) - F_{s+1}(t)\big)^{m_s-m_{s+1}}. \tag{4.10}$$

Now in order to account for all the possible cases of $m_i, i \in \{i\}_1^{k-1}$ we put summations behind the formula in (4.10) and this completes the proof. $\square$

Corollary 4.1. *The probability that the main task finishes before time t is the complement of the probability calculated in Theorem 4.1. Hence, it is equal to:*

$$Pr(\tau_{main} \leq t) = \\ 1 - \sum_{m_1=1}^{N} \sum_{m_2=0}^{m_1} \cdots \sum_{m_{k-1}=0}^{m_{k-2}} \prod_{s=0}^{k-1} \binom{m_s}{m_{s+1}} \big(F_s(t) - F_{s+1}(t)\big)^{m_s - m_{s+1}}, \quad (4.11)$$

Remark 4.1. *In calculating the probability of finishing the main task before a deadline, we assume that the communication time between the workers and the master is negligible.*

4.4.4 Optimizing the Completion Time

As previously mentioned, in the HCDC scheme, k is not a fixed variable and can be chosen based on the requirements of distributed computing. If we take a closer look into the calculation process for the probability distribution of the finishing time of the main task, we can see that one of the important elements in its derivation is the finishing time distribution of a single subtask in (4.1) which in turn depends on L and μ. Here, L is itself affected by k as it is equal to the load of the main task divided by k, and μ is the average additional time delay for completing a subtask with load L. Hence, the choice of k depends on the dynamic between μ and L in (4.1) and how k can affect this dynamic for a desired distribution for the finishing time of the main task.

One way to obtain the k is to maximize the probability of finishing the main task before a specific deadline. Let us denote the deadline by t_d, then we can write:

$$k_d = \underset{k}{\operatorname{argmax}} \left\{ \Pr_k(\tau_{\text{main}} \leq t_d) \right\}, \quad (4.12)$$

where $\Pr_k(\tau_{\text{main}} \leq t_d)$ is calculated based on equation (4.11). In fact (4.12) is an integer optimization problem.

Another possible way is choosing k such that the average time of finishing the main task is minimized:

$$k_e = \underset{k}{\operatorname{argmin}} \left\{ \mathrm{E}\left[\tau_{\text{main}}\right] \right\} \quad (4.13)$$

where τ_{main} follows the probability obtained in (4.11). The decision to choose any of these methods for determining k depends on the needs of the user that employs the distributed computing system. Additionally, it is also noteworthy to mention that these two methods may not necessarily yield different k.

4.4.5 Subtasks per Worker

In the system model described in Section 4.2 we set the number of rows in each worker to be equal to k. As a result, the master needs to create a total of $k \times N$ coded subtasks. This will increase the encoding time compared to the single-task method where this value is only N. However, the experimental results show that when the workers execute the subtasks of a few first rows, enough number of subtasks are generated, enabling the master to start decoding and finalizing the main task. In other words, before the workers reach to their subtasks with low priorities, the process is ended by the master. Hence, some of the coded subtasks are almost never executed.

Despite this observation, the random nature of the straggling behaviour prevents one from being able to choose an exact lower number of subtasks that may be assigned to each worker for the successful completion of the main task within the same time as the case that k subtasks are assigned to each worker. For instance, if a large portion of the workers fail permanently, the remaining workers might need to run subtasks with very low priorities or even all their subtasks to produce enough results for the master to finalize the computation. Nonetheless, such a situation is very rare, and using our probabilistic approach, we can determine the row such that the probability of any worker reaching a subtask in that row is extremely low.

Let us assume that the proper k has already been determined. Let us assume that the proper k has already been determined. First we want to obtain a time that we are almost sure (in a probabilistic sense) that the main task is completed by. We will denote this time by t_θ:

$$t_\theta = \min\{t \,|\, \Pr(\tau_{main} \leq t) \geq 1 - \epsilon_1\},$$

where ϵ_1 is an extremely small value and $\Pr(\tau_{main} \leq t)$ is calculated based on

(4.11). Let us denote the event that a worker will finish at least s subtasks before t_θ by $E_{s,\theta}$ and the event that the rest of the workers, i.e. $N-1$ other workers, do not complete at least $k-s$ subtasks before t_θ by $E'_{k-s,\theta}$. We want to calculate the probability that a worker will finish at least s subtasks by t_θ while the rest of the workers are not able to complete at least $k-s$ subtasks altogether at the same time. If we denote this event by E_s we can write:

$$\Pr(E_s) = \Pr(E_{s,\theta}) \times \Pr(E'_{k-s,\theta}). \tag{4.14}$$

In (4.14), $\Pr(E_{s,\theta})$ is calculated based on (4.4) and $\Pr(E'_{k-s,\theta})$ is calculated from (4.5) but for $N-1$ workers. Hence, $\Pr(E_s)$ can be written as:

$$\Pr(E_s) = \frac{\gamma\big(s, \frac{t_\theta - sL}{\mu}\big)}{\Gamma(s)} \times \\ \sum_{m_1=1}^{k-s} \sum_{m_2=0}^{m_1} \cdots \sum_{m_{k-1}=0}^{m_{k-2}} \prod_{s=0}^{k-1} \binom{m_s}{m_{s+1}} \big(F_s(t_\theta) - F_{s+1}(t_\theta)\big)^{m_s - m_{s+1}}. \tag{4.15}$$

$\Pr(E_s)$ is a good estimate for the probability of the event in which a worker finishes s subtasks before the master ends the process, i.e. before the rest of the workers complete $k-s$ subtasks. As a result, to reduce the encoding complexity, we can search for s such that it reduces $\Pr(E_s)$ to a desirably-low value and use this s as the proper number of rows (instead of k) in the HCDC scheme without compromising the performance:

$$s_\theta = \min\{s \,|\, \Pr(E_s) \leq \epsilon_2\},$$

where ϵ_2 is a very small value.

