Coded computing systems decoded: dealing with unreliability and elasticity in modern computing

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering.

Yaoqing Yang
B.S., Electronic Engineering, Tsinghua University

Carnegie Mellon University
Pittsburgh, PA

May 2019

Dedicated to my parents, who teach me to be kind, tell me that life is about happiness when I am tired and give me support when I want to reach for the stars.

Acknowledgments

First of all, I want to thank my advisors, Pulkit Grover and Soummya Kar. If I can truly become a researcher in the future, I owe it all to you. You changed my research from deriving mathematical equations to understanding the meanings of the society and universe, from designing algorithms to solving puzzles, from writing papers to creating my artworks. Thank you for pointing me to profound questions that matter, for showing me how enthusiastic researchers can be to their research and career, and for always being there, standing by the whiteboard, to discuss the most exciting equations and algorithms.

I want to thank my committee members José Moura, Jelena Kovačević, Babak Hassibi and Matteo Interlandi for their profound insights and helpful comments which shaped my thesis. I admire you as real scholars. Matteo, thanks for your tremendous help on my implementation of the elastic computing algorithms during my internship at Microsoft.

I want to thank my research mentors, Mohammad Ali Maddah-Ali, Chen Feng, Dong Tian, Wei Chen, Pingyi Fan, Longbo Huang, and Bo Bai, who gave me valuable suggestions on both research and life. It is a pleasure for me to work under your guidance.

I want to thank my great collaborators, Viveck Cadambe, Siheng Chen, Yiru Shen, Haewon Jeong, Farzin Haddadpour, Malhar Chaudhari, Markus Weimer, Saeed Amizadeh, Chunguo Li and Keqin (Kevin) Liu. Studying from you is always the best way for me to grow as a researcher.

I am fortunate to have some wonderful friends during my graduate study. I would like to thank Yujun Yan, Anit Sahu, Siheng Chen, Sanghamitra Dutta, Praveen Venkatesh, Haewon Jeong, Alireza Chaman Zar, Chaitanya Goswami, Jiaming Cao, Pin-Yu Chen, Yong Zhuang, Jiyuan Zhang, Yingrui Zhang, Jian Du, Jonathan Mei, Yongjune Kim, Yuan Chen, Manzil Zaheer, Javad Mohammadi, Hadi Amini, Guannan He, Brian Swenson, Sérgio Pequito, Subhro Das, Mark Cheung, Jianyu Wang, Jin Sima, Wenbo Liu, Zhiding Yu, Liangyan Gui, Yuxiong Wang, Ankur Mallick, Mark Cheung, Jie You, Shanghang Zhang, Xia Miao, Xiao Zhang, Fan Tong, Zongge Liu, Zhonghao Luo, Rawad Bitar, Yunan (Charles) Wu, Hao Zhang, Zilin (Roy) Jiang, Youzhi (Tiffany) Bao, Hao Feng, Chaojing Duan, Swanand Kadhe, Vipul Gupta, Majid Mahzoon, Songze Li, Qian Yu, Satwik Kottur, Vishwanath Saragadam, Utsav Sheth, Duanshun Li, Jian (Skyler) Zheng, Bihan Wen, Wei-An Lin, Han Zhao, Michael Rudow, Yang Li, Boyue Li, Laixi Shi, Tian Tong, Tina Tong Liu, Lu Qu, Qing Ye, and Hao Gu. Special thanks to Yujun. Her support made my dull days exciting, and I look forward to continuing the discussions with her on life, universe and Einstein. Thanks to Anit and Jonathan, I always remember the good old days at Hunan Bar. Thanks Siheng for giving me advise on research. Thanks my awesome group members (from both Pulkit and Soummya's groups) for so many celebrations and parties.

I want to thank Osman Yağan, Tze Meng Low, Gauri Joshi, Christos Faloutsos, Bruno Sinopoli, Yuejie Chi, Rohit Negi, Vijayakumar (Kumar) Bhagavatula, for giving me guidance on the research and offering the best courses in CMU. You gave me the continued enthusiasm to address interesting scientific questions.

Thanks to all the CMU staffs who have helped me. I gratefully acknowledge support from NSF, the Bertucci Fellowship, and the SONIC center at UIUC. This thesis is supported by NSF ECCS-1343324, NSF CCF-1350314 (NSF CAREER) and NSF CNS-1702694, NSF ECCS-1306128, NSF CCF1513936, the Bertucci Graduate Fellowship, and by Systems on Nanoscale Information fabriCs (SONIC), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

Abstract

Robustness is a fundamental and timeless issue, and it remains vital to all aspects of computation systems, regardless of specific computation platforms, architectures, and algorithm design. The issue is also timely: modern computing systems are increasingly built on unreliable substrates. This thesis designs reliable computing techniques for distributed systems, circuits and networks. We primarily study techniques inspired from coding theory to address the robustness issues such as system elasticity, stragglers (slow workers), machine failures and soft errors, by carefully weaving redundancy into the data and the design of the algorithm.

We primarily focus on three aspects of coding-based computation techniques. The first aspect is to design adaptive computing techniques in large-scale systems that are *elastic*, i.e., the number of machines can change with time. The second is to make the coding-based computation schemes cater to the specific needs of iterative and multi-stage algorithms, such as power iterations and gradient descent that are essential for today's data analytics. The third aspect is to study the fundamental limits of information propagation in computation networks, provide information-theoretic outer bounds on error accumulation, and design in-network computing schemes to compensate for this error accumulation, e.g., in a circuit network where each computation component can be unreliable.

This thesis presents theoretical results that are fundamental to the understanding of computation systems, and opportunities for radical improvements, e.g. in computation load, communication overhead, and storage cost. We also present real-system implementations and real-data experiments and show our progress on taking these theoretical results closer to practice. The academic results presented in this thesis advance on classical results in the historical field of reliable distributed computing, while simultaneously addressing timely issues arising in today's computing systems.

Contents

List of Figures

List of Tables

Chapter 0

Maxwell's demon, von Neumann, and Moore's law: different fundamental perspectives on computing systems

Fundamental physics perspective

We start with a story of Maxwell's demon depicted in Figure 1. There is a demon who is in charge of two rooms of gas, which are room 1 and room 2. There are only two types of gas, type A and type B, and they are mixed evenly in these two rooms at the beginning. There is a light door between these two rooms, and the demon can open or close the door at any time it wants. Then, the demon decides to do the following thing: when a particle of type A is going to enter room 1 or a particle of type B is going to enter room 2, it opens the door. Otherwise, it closes the door. Clearly, if the demon is capable of completing this task, after a finite amount of time, room 1 is full of the gas of type A, while room 2 is full of the gas of type B. This is surprising because, according to the second law of thermodynamics, the total entropy of an isolated system can never decrease over time. However, the entropy of the demon's room eventually decreases to the minimum.

So from where does this new information come? The new information comes from the demon itself! By repeatedly opening the door to let the particle go through, the demon loses its information, and the entropy of the demon itself increases. Another interpretation is that in this story, the physicists assume that friction can be as small as possible. The next part of the story is that Landauer proved the fact that "any logically irreversible manipulation of information, such as the erasure of a bit or the merging of two computation paths, must be accompanied by a corresponding entropy increase in non-information-bearing degrees of freedom of the information processing apparatus or its environment" [146]. He showed in another paper that one could communicate with arbitrarily small energy consumption [147], again with the assumption that one can lower the friction and noise in communication to zero. We include this story because it connects to the mental map of the thesis, which is on the fundamentals of computation systems, how the theory of information can play a role here, and how computation and information interact between each other. Landauer's results come from a theoretical

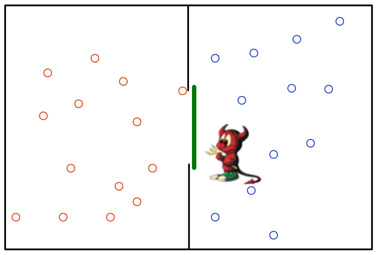


Figure 1: Maxwell's demon

physics perspective, which assumes away some critical points of the engineering problem and aims to understand the fundamental limit of any form of computation.

Von Neumann and Shannon' perspective

A more engineering perspective was taken by Von Neumann [253] in 1956. In this seminal work, he showed that one could construct a reliable computing circuit using entirely unreliable logic components. The main idea is to replicate each logic operation and wire and use a restoring mechanism to reduce error at each computation stage. This work is in spirit closely related to Shannon's work on communication over noisy channels [227] because von Neumann also established his theory based on comprehensive and rigorous modeling of *noisy* computing components.

One may think the work by Von Neumann is profound and universal because it applies to all kinds of computation that can be expressed as a logic circuit. However, if one considers today's unreliable systems, the techniques in this work are sometimes overkilling because the methods that are useful in typical computation systems nowadays are often lightweight and straightforward because the failures are usually assumed to happen in the level of individual processors or machines.

When I start formulating problems in the computation theory, I come to the belief that it is tough to find the *fundamental theory* for computation systems. The reason is that there are so many different theories and problems on computation, and all of them are limited in some sense, including the one by Landauer and von Neumann. However, a useful result of this tour in Von Neumann's world is that I come to the understanding that different perspectives can lead to *fundamentally different fundamental* theories, and it is fine to respect all of them and accept the fact that there is no *universal theorem* for computation problems.

A modern perspective

The same principle applies to many seemingly universal theories and techniques. For example, for over fifty years, people have been trying to follow Moore's law and increase the number of transistors on a circuit by making the transistors smaller. However, this trend is going end sometime inevitably, due to the limit imposed by power density issues and quantum effect. There may be other types of breakthrough technologies that can make the computation components even smaller and faster, but the main point is that the fundamental theory of transistors may not apply anymore to the new devices and platforms.

The diversity of computing systems has led to many *practical theories* that are successful in specific application domains. One example is Andrew Yao's famous theoretical framework on *communication complexity* [284]. This formulation is very intriguing and has spawned many works on distributed computing problems, such as the one on coding for computing by Orlitsky and Roche [189] which uses a definition of conditional graph entropy to characterize the information-theoretic limits of distributed function computation. However, the setting of these works is often only suitable for calculating the limit of communication, possibly with distributed dependent sources as in [189], but it is not easy to generalize these ideas to address robustness issues and some modern computation problems such as iterative and multi-stage problems. There are many other examples of the same nature. In the 80s, Thompson established the VLSI model of computation [246] by explicitly considering the mathematical models of circuit components and was able to show the implementation and time complexity of many modern computing problems such as the Discrete Fourier Transform and sorting. In [118], Huang and Abraham established the algorithm-based fault tolerance (ABFT) which is arguably the most successful and practical coding-based computing framework. We are now in the big trend of data-driven machine learning [214], and the framework of deep learning is also endorsed by the fast development of specialized hardware and distributed computing platforms which are not available when deep learning was invented. These computation theories and frameworks are tightly related to the development of effective platforms and are often suitable for designing domain-specific techniques.

When studying the problem of robust computing, the researchers often wish to establish a fundamental theoretical framework like the one by Shannon. At least from the current development of computation theory, I think the more appropriate way to say is that we have established many ways to address these fundamental questions from different perspectives. However, this is still very encouraging in some sense. Since there are so many different theories, models and techniques, we have to keep innovating our results to adapt to the changes in the problem itself. The inevitable saturation of Moore's law has made researchers look into the possibility of deliberately allowing noise in the computation to reduce energy consumption, leaving room for novel devices.

The perspective of this thesis

The fundamental physics perspective focuses on the universal limitations of general or even arbitrary computing systems by abstracting out the engineering features, such as noise and friction. On the contrary, our perspective explicitly incorporates technological limitations of specific computational systems. Our aim is to examine different levels of abstraction in a computing system, and enable failure and error-prone technologies to be used in modern systems while ensuring a target reliability. For example, these limitations can arise from the device technology used (e.g., CMOS 22nm [36, Extreme variations]) or straggling due to job scheduling and resource contention [271, Section 4.2.2], variation across workers [24, Section IV.C] or network-level issues [13, Section 4.2 Crossrack traffic], etc. Since we are focused on the end-engineering goal, we are interested in a real-system validation and taking the techniques from theory to practice. Therefore, our perspective unifies that of von Neumann and the more modern perspective, depending on at which level these unreliabilities arise, but also goes beyond these perspectives because we aim to address timely issues in novel systems and platforms. We also connect to and advance on Von Neumann and Shannon's classical perspective to develop new theories and techniques.

For example, Chapter 5 presents the first work that connects coding algorithms to elastic systems and provides implementation results on Microsoft's real systems. Some of them are algorithm level issues, such as Chapter 1 and Chapter 7 which present novel techniques that can increase the noise tolerance of coding-based reliable computing by exploiting *specific* properties of multi-stage and iterative learning algorithms. Some of them are intellectual issues that transcend platforms but can advance the understanding of practical systems. For example, results in Chapter 4 tighten the communication outer bound of distributed in-network matrix-vector multiplication by an arbitrarily large margin by connecting to another line of exciting works on information dissipation [79, 198]. Chapter 3 presents the first work to show that coding-based binary matrix multiplication techniques using noisy computing components can beat replication-based techniques in a realistic setting.

All models are wrong, and all theories are limited. Some excellent technologies may last for a long time, but we still have to understand their application domain. Therefore, the thesis touches different aspects of computation systems, from the gate level to the processor level, and computation networks. It also presents various aspects of computation problems, from computation failures and soft errors to stragglers and elastic events. We focus on the interaction between theory and practice and put effort into both deriving fundamental bounds and developing system implementations. In a nutshell, we focus on developing from the modern perspective of computing systems and on limited non-universal but useful theories and practice.

Chapter 1

Introduction

1.1 Connecting coding, computing and platforms

In the first chapter, we discuss the connections between the following three components.

- **Coding:** the information-theoretic algorithms and analytical tools of error-correcting codes and compression techniques;
- **Platform:** various robustness issues that arise in today's computing systems, circuits and networks, and typical optimization goals of these computing platforms;
- **Computing:** different types of computation problems, such as numerical linear algebra, machine learning algorithms, and data analysis problems.

We plot these three components and the connections between them in Figure 1.1 to show where our works are situated. In this figure, there are two blocks about *coding* techniques, which correspond to channel coding and source coding respectively. On the *computing* side, we categorize computing problems into two types, which correspond to matrix operations, such as matrix-matrix multiplications, and applications that have a multi-stage nature, such as gradient descent algorithms that are generally beyond basic matrix operations. On the platform side, we mainly consider three types of issues or constraints, namely system-level noise, gate-level noise, and system constraints imposed by the communication network. The texts on the solid lines between different blocks represent some existing techniques in the literature. The dashed lines connect each chapter in this thesis to different types of platforms and different categories of computing.

In this figure, we can see many existing theoretical frameworks and techniques on the connections between different blocks. These connections often lead to important problems that people care. We will provide a thorough literature review in Section 2.3 on these connections. Here, we choose to comment on some of these connections. We want to use these connections to present some critical coding-based problems where intellectual problems remain open, and how our works can address these issues.