4.5 Experimental Results

In this section, we will simulate the time to finish a general task with the load L_{main} and show the advantages of the HCDC scheme over the conventional single-task schemes as well as conventional schemes based on the MMC model that fix the load of the subtasks in advance.

4.5.1 Finding the Optimal k in the HCDC Scheme

We assume that we have a task $g(x)$ that has the load $L_{main} = 600$s, i.e. it will take 600 seconds for each of the workers to complete it on their own. We consider a scenario where the number of workers available to the master is $N = 10$. As we previously mentioned, the average time for the completion of a subtask in addition to its normal computation time denoted by μ in equation (4.1) depends on the the characteristics of the workers and the distributed system. Additionally, for different μ values, the optimal number of information subtasks k may change. As a result, we vary μ from $1s$ to $160s$ and find the optimal value of k for each μ. The optimal k is the number of information subtasks as well as the number of subtasks that are assigned to each worker in the HCDC scheme.

Here, we find the optimal value of k with both of the methods explained in Subsection 4.4.4. First, we find the optimal k by maximizing the probability of finishing the main task before a deadline. The deadline that we consider here is obtained as $t_d = \frac{L_{main}}{N} + \mu$. Please note that this is an arbitrary choice and other values for the deadline could be considered as well. Then we find k that maximizes the probability of finishing before t_d based on Equation (4.12). In Figure 4.3, the probability of finishing before the specified deadline is plotted for different μ with k varying from 1 to 30. We can see that all the curves follow the same trend. For small and large values of k the probability is low but for middle k values it grows until it reaches to the peak value and starts dropping. The k corresponding to the peak value is the chosen k in the HCDC scheme. We also observe that the optimal value of k for different μ varies.

In the second method, we find the optimal k that minimizes the expected completion time of the main task according to (4.13). In Figure 4.4, the expected completion time is depicted. Similar to the first method, The horizontal axis shows different k from 1 to 30 and each curve corresponds to a specific μ. The curves in this figure start with a high expected finishing time, later drop to a minimum value for middle k values, and finally rise to higher values for larger values of k again. The optimal k which depends on μ, is the number of

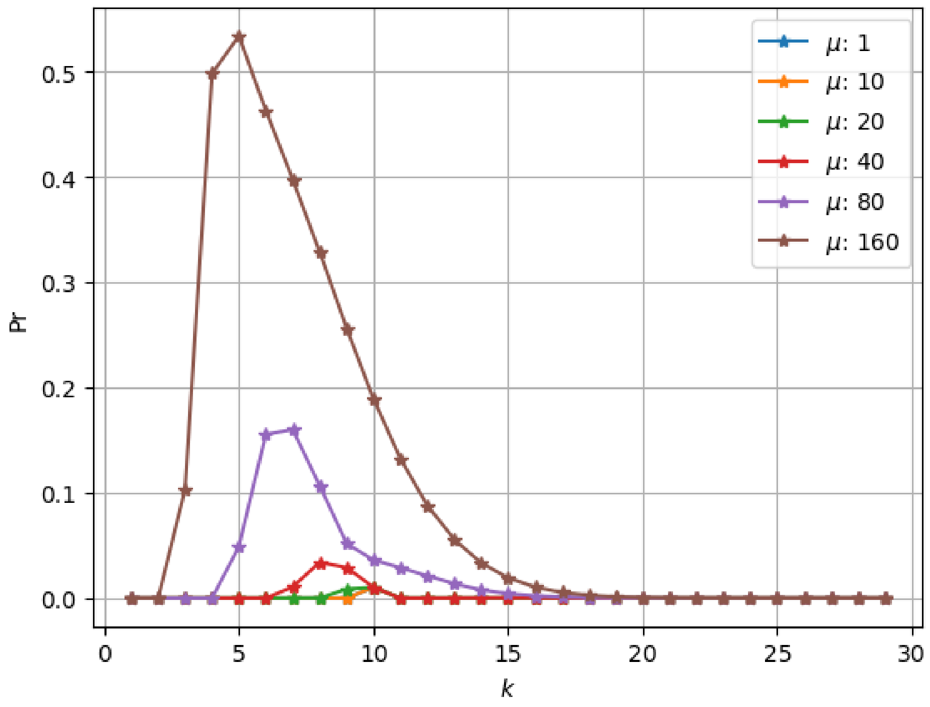


Figure 4.3: Probability of finishing the main task before the deadline t_d for different k. Here each curve corresponds to a specific μ.

information subtasks in the HCDC scheme.

In Figure 4.5, the values of k corresponding to the peak values of the probability curves in Figure 4.3 and the values of k corresponding to the minimum values of the expected finishing time curves in Figure 4.4 are plotted together. Depending on the objective of the optimization, the curves show the number of information subtasks in the HCDC scheme for different μ values.