On the connection between the block of *error correcting codes* and the gate-level noise, we have Von Neumann's seminal work on noisy computing [253]. This work is critical in that it tries to establish a grand theoretical framework for computing with noisy components that can fail in the level of logic gates. In my view, It is as important as

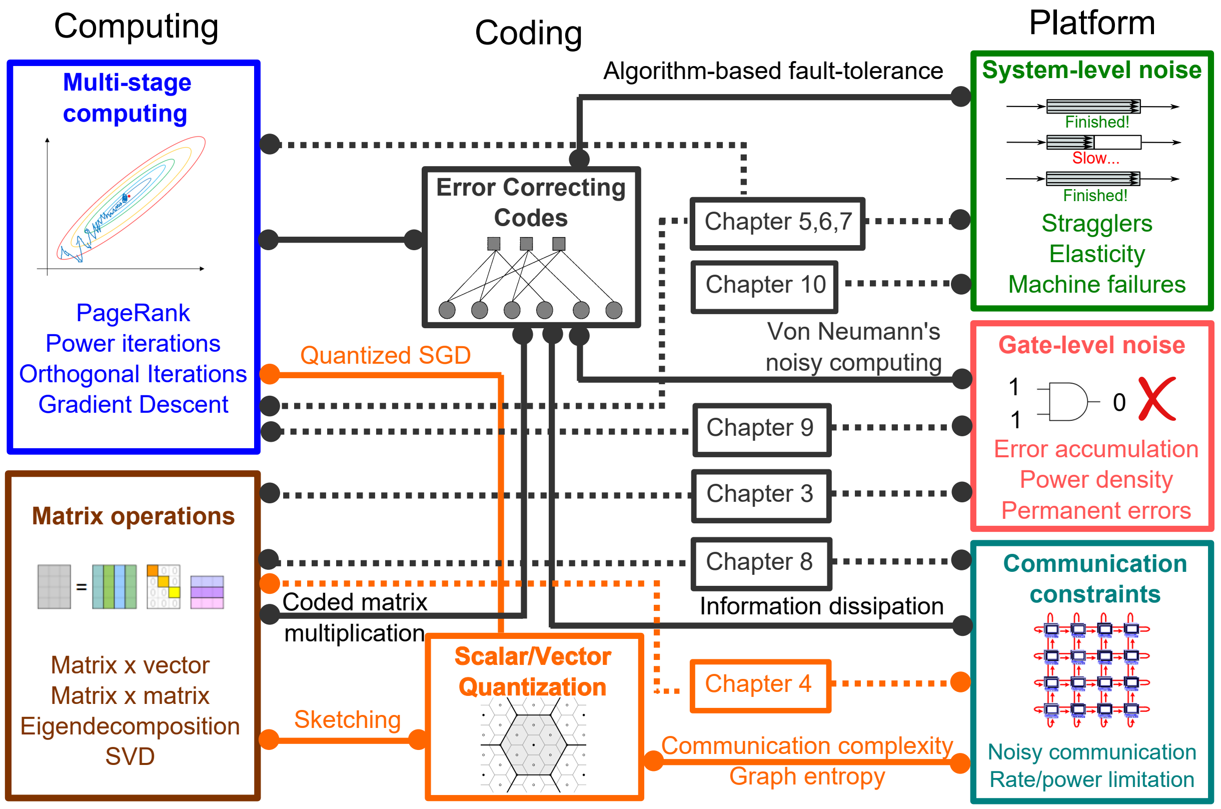


Figure 1.1: This map shows the interactions between coding, computing and platforms, and where our works are situated.

the work by Shannon [227] on communication using a noisy channel. Although this work uses robust techniques that are essentially replication-based, it has inspired a lot of coding-based techniques in the following fifty years [70, 109, 194, 244] about computing systems. In Chapter 3, we explore this direction of computing with noisy components. Our work is the first one to show that binary matrix-vector multiplications computed using entirely unreliable components can beat replication-based techniques in a practical parameter setting. Although it requires a substantial effort in proving the theoretical existence of some error-correcting codes, we observe a scaling-sense reduction in the final computation error using a moderately long code (of length 4000).

Another example is the connection between the block *scalar/vector quantization* and the block *communication constraints*. There are many information-theoretic works on the communication complexity [142, 284] of distributed computing, such as [60, 189, 240]. In Chapter 4, we present a new analytical tool which characterizes the communication complexity of linear function computing (i.e., matrix-vector multiplication) in a multi-stage network. The new theoretical lower bound obtained by this analytical tool is infinitely tighter than classical bounds based on cut-set techniques [60, 240].

The next point that we want to mention here is the *elasticity* entry in the block *system-level noise*. In Chapter 5, we present the first work in the literature that connects coding theory to the problem of elastic computing, and we show that one can design new coding-techniques by utilizing the properties of system elasticity. We also provide the first system implementations of coding-based elastic computing techniques and have applied them to the software at Microsoft (see Chapter 5 for details).

Last but not least, on the connection between *Scalar/Vector Quantization* and *matrix operations*, we have the sketching technique [166, 267]. Currently, researchers are trying to connect sketching-based techniques with coding [100]. Therefore, there will be more connections in Figure 1.1 in the future.

In this section, our first goal is to identify some critical connections between coding and computing problems and platforms, on which both important intellectual issues and application-oriented problems remain. Starting from Section 1.2, we look at the computation platform side of Figure 1.1. We will explain the details of each entry that has shown on the platform side in Figure 1.1 because these entries represent the primary motivation for us to study coding-based computing techniques. Here, we briefly summarize what we care about from the three angles of computing platforms.

Major issues that plague today's computing systems (more details in Section 1.2)

- **System/processor-level noise:** Noise in this level occurs in the form of processor or machine failures. Some processors can be slow or fault-prone. We will especially focus on the elasticity issue where machines can join or leave during computing. We will provide more details about system-level noise in Section 1.2.1.
- **Gate-level noise:** Noise in this level occurs in the form of gate failures, e.g., the output of a gate is always zero due to a short circuit. The problem of error accumulation is severe in this case because there may be thousands of gates for a simple function. We will provide more details about system-level noise in Section 1.2.1 as well.
- **Communication constraints in distributed/multi-processor scenarios:** Communication is probably one of the most critical limitations in large-scale distributed systems. We will show more details in Section 1.2.2.

In the following chapters, we will show how coding techniques can help address these different problems in various computation platforms. Generally speaking, coding techniques, especially those based on error correcting codes, are redundancy techniques. However, they often go beyond simple replication schemes that may require extremely high redundancy [163]. A natural question to ask is how coding-based techniques perform when compared with other schemes. In Table 1.1, we list some of these schemes. From this table, it is easy to see which problems are suitable for coding. In the table, the *ignore* strategy does not care about system noise and continue the computation regardless of any computation failure. The *stop-the-world* scheme means that the entire system waits for the failed or straggling machine to recover. This scheme can involve the check-pointing mechanisms [115, 197] which may require a significant amount of time for machine and state recovery. Dynamic task allocation means the relaunching of failed tasks especially when some tasks are straggling [9, 10, 12, 90, 242, 259, 260]. In

Section 2.3, we provide a thorough literature review on related techniques and show how coding-based techniques compare with them.

1.2 Motivation: what do we care about in today's computing platforms?

This thesis is centred around improving the robustness and efficiency of today's general computing systems. To understand the main bottlenecks of these systems, we try to identify significant issues and main performance bottlenecks plaguing computing systems, especially those that end users and practitioners pay attention to. Then, we try to abstract out theoretical models of these systems and propose algorithmic and mathematical solutions. Finally, we try to implement these solutions in real systems and new platforms to close the loop. The motivation of the thesis is categorized into the following aspects. The first two come from practical issues, while the last one is for advancing the theoretical knowledge and understanding of computing systems.

- Robustness issues in modern computing systems and circuits;
- Major bottlenecks and optimization metrics of computing systems;
- Theoretical fundamental limits on information propagation in computing systems.

1.2.1 Major robustness issues in computing systems

Achieving robustness against system uncertainty issues has been one of the most critical goals since the beginning of computer designs. In general, designing computing systems that are robust to any possible failures is hard. As we will show later in this section (Section 1.2.1), there are too many different kinds of computation failures, uncertainty issues, soft errors, and it is impossible to address all of them using one universal technique. The focus of computation robustness also slightly changes over time. For example, during the time of the earliest papers that are related to this thesis [245, 253], the focus is on the "gate-level", which concerns mostly about computational units that can represent simple logic operations. A failure at the gate-level means that may be an adder or an XOR gate fails. Nowadays, the research on computation robustness tends to focus more on the level of nodes, processors, or machines. There can be tens or even thousands of computing nodes in a large distributed network, and some of the nodes can fail during the computation. In different computing systems, the primary uncertainty issues can be changed. For the same type of issues, it can change over time as well. For example, for the problem of *stragglers*, it used to be critical at the early age of cloud service, but the issue has gradually become less significant. However, for new platforms such as serverless computing [123], the straggler issue remains.

Our goal, in this section, is to give a thorough overview of some computer system uncertainty issues that have been attacked using the ideas of error-correcting codes. As

Table 1.1: This table shows the comparison between different robustness techniques in computing systems.

Schemes	Storage redundancy	Compute redundancy	Robust	Machine recovery/relaunching time cost	Elastic	Get same result
Replication	High	High	Yes	Zero	Sometimes (redundant copies of new machines sacrifice elasticity)	Yes
Ignore	Zero	Zero	No	Zero	No	No
Stop-the-world	Low (for checkpointing)	Zero	Sometimes (can only tolerate rare failures)	High	No	Yes
Coding	Low	Low (including time for decoding)	Yes	Zero	Yes (see Chapter 5)	Yes
Dynamic task allocation	Zero	Low (for recomputing)	Sometimes (hard for frequent failures)	High (computation relauching)	No	Yes

we have mentioned, there is no way that we can solve all of these issues using a single technique. However, we hope to make a convincing argument that the coding-based techniques have applicability in many system issues, and the generalization of these techniques to different situations is also predictable and has rules to follow. From a theoretical point of view, this ability to generalize makes it possible to have a bread and deep conceptual framework. From a practical point of view, it enables a simple comparison between different techniques and backward compatibility with new system issues and modern techniques.

Computation elasticity in cloud service

Elasticity has been a focal point of system design from the early stage of cloud computing [85]. There are two ways to interpret elasticity. The first one is to view it as the ability for a distributed system to scale to an arbitrary number of computing nodes in a flexible and fast manner. The second way is to view it as the ability to continue the distributed computation tasks even if the number of nodes can drastically change. The first one is more from a system design perspective, while the second one is more from an algorithm design perspective. In this thesis, we focus on the second algorithm design perspective. For this perspective, there two possible problems to consider:

- **Passive-elasticity:** Cloud-service providers allow the exploitation of under-utilized Virtual Machines (VMs) at a fraction of the original cost [2, 4]. For example, Azure Batch provides low-priority machines at about one-fifth of the cost of ordinary virtual machines [5]. Similarly, Amazon Spot Instances provide machines at market price with a discount which can reach up to 90% concerning regular on-demand prices [3]. Such offerings, however, have the drawback that machines can be taken away at any time if a high-priority job appears. This drawback, in turn, will surface as a computation failure at the application level. During a computation task, if some machines are taken away, the intermediate computation states and results, including the local data stored at these machines, are also taken away [180]. Therefore, the algorithm should be designed in such a way that it has certain redundancy to compensate for the loss of some machines. The event that some machines are taken away is often called *preemption* or being *preempted*.
- **Positive-elasticity:** The fact that the number of machines can change also means that some new machines can join distributed computing. For example, this can happen when the machines have to be reallocated to achieve fairness [116] between users in a multi-tenancy cluster or meet the specific needs of some users at runtime. While ignoring these new machines can certainly make the computation continue, it is more useful if we can immediately employ these new machines in the computation, and maybe shift some computation load to these new machines to achieve the purpose of load balancing.

The elasticity event can happen on a large scale. For example, in the case of using a greedy bidding strategy to purchase machines from a spot-instance market, the number of computers can drop by a factor of more than 50% during the computation [165, Figure

3.c]. For some of the machine learning applications, it is fine to use partial data, especially if the precision requirement is not high, or if the number of data points is far beyond the model capacity. However, in many cases, it is still preferable to have all the data present during the entire process of model training. From the perspective of customers, it is merely more trustworthy if the whole dataset is available during training. Now that 50% of the machines can be preempted during the computation process, either the local data has to be shifted continuously between machines, or some redundancy has to be introduced into the data to compensate for the information loss. Ignoring these preempted machines may not be a good idea when preemptions can happen frequently and on a large scale. There are many different ways to address this system issue. We refer the readers to Chapter 5 for further details on the possible techniques.

Stragglers

State-of-the-art large-scale computing systems can feature thousands of machines, petabytes of memory size, hundreds of petabytes of persistent storage, and provide highly parallel and distributed platforms [26, 162] for executing complex computational tasks. A vital feature of these systems and the underlying applications is heterogeneity: these systems are built to perform diverse computational tasks with different data, memory, processing capabilities, and simultaneously cater to a broad class of end-users (commercial customers, institutions, individuals, etc.) with varying task types, resource requirements and priorities. Even for embarrassingly parallel distributed tasks of the same amount of computation load, the processing time can also differ at different machines. Therefore, the performance of these systems can often be bottlenecked by a small number of slow machines, known as the *stragglers* [63, 155]. The computation time of each machine is inherently variable, even if the computational load remains fixed. Thus, the overall computation time is going to be determined by the slowest machine among all.

This straggling effect has been widely observed in experiments. For instance, it was reported that stragglers are the most significant limiting factor when implementing distributed coordinate descent in the internal Google cloud [211]. It has also been observed in the literature that for the Amazon EC2 service, the slowest machines can take more than five multiples of the typical computation time to finish the same job [243]. Another example is that for the AWS Lambda service, which has a cloud-based massively scalable system, about 5 percent of the workers can take much longer time than the other workers [101]. For distributed computing using mobile devices, the straggler problem can become even more significant, e.g., in the federated setting [234]. The problem caused by stragglers can only become more severe in the modern *exascale* era [26, 162] where the number of machines increases due to the big data boom.