As we see in Figure 4.5, both of the curves show that when μ is close to zero, i.e. the average delay of computing a subtask in addition to the normal computation time is negligible, the optimal number of information subtasks is equal to the number of workers. This is expected since $\mu \approx 0$ means there is no straggling behavior. Hence, the best solution is to divide the main task into a number of subtasks equal to the number of workers, N, and give each worker one information subtask as its highest priority subtask. As μ increases, i.e. the straggling behavior is more severe, the number of information subtasks in the HCDC scheme drops. In other words, due to a larger delay in completing

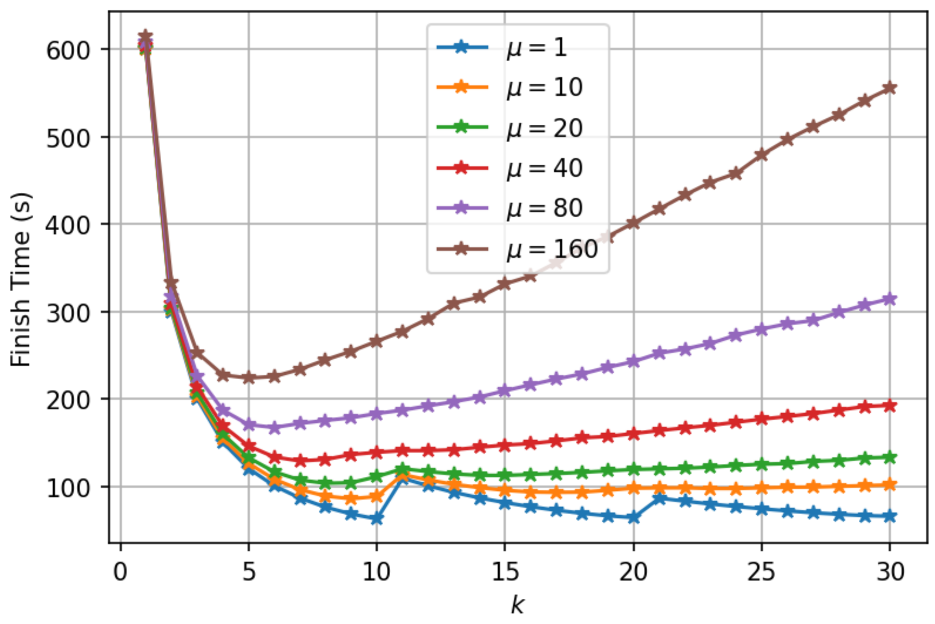


Figure 4.4: Expected finishing time of the main task for different k. Each curve corresponds to a specific μ.

each subtask, waiting for the completion of the subtasks with lower priority in the workers becomes costly. Thus, the system prefers to reduce the number of the subtasks it needs to finalize the main task, i.e. k, even if this comes at the cost of a bigger computation load per subtask. Please note that although the points of the two curves do not necessarily overlap, both of the curves follow the same trend and the corresponding k values are very close to each other.

4.5.2 HCDC Performance versus Single-task Scheme

We assume that we have the same setting as the previous subsection, i.e. a main task with $L_{main} = 600$s, $N = 10$ workers, and μ varying from 1s to 160s to compare the performance of single-task and HCDC schemes over different conditions. Let us first consider the single-task scheme. As described in Section 4.2, in the single-task scheme the code length is equal to the number of workers: $n = N$. Regarding the choice of k, we want our system to be tolerant in the face of three permanent stragglers, i.e. the minimum distance of the code at least should be $d = 4$. Since in MDS codes $d = n - k + 1$, we have $k = 7$.

On the other hand, in the HCDC scheme, each worker has $p = k$ subtasks.

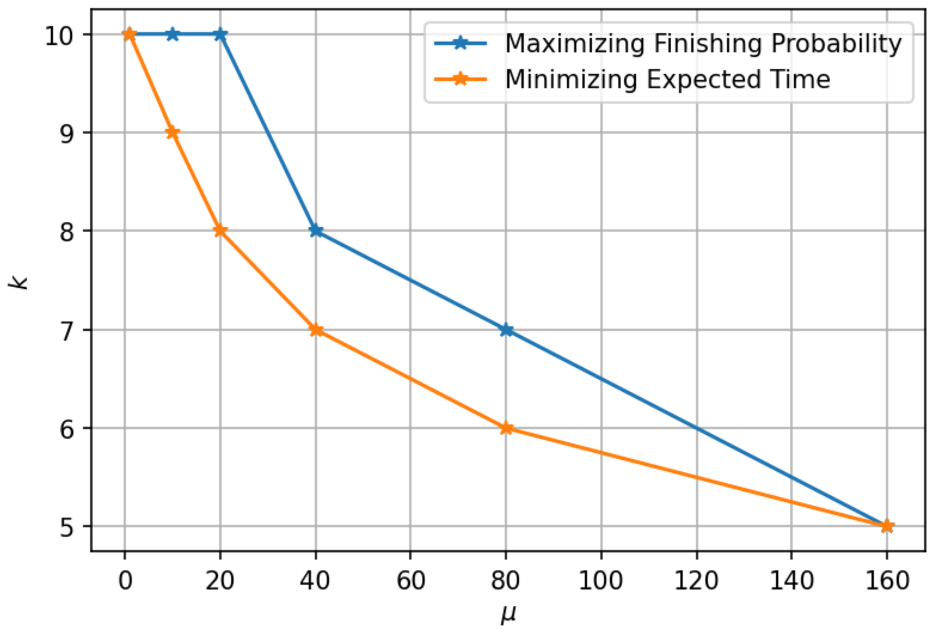


Figure 4.5: k in the HCDC scheme for different μ. The blue curve shows the values of k obtained by maximizing the probability of finishing the main task before t_d and the orange curve shows the values of k that are obtained by minimizing the expected completion time of the main task

Hence, the length of the MDS code is $n = k \times N$. Here for HCDC the value of k, is obtained by minimizing the average finishing time of the main task as stated in (4.13). Since $p = k$, even if all but one of the workers become permanent stragglers, the system can still complete the main task. In Figure 4.6, the average finishing time of the single-task and HCDC schemes are plotted. As depicted, HCDC has a lower average finishing time over the entire range of μ.