Machine failures, memory faults and soft errors

There are typically thousands of hard drive failures and about a thousand individual machine failures each year in an exascale computing system, such as a new cluster in Google [62]. The failures can also happen in graphics processing units for general-

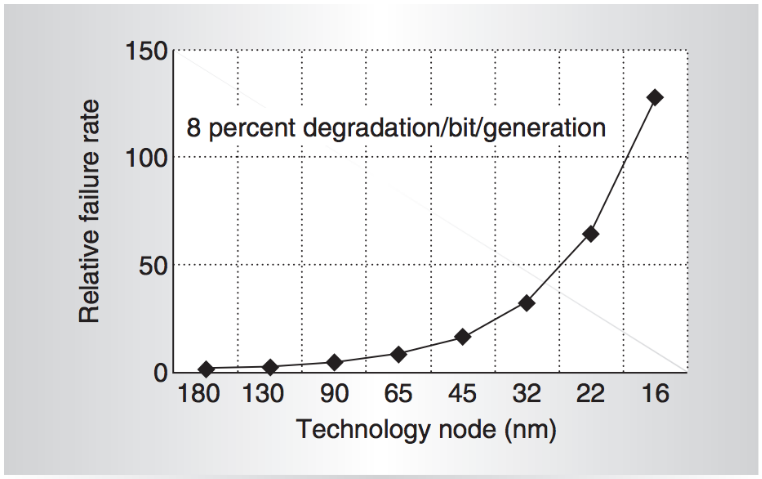


Figure 5. Soft-error failure-in-time of a chip (logic and memory).

Figure 1.2: This figure is from [36, Figure 5]. It shows that the soft error rate by the 16nm generation is more than 100 times that of the 180nm generation.

purpose computations (GPGPUs), which are commonly used in deep-learning systems and simulations of molecular-dynamics. For example, it was shown in an experiment on 50000 commodity GPUs used for the Folding@home distributed computing network that "two-thirds of tested GPUs exhibit a detectable, pattern-sensitive rate of memory soft errors" [112].

On the circuit level, the noise and variation issues are common as well. An urgent motivation to study noise in circuits comes from the saturation of "Moore's law" or the "Dennard's scaling", a scaling law on the energy with smaller technology [65]. In fact, greedily reducing CMOS transistor size can lead to severe robustness issues (see Figure 1.2). As transistors become smaller and clock frequencies become higher, the noise margin of semiconductor devices is reduced [297]. Meanwhile, timing jitter, voltage variation, crosstalk, thermal noise caused by increased power density and quantum effects can all jeopardize reliability. Many new devices are being explored to continue reducing energy consumption, e.g. [181]. However, such emerging low-energy technologies generally lack the reliability of CMOS as well. On the other hand, aggressive design principles, such as "voltage-scaling" (which is commonly used in modern circuits), reduce energy consumption [193], but often at a reliability cost. Beyond CMOS, circuits for many emerging technologies, such as those built out of carbon-nanotubes [229], suffer from reliability problems, such as wire misalignment [191]. While most modern implementations use overwhelmingly reliable transistors, there are also aggressive circuit design principles to deliberately allow errors in computation to achieve lower energy consumption [226]. Thus, circuit reliability is becoming an increasingly important issue and requires efforts from different perspectives such as circuit modeling, simulations and theoretical analyses.

1.2.2 System level optimization beyond fault-tolerance

When dealing with modern large-scale systems, e.g., a cloud center, a high-performance computing cluster, or other exascale computation platforms, we have to face the problems of arranging thousands of machines, connecting these machines in a massive network with different (and possibly varying) topologies, scheduling a large amount of distributed tasks on these machines in a efficient manner, and monitor the robustness of the entire system at a regular basis. The sheer size of the system design and optimization offers tremendous flexibility and numerous opportunities. Although the name of error-correcting codes makes it sound like the focus of the coding-based techniques are only issues of robustness, these techniques have been widely employed to improve the performance of distributed systems from various perspectives beyond fault tolerance or system uncertainty. In this subsection, we will briefly introduce three system-level optimization opportunities that we will talk about in the thesis, namely communication time, computation time, and energy consumption. These system performance metrics can also be related to robustness issues. For example, in machine learning training, if some data are missing from the entire dataset due to robustness issues, the training might take longer time to converge, and hence increasing the number of communication and computation rounds. Another example is that stragglers, although viewed as robustness issues, can directly affect the overall communication and computation time.

Reducing communication time

The network communication time in large-scale systems is arguably the most important metric to consider in the design of distributed systems and algorithms. The reason for this argument is threefold: (1) the time to communicate one bit of information across a distributed network can be larger than that of reading one bit of information from fast memory [62] (2) the increasing of the speed of computing is more rapid than the increasing of network information transmitting rate (for example, by Nielsen's law of internet bandwidth, users' bandwidth grows by 50% per year [188], which is 10% less than Moore's Law for computer speed) (3) while the computation time of many distributed computing tasks (e.g., the embarrassingly parallel tasks such as gradient evaluation and Monte Carlo simulations) can be directly reduced by adding more computation nodes, the communication time might stay the same or most likely increase with a larger number of nodes. Although the communication problem can be mitigated in small-scale distributed computing or large-scale systems with proper implementation of communication primitives, the communication time is likely to maintain a bottleneck for modern computing systems.

The well-known fact is that the communication time of point-to-point communication can be modeled by $T = \alpha + \beta N$, where α is the time to establish connection, determined by the network latency, β is the time to transmit one bit of information on the established communication link, determined by the network bandwidth, and N is the number of bits to send. Usually, α is much larger than β. However, βN can still be the dominating factor is the number of bits N is large. Therefore, the communication time can be reduced mainly from two perspectives. The first one is to reduce the number of communication rounds, and the second one is to reduce the overall number of bits to be sent. The communication time is often critical for I/O bound iterative algorithms in machine learning. We will discuss how to use coding-based techniques to reduce communication time mainly in Chapter 7.

Reducing computation time

Although in the previous part, we mentioned that communication time is an important metric, the computation time is essential as well. This is simply because, in many distributed computing problems, the computation can be much more complicated than reading or sending bits. In fact, for memory-bound machine learning applications, even for the simplest ones with only matrix operations, the computation process often requires to process data that has orders of magnitude larger size than that of the transmitted messages. For example, in distributed logistic regression, the size of the signal to communicate is either equal to the size of the model, or the number of samples at each machine (when dual methods are used [224]). However, since the size of each sample may be much larger than one, the computation cost to process each sample is much larger than transmitting one bit. For more advanced machine learning models, such as deep neural networks, the computation time involves complicated data processing, such as convolution and nonlinear operations, which is not negligible. The phenomenon that the computational cost becomes dominant is observed in experiments, such as in Chapter 5 and Chapter 6.

Therefore, in the case of a small-scale or medium-scale distributed computing task, the computation time can be a significant part of the overall time. Sometimes, the robustness issues, such as straggling, can also significantly increase the amount of computation time, and can often cause more severe straggling during computation that that during communication. Note that most of the computation time, from a real system perspective, comes from the task of loading data from slow storage to fast storage in the memory hierarchy (which can either be from memory to cache, or from local disks to fast memory if the local data does not fit into the main memory), while the CPU processing time is often negligible. In the case that communication time and computation time have comparable significance, one needs to have a balance between these two. Since computation time is often an essential metric for system optimization, we will discuss ways to reduce the computation time in most of the problem formulations mentioned in this thesis.

Reducing energy consumption

Another advantage of saving computation and communication time is that it often directly relates to reducing energy consumption in distributed systems. It has been widely recognized that the energy consumption in data centers is becoming increasingly critical in recent years. For example, in [61], it was mentioned that "U.S. data centers use more than 90 billion kilowatt-hours of electricity a year, requiring roughly 34 giant

(500-megawatt) coal-powered plants. Global data centers used roughly 416 terawatts (4.16 x 1014 watts) (or about 3% of the total electricity) last year, nearly 40% more than the entire United Kingdom. And this consumption will double every four years."

Before the saturation of Moore's law, we can reduce energy consumption and increase the computation speed by reducing the size of transistors. However, due to the power density issue and the robustness issues mentioned in Section 1.2.1, people can no longer achieve these goals by making transistors smaller. Meanwhile, due to the fast growth of hyperscale and exascale data centers developed by the technical giants, and the exponentially growing size of the data and need to process those data, the power and cooling cost at the data centers worldwide are continuously increasing.

The energy consumption problems arise from not only the giant data centers and cloud computing centers, but also small and energy-hungry mobile devices. These mobile devices, although significantly limited in the battery power, are viewed by many as the emerging platform for the next-generation cloud services [7, 172]. Therefore, understanding the energy consumption issue, along with the robustness issues such as straggling and adversaries in these settings, is essential as well. In Chapter 3, we will show that by using the idea of dynamic voltage scaling, one can reduce the energy consumption of iterative computing by orders of magnitude.

1.2.3 Intellectual motivation: understanding information propagation in computing systems

The aim of this section is to review some of the historical developments of the information-theoretic understanding of the robustness of computing systems, and to provide the intellectual motivation for the thesis: understanding the fundamental limits of information propagation in computing systems, and uses that to provide guidance for problem formulation and the design of achievable schemes. The extended context of this section will appear in Chapter 3 and Chapter 8 (achievable schemes for computing in a noisy network) and Chapter 4 (a fundamental limit for linear computation in a tree network).

We want to start this section with some review of history. The field of robust computation using unreliable components started from Von Neumann in 1956 in the seminal work [253], where he designed the technique to replicate computation and communication wires to achieve reliable computation using unreliable components. More precisely, for a network that reliably computes a function using l reliable logic gates, Von Neumann replaces each reliable gate by $\mathcal{O}(\log l)$ unreliable gates and replaces each reliable wire by a bundle of $\mathcal{O}(\log l)$ wires. He then uses "restoring organs" to correct errors. Using this delicate construction, he shows that the function network comprising l reliable gates can be implemented using $\mathcal{O}(l \log l)$ gates. The work of Von Neumann inspired many works on robust computation using unreliable components, including the first coding-based technique by Taylor [245] and the works on *noisy computing* [79, 195]. It also closely relates to the works on information dissipation [198]. I think many people believe that the idea of applying coding techniques to computation systems starts from the work of algorithm-based fault tolerance (ABFT) [118]. However, it is surprising that this line of works has a much longer history. Our thesis would like to serve as the first one to connect the two worlds of researchers on coding-based techniques for computing systems. The results shown in Chapter 3, 8 and 4 also illustrate that our works are deeply inspired from these prior works.

Although the construction of Von Neumann provides a clear way to deal with noise in an arbitrarily large computation network, the replication factor $\log l$ is large. The problem comes from the fact that the noise caused by unreliable computation and communication may accumulate through the network. Therefore, to compensate for this information loss, replicate has to be implemented at each stage of the computation. Interestingly, on any single path of computational units, the computation error will *must* accumulate, as discussed in information dissipation work of Evans and Schulman [79, Lemma 2] (see also [198]). Thus, to reduce the error accumulation, we should maximally distribute the information in the network, so that information can be integrated from different paths to fight dissipation on each path. This idea leads to the natural choice of expander graphs, which are widely used in LDPC codes [233]. It is also quite reasonable to maintain the computational results in a coding graph, or more precisely, to store the intermediate results in a *coded* form, so that forward error-correcting codes can help decode the results in a fully distributed way. Note that all computational units can be fault-prone, even including the decoding itself.

Coding has always played a fundamental role in information propagation, e.g., in communication systems [227] and storage systems [66]. In this thesis, we aim to develop this understanding of coding theory and techniques from communication systems to the much broader field of computation systems. As we have mentioned above, the information dissipation or the accumulation of errors can be mitigated by combining the information from other information paths, and by storing intermediate results in a coded form. This idea has a better interpretation (which I believe was first explicitly proposed in [109]) that the decoding can wait until the number of errors is large enough. In other words, as long as the noisy version of intermediate results scattered in the information path is sufficiently close to the coded version of the actual result, one does not need to remove all the errors at each decoding stage entirely. We will show in Chapter 3 that by using ideas from coding, one can save computational cost by orders of magnitude than replication-based techniques, and by using voltage scaling, one can further save energy consumption by orders of magnitude.

Although the understanding of information propagation and other fundamental limits in computation systems has immense potential impacts on practice, the achievable schemes provided in the history of this field thus far have become mostly theoretical. For example, For the current being, the real-world large scale systems have fundamentally different structures from the ones assumed in the work of Von Neumann, and the later works on noisy computing. In fact, the focus of robustness in large-scale distributed systems nowadays are mostly on the level of machines, instead of gates, and a direct consequence of this change of attention is that the depth of the standard computation network is usually small, e.g., a hypercube where the diameter is logarithmic in the number of nodes. However, we should not merely discard these remarkable prior results because of the deviation from practice. To some extent, the results in noisy circuits, noisy computing and information propagation have a major influence on the research nowadays. In my own opinion, these first results and the constructive algorithms can often be *simplified* to derive the results that can cater to current systems (because a network of noisy gates considered in [253] is intuitively much more general and complicated than a network of machines). Nonetheless, it is still vital to build a bridge between these results to practice, especially to system implementations and new computation platforms, to address specific issues in modern systems (e.g., computation elasticity). The content of this part is explicitly shown on some connections in Figure 1.1.

1.3 The contributions of this thesis

The main goal of the study is to understand the fundamental problems in computation systems and develop theory and practice to improve the performance of real platforms. There are three primary purposes of the thesis.

- The first purpose is to address grand challenges, especially new problems in today's unreliable systems. We will see problems, e.g., elastic computing and multi-stage computing, that have not been focused on in the literature of robust computation but can have a major impact on real systems.
- The second purpose is to develop fundamental theory and achievable techniques to solve or partially solve these problems. Each of the chapters in the thesis is centered around one aspect or one problem, but these problems are all connected to robust computation and the optimization of computing systems.
- The third purpose is to present implementation results on real systems and novel platforms, to prepare for the application of the proposed techniques into real products.

The main focus of the thesis is to develop practical techniques of and fundamental understanding of *algorithmic robust computing*. One approach that we focus on is to carefully weave redundancy into the computation by tailoring error-correcting codes and melding them with the computation. In the following, we present the main contributions of the thesis.

1.3.1 Coded elastic computing: problems, system implementations, and new platform

In Section 1.2.1, we have explained the problem of computation elasticity. Our work [281] is the first one to apply coding techniques to address the robustness issue in elastic systems. The main contribution of the paper comes from designing an adaptive data partitioning scheme and applying the adaptive scheme into the coding techniques for the elastic computing problem. The other contribution is the implementation of the proposed technique under the Apache REEF [56] framework.

The main idea of the paper comes from the following two observations of the problem:

- (Utilizing new machines) As mentioned in Section 1.2.1, there are two major aspects of the problem of computation elasticity. Although the first one, passive-elasticity, or the machine preemption problem, is substantially similar to machine failures, the second problem of positively utilizing new machines is beyond the usual motivation of coded computation. Therefore, the first observation is that the coding technique should be able to adapt to the changing number of machines, especially the new machines.
- (Utilizing information on elasticity) The second observation is a subtle one: although the number of machines can change over time, at the time a change happens, the entire system can be notified about the change immediately, and thus the information can be used for the design of adaptive systems. In other words, although we do not know when an elastic event (some machines are preempted, or some new machines join) will happen, and how much does the number of nodes change, we do know that the number of machines remains fixed for a certain period of time in the future, and then this information can be used.