Another advantage of the HCDC over the single-task scheme becomes apparent when the distributed system is prone to permanent stragglers also referred to as permanent failure of the workers. To compare the performance of the two schemes in the presence of permanent stragglers, we consider a scenario where a main task with $L_{main} = 600$s is distributed among $N = 10$ workers and $\mu = 40$. Please note that this corresponds to the point in Figure 4.6 that the single-task and HCDC schemes have the same average finishing time in the absence of permanent stragglers. Figure 4.7 shows the rate of the successful completion of the main task in the single-task and HCDC schemes. In this

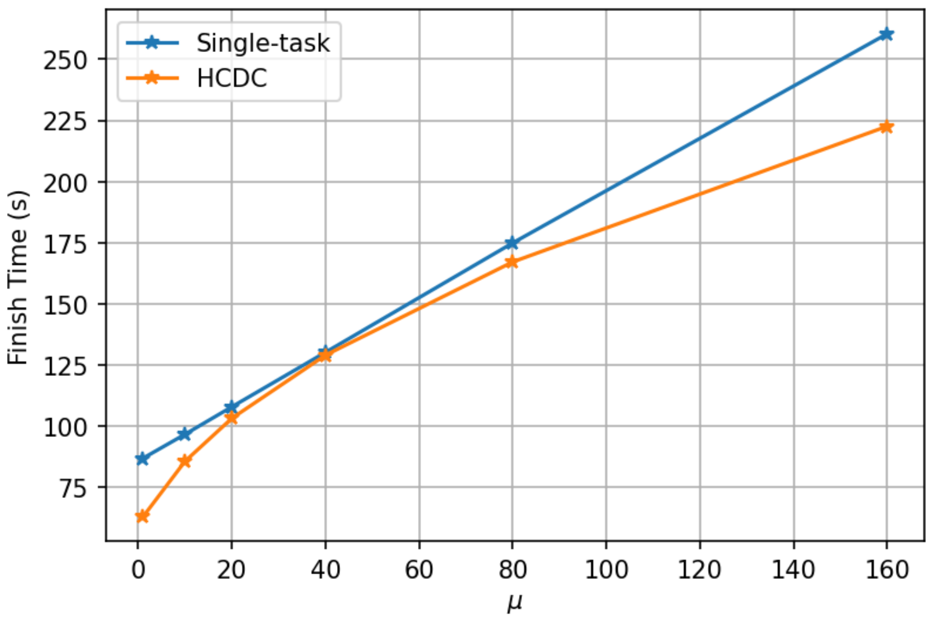


Figure 4.6: Comparison of the average finishing time of the main task between single-task and HCDC schemes

figure, the horizontal axis is the probability of being a permanent straggler in each of the workers. We can see that as the probability of failure in each worker increases from zero, the success rate in the single-task scheme drops while it stays at 1 over the entire range of the probability of failure in the HCDC scheme.

4.5.3 HCDC versus Conventional MMC Schemes

Let us consider a setting similar to the previous section, i.e. $L_{main} = 600$s and $N = 10$. As we discussed, in the conventional MMC scheme the number of information subtasks and the number of total subtasks given to each worker are fixed values without taking into account the characteristics of the workers such as the average delay in addition to the normal computation time. Here, we obtain the expected finishing time of the conventional MMC scheme in two cases and compare it with HCDC. In both cases, the total number of subtasks given to a worker is equal to $N = 10$. In one case the number of information subtasks given to each worker is one and in the other case, it is two, i.e. $k = N$ and $k = 2N$ respectively. In Figure 4.8 the expected finishing times

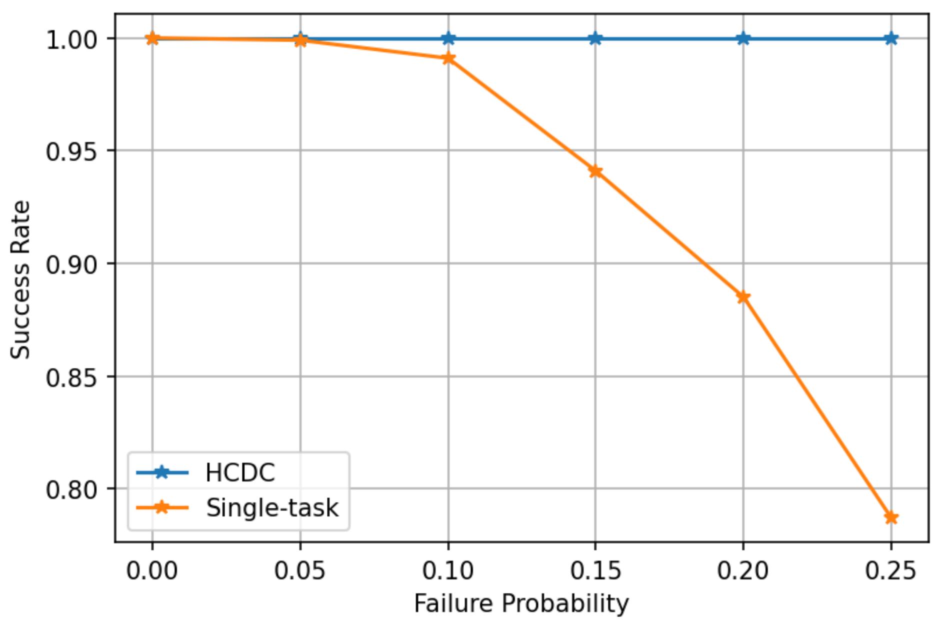


Figure 4.7: Rate of the successful completion of the main task in single-task and HCDC schemes over different probability of permanent failure in the workers

are plotted. We can see that the HCDC scheme has a better performance compared to the conventional MMC schemes over the entire range of μ.

4.5.4 HCDC versus Hierarchical Coded Distributed Computing

One of the recently proposed schemes for the MMC model is the hierarchical coded computing (HCC)[20]. Unlike HCDC, and similar to conventional MMC, the number of total information subtasks is a fixed number in HCC. A number of the information subtasks are assigned to each row of the subtasks. In each row, a separate MDS code is employed and some redundant subtasks are generated to be assigned to the workers that do not have information subtasks. Since the subtasks in higher rows are more probable to be finished, the number of the information subtasks assigned in the upper rows is higher than the lower rows. Following this general rule, the number of information subtasks assigned in each row is determined by either optimizing the probability of finishing the main task before a deadline or the expected finishing time of

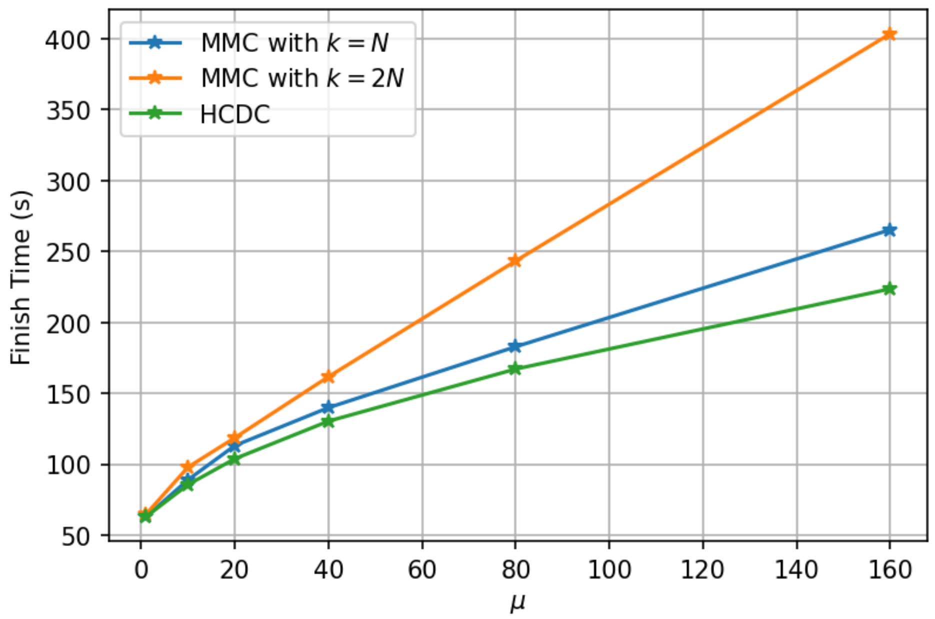


Figure 4.8: Comparison of the expected finishing time between the HCDC and the conventional MMC schemes with $k = N$ and $k = 2N$

the main task.

To compare the performance of HCDC with HCC, we assume a setting with $N = 8$ workers and a main task with the load $L_{main} = 600$s. We choose the N a smaller number here because the computational complexity for the optimization in the HCC scheme grows exponentially. For the HCC scheme, the number of subtasks assigned to each worker is $p = 4$, meaning that overall, there are $pN = 32$ subtasks. The total number of information subtasks is $k = 16$ and the number of information subtasks in each row is found by optimizing for the expected finishing time of the main task. The parameters of the HCDC scheme are also obtained by optimizing for the expected finishing time. In Figure 4.9 the performances of the two schemes are depicted.

4.6 Conclusions

In this work, we introduced a novel coded scheme, hybrid coded distributed computing (HCDC), to mitigate the straggler problem in distributed computing. HCDC is designed for the MMC model of distributed computing where