Based on the two observations mentioned above, our work [281] proposes an adaptive coding scheme that can automatically change the workload at each machine based on the number of existing machines and is observed to continue the exact computation in the presence of machine preemptions in the multi-tenancy cluster at Microsoft. The proposed technique can achieve up to about $2\times$ speedup when compared to the existing way of coded computation and can achieve almost the same time cost as the case when there is no machine failure. In theory, it can achieve infinite gain when the number of machines approaches infinity[1]. The proposed technique is an exact scheme, meaning that it provides the same computation result as if no machines are preempted. This property of having an accurate result is helpful in applications where all the data are required to be present during the entire computation. As we have mentioned in Section 1.2.1, this property also ensures the requirement from a costumer's perspective.

1.3.2 Coded computation for multi-stage problems

Multi-stage computing, or iterative computing, is one of the most common computation patterns in machine learning. Usually, coded computation requires an encoding stage to weave the redundancy into the computation. If the computation is one-shot, it is often not desirable to have an expensive encoding step that reduces the applicability of coded computation. However, for iterative computation, the same encoded data can be repeatedly utilized for multiple iterations, and the encoding cost can be amortized during the entire computation. Therefore, coding for iterative computing is more useful than one-shot problems in terms of reducing encoding cost. Moreover, iterative computing is the basic format of many gradient-descent type computations and power-iteration based

[1]As per the discussion with Professor Viveck Cadambe, achieving the benefit when the number of machines is large may require a better scaling of the communication scheme. In reality, when the number of machines approaches infinity, the serial communication time usually dominates, and the saving on the computation becomes insignificant.

computations. Thus, coded iterative computing has wide applications.

Although designing coded computing techniques for iterative computing problems is widely applicable in many problems, it is more than just applying coding techniques to hot problems. One reason is that these techniques can go beyond the simple combination of multiple rounds of the basic one-shot coded computation, and includes the specific properties of different iterative algorithms. Coded computing has been applied to iterative computing, e.g., in gradient descent [47, 110, 208, 243]. However, these techniques apply coding to each single iteration of the iterative computation and do not utilize the relationships between consecutive stages. This type of applying coded computing into a multi-stage problem is essentially the same as coded one-shot computing. What we are interested in are the techniques that can have the following properties:

- They can utilize the properties of iterative algorithms to create opportunities beyond one-shot coded computing. For example, they can provide reasonable ways to combine intermediate results from consecutive stages to speed up computation and increase the algorithmic robustness to system uncertainties.
- The techniques may directly combine and encode the computation in several consecutive iterations, instead of coding a single iteration.

The idea of coded multi-stage and iterative computation will appear in two chapters of the thesis, namely Chapter 6 and Chapter 7. We will mainly talk about two techniques that meet the above properties [272, 277]. The main observation in both of these two techniques is that compared to one-shot computations, the iterative computation has the specific property that can be used to boost the performance of coded computing. For example, in Chapter 7, we show that by carefully weaving the intermediate result from the previous iteration, we can improve the tolerable number of failures from $\mathcal{O}(1)$ to $\mathcal{O}(n)$. In another line of collaborative works lead by Haddadpour [104, 105], we observed that jointly coding several iterations of computation can reduce several iterations into one, and lead to scaling-sense savings on the communication time of power iterations. This result further shows that the design of coded computation in iterative computing has some flexibility, and can often result in creating techniques beyond coded single-shot computation. In the following, we briefly discuss the contributions of these two techniques of coded iterative computation.

Coded computation for linear inverse problems

In Chapter 6, we consider the problem of solving multiple linear inverse problems $\min_{\mathbf{x}_i} \|\mathbf{A}\mathbf{x}_i - \mathbf{y}_i\|$ using multiple machines. The number of machines is larger than the number of linear systems, but some of them may be slow. The same linear system matrix $\mathbf{A}$ is replicated at each machine, and different input vectors $\mathbf{y}_i$ are given to different machines. Since the input $\mathbf{y}_i$ and the output $\mathbf{x}_i$ of each linear system have the same linear relationship $\mathbf{x}_i = \mathbf{A}^\dagger \mathbf{y}_i$ (where † means the pseudo-inverse), the linear encoding on all the inputs $\mathbf{y}_i$'s naturally transfers to the linear encoding on the output vectors $\mathbf{x}_i$'s. In Chapter 6, we will see that this linear encoding property makes it possible to apply off-the-shelf coded computing techniques to this problem directly.

As we have mentioned, the critical observation in Chapter 6 is that the computation of the linear system at each machine usually proceeds with iterative algorithms, especially when the matrix $\mathbf{A}$ is hard to invert. Therefore, even for the slow workers, at which the computation of the linear system has not converged, the intermediate result is still useful. Therefore, directly discarding these results at the slow workers may not be the optimal way to combine results. Thus, we propose to combine these results using different weights that are proportional to the number of completed iterations. The larger the number of iterations has been completed, the closer the intermediate result is to the optimal value, and the larger the combined weight is. The proposed technique achieves a graceful degradation of the overall computation error concerning the number of stragglers, which differs from ordinary coded computing (e.g., coded matrix multiplications) that usually has a fundamental limit on the tolerable amount of stragglers. In other words, for ordinary coded computing, there is typically a *threshold* time before which the computation result is not obtainable. However, if the properties of the linear systems are utilized, approximate computation results can be obtained at any time.

Interestingly, another critical issue arises in the problem of coded distributed inverse solver, which turns out to be even more severe than in ordinary coded computing. The issue is that at the early stage of the inverse problem-solving process, since the computation result at each worker is an approximation of the exact result $\mathbf{x}_i = \mathbf{A}^{\dagger}\mathbf{y}_i$, the result at each machine has a quite large noise. Then, this noise can get amplified if the encoding matrix is not appropriately designed, e.g., some of its submatrices have a large condition number. It turns out that one can bypass the issue by combining all the results, instead of using just part of the results (e.g., the results from the non-stragglers). Various techniques, such as truncated SVD, can also solve the problem to some extent. However, the issue of a large condition number is something worth paying attention to for the design of coded computing techniques.

Coded iterative computation using substitute decoding

Iterative problems sometimes have certain advantages that coded computing techniques can exploit. In this section, we will briefly introduce one of these advantages and discuss a method that can use this advantage.

For linear coded computing, sometimes it is preferred that the code is sparse. Here, what we mean by saying a linear code is sparse is that each code symbol is a linear combination of a small constant number of data samples. The requirements to make the codes sparse come from the following aspects:

- Sometimes, the data itself is sparse. If we apply dense coding to sparse data, the coded data after computing dense linear combinations becomes dense as well. For example, suppose we have 20 parts of the data $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_{20}$, and each part is a sparse matrix of a fixed size. Suppose each coded symbol is a linear combination of the form $\mathbf{Y}_i = \sum_j \alpha_{ij}\mathbf{X}_j, i = 1, 2, \ldots, 25$ for some coefficients α_{ij}. Then, the problem is that the coded data can be at most 20 times as dense as the original data. This is not desirable from the perspective of reducing the storage cost.

- Sometimes, the code is applied in such a way that the communicated messages, instead of the data, are encoded. For example, in [110, 208, 243]. In this case, the data is essentially replicated, and the replication factor, which relates to the density of the code, directly determines the overhead in both storage and computation time. Although the main observation is that the communication time may dominate the overall time cost, and hence increasing the computation time may be acceptable, it is desirable that the overall storage cost does not increase by much. Therefore, a sparse code is better than a dense code in this case.