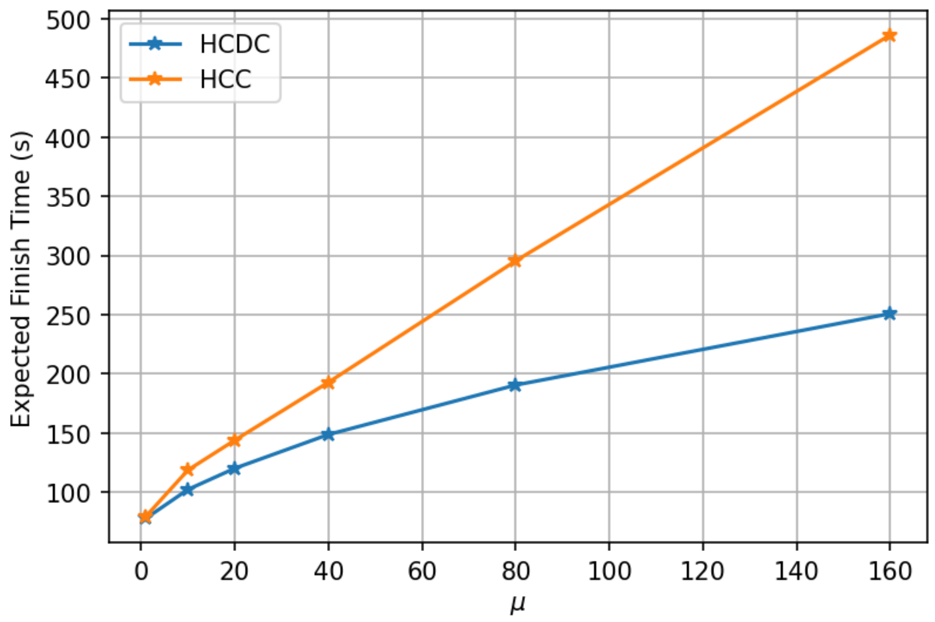


Figure 4.9: Comparison of the expected finishing time between the HCDC and the HCC schemes

each worker receives multiple subtasks and sends multiple messages to the master. In HCDC, a systematic MDS code is used for encoding the subtasks and the total number of the subtask assigned to each worker is equal to the information length of the MDS code, k. Unlike conventional MMC schemes, we do not fix k which in turn, determines the computation load of the subtasks. Instead, k in the HCDC scheme is one that optimizes the finishing time or the probability of completing the main task before a deadline.

More specifically, we built a mathematical model based on the main computation load and characteristics of the workers such as average delay in addition to the minimum computation time for completing a single subtask in each worker. Then we used this model to derive the probability of finishing the main task and find the k that optimizes it. We show that HCDC outperforms single-task schemes as well as conventional MMC schemes in terms of the average completion time of the main task. It is also maximally tolerant in the face of persistent stragglers, guaranteeing the completion of the main task even if all but one of the workers permanently fail.

Chapter 5

Conclusions and Future Directions

5.1 Conclusion

In this thesis, we started by reviewing the rapid rise in the demand for distributed computing and the reasons behind this growth in Chapter 1. In the same chapter, we discussed some of the challenges in distributed computing such as straggler bottleneck, communication bottleneck, and privacy. Then we reviewed the use of coding techniques to overcome some of these issues.

In Chapter 2, we focused on the straggler problem and the use of coding techniques to mitigate this issue in distributed computing. First, we laid out the general system model for master-worker distributed computing. Then we studied some of the fundamental concepts and mathematical basis in coding theory. Finally, we studied the use of coding in distributed computing for straggler mitigation. We introduced a general coded system model and reviewed some of the prior work on this topic.

One of the missing points in the literature related to the use of coding in distributed computing is attention to the fact that the encoding and decoding complexity may not be always negligible. In fact, the time spent for the encoding and the decoding may even become the main bottlenecks, especially, when these two steps cannot be performed offline, such as in large-scale machine learning applications. To this end, In Chapter 3 we considered large-scale distributed computation of matrix multiplication which is widely needed in many applications. We introduced a new scheme based on binary locally repairable codes (BLRCs). This was in contrast to the existing literature which predominantly focuses on MDS codes.

MDS codes have a high encoding and decoding complexity while our code in Chapter 3 had a very low encoding and decoding complexity. We proved that our code had the minimum encoding complexity among a large class of locally repairable codes and then provided an algorithm that decoded our code with the minimum complexity. Finally, we showed that the same matrix multiplication problem computed based on our scheme has a significantly lower overall completion time (encoding, computation, and decoding time) compared to the widely adopted MDS scheme.

In Chapter 4, with a focus on computation time, we considered a multi-message communication model (MMC) for distributed computing where each worker receives more than one subtask and is able to send multiple messages to the master. Our focus in this setup was on the computation time. We introduced a new scheme, hybrid coded distributed computing (HCDC), for the MMC model that reduced the computation time for completing a task compared to single-task schemes and conventional MMC schemes.

Unlike conventional schemes, in HCDC the load of the subtasks is not presupposed. Instead, it is chosen such that the distributed process has the optimal completion probability or finishing time. HCDC is maximally tolerant in the face of persistent stragglers, guaranteeing the successful completion of the task even if all the workers but one, permanently fail.

5.2 Future Directions

As we have previously mentioned, matrix multiplication is usually used as a general term to refer to matrix-matrix or matrix-vector multiplication. Both of these operations are in high demand in most machine learning and data analytic applications. As a result, in Chapter 3, we considered matrix-vector multiplication and introduced a novel scheme based on a family of BLRCs to reduce the encoding and decoding complexity. The high encoding and decoding complexity is also a serious problem in matrix-matrix multiplication. There is a potential to address the complexity issue in matrix-matrix multiplication with a novel scheme similar to our scheme in Chapter 3, i.e. based on the BLRCs. The challenge in such an attempt is minimizing the number of products between the submatrices that are required for finalizing the main task. Otherwise, the increase in the computation time might surpass the reduction in the encoding and the decoding time by using a BLRC.

As we discussed in Chapter 4, schemes based on the MMC model have some advantages over the conventional single-task schemes. In Chapter 4, similar to the majority of the work in the literature, we focused on the computation time and introduced the HCDC scheme. However, when the overall completion time of a task is important, the encoding and decoding times should also be accounted for in the proposed schemes. Hence, schemes based on MDS codes may not be the best choices. One interesting direction to follow is developing a scheme based on non-MDS codes for the MMC model, e.g. a scheme based on BLRCs. In the process of designing a scheme based on BLRCs for the MMC model, there are some challenges that need to be addressed.

One important aspect of the design is creating a systematic BLRC that has the lowest encoding and possibly decoding complexity. This will reduce the time spent in these two steps compared to MDS-based schemes. Furthermore, BLRCs are not MDS, i.e. the number of coded symbols required to recover the information symbols is not minimum. Hence, Another aspect that should be considered is choosing a BLRC as close as possible to the MDS codes in that regard. This helps to keep the computation time close to the MDS-based schemes, otherwise, the gain in the encoding and the decoding times might be overshadowed by the increase in the computation time.

One interesting fact about the systematic BLRCs is that the complexity of the encoding and decoding is not the same for all of the redundant coded symbols. In other words, some redundant symbols are constructed from a lower number of information symbols, which makes their encoding and decoding less complex. On the other hand, in the MMC model, a worker receives multiple subtasks with different priorities. Hence, there is an opportunity to optimize the allocation of subtasks and their priorities to reduce encoding or decoding complexity. For example, giving a higher priority to the less-complex subtasks over more complex ones will increase the probability of having enough less-complex subtasks to finalize the main computation, hence, helping reduce the decoding time when the master finalizes the process.

We hope that our work in this thesis motivates future researchers to follow these directions and that it provides a base for future contributions in this field.

References

[1] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones.," in *NSDI*, vol. 13, 2013, pp. 185–198.

[2] M. H. Ardakani, M. Mehrabi, M. Ardakani, and C. Tellambura, "On allocation of systematic blocks in coded distributed computing," *IEEE Communications Letters*, vol. 26, no. 4, pp. 748–752, 2022.

[3] B. C. Arnold, N. Balakrishnan, and H. N. Nagaraja, *A first course in order statistics*. SIAM, 2008.

[4] B. Bartan and M. Pilanci, "Polar coded distributed matrix multiplication," *arXiv*, vol. 2019, 2019.

[5] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 351–371.

[6] 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*, 2012, pp. 13–16.

[7] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[8] 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.

[9] N. Charalambides, H. Mahdavifar, and A. O. Hero III, "Numerically stable binary coded computations," *arXiv preprint arXiv:2109.10484*, 2021.

[10] B. Chen, W. Fang, S.-T. Xia, and F.-W. Fu, "Constructions of optimal (r, δ) locally repairable codes via constacyclic codes," *IEEE Transactions on Communications*, vol. 67, no. 8, pp. 5253–5263, 2019.

[11] V. Cristea, C. Dobre, C. Stratan, F. Pop, and A. Costan, *Large-Scale Distributed Computing and Applications: Models and Trends*. IGI Global, 2010.

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

[13] Y. Deng and M. Dong, "Heterogeneous coded distributed computing with nonuniform input file popularity," in *ICC 2022-IEEE International Conference on Communications*, IEEE, 2022, pp. 1936–1941.

[14] 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.

[15] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," *Advances In Neural Information Processing Systems*, vol. 29, 2016.

[16] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," in *2017 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2017, pp. 2403–2407.

[17] 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*, vol. 66, no. 1, pp. 278–301, 2019.

[18] J. Edmonds and M. Luby, "Erasure codes with a hierarchical bundle structure," *IEEE Transactions on Information Theory*, 2017.

[19] X. Fan, P. Soto, X. Zhong, D. Xi, Y. Wang, and J. Li, "Leveraging stragglers in coded computing with heterogeneous servers," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, IEEE, 2020, pp. 1–10.

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

[21] N. S. Ferdinand and S. C. Draper, "Anytime coding for distributed computation," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2016, pp. 954–960.

[22] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *IEEE Transactions on Information theory*, vol. 58, no. 11, pp. 6925–6934, 2012.

[23] S. Goparaju and R. Calderbank, "Binary cyclic codes that are locally repairable," in *2014 IEEE International Symposium on Information Theory*, IEEE, 2014, pp. 676–680.

[24] S. Gupta and V. Lalitha, "Locality-aware hybrid coded mapreduce for server-rack architecture," in *2017 IEEE Information Theory Workshop (ITW)*, IEEE, 2017, pp. 459–463.

[25] J. Hao, S.-T. Xia, and B. Chen, "Some results on optimal locally repairable codes," in *2016 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2016, pp. 440–444.

[26] C. Huang, H. Simitci, Y. Xu, *et al.*, "Erasure coding in windows azure storage," in *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012, pp. 15–26.

[27] P. Huang, E. Yaakobi, H. Uchikawa, and P. H. Siegel, "Cyclic linear binary locally repairable codes," in *2015 IEEE information theory workshop (ITW)*, IEEE, 2015, pp. 1–5.

[28] H. Ishii and R. Tempo, "The pagerank problem, multiagent consensus, and web aggregation: A systems and control viewpoint," *IEEE Control Systems Magazine*, vol. 34, no. 3, pp. 34–53, 2014.

[29] D. B. Kahanwal and D. T. Singh, "The distributed computing paradigms: P2p, grid, cluster, cloud, and jungle," *arXiv preprint arXiv:1311.3070*, 2013.

[30] C. Karakus, Y. Sun, and S. Diggavi, "Encoded distributed optimization," in *2017 IEEE international symposium on information theory (ISIT)*, IEEE, 2017, pp. 2890–2894.

[31] O. Khan, R. Burns, J. Plank, and C. Huang, "In search of I/O-Optimal recovery from disk failures," in *3rd Workshop on Hot Topics in Storage and File Systems (HotStorage 11)*, 2011.

[32] M. Kiamari, C. Wang, and A. S. Avestimehr, "On heterogeneous coded distributed computing," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*, IEEE, 2017, pp. 1–7.

[33] 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.

[34] D. Kim, H. Park, and J. K. Choi, "Optimal load allocation for coded distributed computation in heterogeneous clusters," *IEEE Transactions on Communications*, vol. 69, no. 1, pp. 44–58, 2020.

[35] Y.-S. Kim, C. Kim, and J.-S. No, "Overview of binary locally repairable codes for distributed storage systems," *Electronics*, vol. 8, no. 6, p. 596, 2019.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] S. Li, S. Supittayapornpong, M. A. Maddah-Ali, and S. Avestimehr, "Coded terasort," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, IEEE, 2017, pp. 389–398.

[42] S.-J. Lin, T. Y. Al-Naffouri, Y. S. Han, and W.-H. Chung, "Novel polynomial basis with fast Fourier transform and its application to Reed–Solomon erasure codes," *IEEE Transactions on Information Theory*, vol. 62, no. 11, pp. 6284–6299, 2016.

[43] E. Ozfatura, S. Ulukus, and D. Gündüz, "Coded distributed computing with partial recovery," *IEEE Transactions on Information Theory*, vol. 68, no. 3, pp. 1945–1959, 2021.

[44] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," *IEEE Transactions on Information Theory*, vol. 60, no. 10, pp. 5843–5855, 2014.

[45] S. Prakash, S. Dhakal, M. R. Akdeniz, *et al.*, "Coded computing for low-latency federated learning over wireless edge networks," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 233–250, 2020.

[46] K. Rashmi, P. Nakkiran, J. Wang, N. B. Shah, and K. Ramchandran, "Having your cake and eating it too: Jointly optimal erasure codes for I/O, storage, and network-bandwidth," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, 2015, pp. 81–94.

[47] 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.

[48] 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.

[49] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Transactions on Information Theory*, vol. 65, no. 7, pp. 4227–4242, 2019.

[50] 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.

[51] R. M. Roth, "Introduction to coding theory," *IET Communications*, vol. 47, no. 18-19, p. 4, 2006.

[52] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *Sn Computer Science*, vol. 2, 2021. [Online]. Available: `https://api.semanticscholar.org/CorpusID:232322114`.

[53] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, *et al.*, "Xoring elephants: Novel erasure codes for big data," *arXiv preprint arXiv:1301.3791*, 2013.

[54] R. Schlegel, S. Kumar, E. Rosnes, and A. G. i Amat, "Codedpaddedfl and codedsecagg: Straggler mitigation and secure aggregation in federated learning," *IEEE Transactions on Communications*, 2023.

[55] R. Schürer and W. Schmid, "Table for linear codes," *mint. sbg. ac. at/table. php? i= c*, 2014.

[56] 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.

[57] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" *IEEE Transactions on Communications*, vol. 64, no. 2, pp. 715–722, 2015.

[58] M. Shahabinejad, M. Khabbazian, and M. Ardakani, "An efficient binary locally repairable code for hadoop distributed file system," *IEEE Communications Letters*, vol. 18, no. 8, pp. 1287–1290, 2014.

[59] M. Shahabinejad, M. Khabbazian, and M. Ardakani, "A class of binary locally repairable codes," *IEEE Transactions on Communications*, vol. 64, no. 8, pp. 3182–3193, 2016.

[60] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[61] 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.

[62] N. Silberstein, A. S. Rawat, O. O. Koyluoglu, and S. Vishwanath, "Optimal locally repairable codes via rank-metric codes," in *2013 IEEE International Symposium on Information Theory*, IEEE, 2013, pp. 1819–1823.

[63] N. Silberstein and A. Zeh, "Optimal binary locally repairable codes via anticodes," in *2015 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2015, pp. 1247–1251.

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

[65] Y. Sun, J. Zhao, S. Zhou, and D. Gunduz, "Heterogeneous coded computation across heterogeneous workers," in *2019 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2019, pp. 1–6.

[66] I. Tamo, D. S. Papailiopoulos, and A. G. Dimakis, "Optimal locally repairable codes and connections to matroid theory," *IEEE Transactions on Information Theory*, vol. 62, no. 12, pp. 6661–6671, 2016.

[67] Q. H. Vu, M. Lupu, and B. C. Ooi, *Peer-to-peer computing: Principles and applications*. Springer, 2010.

[68] A. Wang and Z. Zhang, "An integer programming-based bound for locally repairable codes," *IEEE Transactions on Information Theory*, vol. 61, no. 10, pp. 5280–5294, 2015.

[69] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," in *The 2014 ACM international conference on Measurement and modeling of computer systems*, 2014, pp. 599–600.

[70] 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.

[71] J. Wang, K. Shen, X. Liu, and C. Yu, "Construction of binary locally repairable codes with optimal distance and code rate," *IEEE Communications Letters*, vol. 25, no. 7, pp. 2109–2113, 2021.

[72] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Timely-throughput optimal coded computing over cloud networks," in *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2019, pp. 301–310.

[73] A. Yazdanialahabadi and M. Ardakani, "A distributed low-complexity coding solution for large-scale distributed FFT," *IEEE Transactions on Communications*, vol. 68, no. 11, pp. 6617–6628, 2020.

[74] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *The 22nd International Conference on Artificial Intelligence and Statistics*, PMLR, 2019, pp. 1215–1225.

[75] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[76] 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.

[77] 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.