In Chapter 7, we will show a way to boost the performance of sparse codes, despite the general belief that sparse codes are *weak* for error correction. We will show that the number of correctable errors can increase from constant to half of the number of code bits in a codeword if one carefully combines the information from a previous iteration's intermediate result. We will use a more end-to-end metric to measure the improvement of the algorithm performance, which is the convergence rate of the iterative computation. To make the connection with coding clear, we will present a theorem showing that by utilizing the information from the previous iteration, the convergence rate of coded iterative computing is almost precisely the same as that of the noiseless computation. The difference is small until the number of erasure-type faults approaches $1 - R$, where R is the coding rate.

We call the proposed technique *substitute decoding* because the main technical idea is to substitute the unknown part of a codeword after decoding with the information from the previous iteration's intermediate result. The main intuition is that the intermediate results in two consecutive iterations are close to each other, and hence the substitution does not introduce too much inconsistency. In Chapter 7, we only present the results on a simple problem of computing power iterations and its applications in PageRank [190]. This substitution technique applies in general to the applications where the intermediate results converge gradually during each iteration, based on the intuition above.

Coded orthogonal iterations for spectrum analysis and singular value decomposition

Our work in Chapter 7 is the first to apply coded computing techniques to spectral analysis and singular value decomposition of large-scale matrix analysis. The power-iteration method is one of the most popular ways to compute the leading eigenvector of a square matrix. One can compute several leading eigenvectors together by repeatedly multiplying the candidate eigenvectors with the square matrix and followed by an orthogonalization step [215]. This *orthogonal-iteration* method is the prototype of many large-scale eigendecomposition methods, and can easily generalize to truncated singular-value decomposition. In Chapter 7, we will show that we can indeed align substitute decoding with the more general coded orthogonal-iteration method, but with a small caveat. Note that one of the significant changes is that in the orthogonal-iteration method, we have a subspace of possible eigenvectors, instead of a single leading eigenvector. Therefore, these eigenvectors may change their order during the computation. This becomes tricky if substitute decoding requires the combination of the results from two consecutive steps, because, if the order of the eigenvectors changes during these two steps, the combination is not meaningful anymore. Thus, we also provide a technique to identify the change in the order of these eigenvectors, thus enabling the correct combination of these two sets of eigenvectors.

1.3.3 Coded robust computation in the gate-level model

Understanding information propagation in computing systems is the main intellectual purpose of the thesis. To serve this purpose, we include a purely theoretical part of the thesis. The proposed techniques and fundamental limits derived in this part are for a general *gate-level* error model. In this model, the errors may happen at the gate or wire level, and one has to look at information propagation in the network structure closely. This part of the thesis consists of three components, listed in the following.

- **Understanding gate-level noisy matrix multiplication:** For the gate-level error model, the noise can accumulate after each logic operation, e.g., AND logic operations and XOR logic operations. Therefore, for a single dot-product that contains many such logic operations, the accumulation of the noise may make the final output a completely random bit. Therefore, we propose to partition the process of computing a matrix-vector multiplication into multiple stages and apply low-complexity decoding after each computation stage, so that the error accumulation is repeatedly suppressed. Our work [278] is the first to show that for binary matrix-vector multiplications, the number of operations of a coded method is strictly less than that of a replication-based method under a realistic parameter setting. At the same time, we also show that by changing the energy consumption at each stage, the overall energy, under mild assumptions on the energy-error model of noisy gates, can be reduced by orders of magnitude than conventional coded computing techniques. From this result, we can see that even for one-shot computing problems like matrix-vector multiplication, it may also be useful to treat it as a multi-stage problem and apply coding after each stage. This idea works in concert with the main message in Section 1.3.2, i.e., when applied to multi-stage problems, the coded computing techniques can have much higher flexibility than in the single-shot problems.
- **Understanding noisy computation in a network structure:** Scaling robust computation in a fully distributed setting requires the design of fault-tolerant techniques that are specifically tailored to the network structure. That is, the coded computation algorithm can only be designed in such a way that the network is capable of implementing it, and the code structure itself has to be realizable by the network structure. In [283], we show two robust distributed computation techniques in two specific sparse network topologies, namely arbitrary geometric networks and random Erdös-Rényi graphs, which can have scaling-sense lower communication cost than in an arbitrary network topology. This work generalizes the distributed coding technique in [86] for complete graphs to many different special sparse graphs while maintaining the achievable result in the communication cost. The main technical difficulty is to analyze the performance of the designed codes that

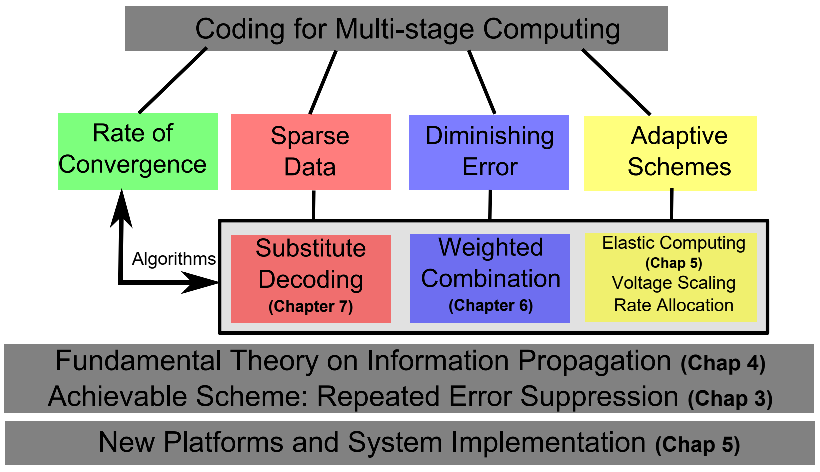

Coding for Multi-stage Computing
Rate of Convergence
Sparse Data
Diminishing Error
Adaptive Schemes
Algorithms
Substitute Decoding (Chapter 7)
Weighted Combination (Chapter 6)
Elastic Computing (Chap 5) Voltage Scaling Rate Allocation
Fundamental Theory on Information Propagation (Chap 4)
Achievable Scheme: Repeated Error Suppression (Chap 3)
New Platforms and System Implementation (Chap 5)

Figure 1.3: This figure shows the mental map of the thesis.

are constrained by the graph topology itself.

- **Understanding the fundamental limits of error accumulation:** We also study fundamental limits on the required communicated bits to compute a linear function to targeted accuracy in any arbitrary tree network [279]. The analysis uses a novel bounding technique that we call "distortion accumulation" [274] which states that for linear sequential function computing, the distortion of the intermediate results accumulates linearly as the number of computation stages increases, and the required communication bits can be lower-bounded by a function of the incremental distortion introduced at each stage. This technique leads to an outer bound that is infinitely tighter than an existing one on sequential function computation [60]. We propose to study the problem in a tree network because it is more general than a line graph which represents the flow of iterative computation. The ideal scenario would be to extend this result to arbitrary directed acyclic graphs which represent the structure of a general computer program.

1.4 Thesis outline: a mental map

In Chapter 2, we briefly review the background on several mathematical models of robustness issues and computation platforms. We also discuss some useful and popular techniques to address these issues. We will carefully provide a thorough literature review of the previous works on the understanding of computing systems and computation theory.

In Chapter 3, we introduce the gate-level error model and present an achievable scheme for binary matrix-vector multiplication. We will show the connections of this problem to historical developments on the research of information propagation in computing systems, and provide our understanding of the communication limitations in computation systems in Chapter 4. To connect with the graph structure of the computing system, we will revisit this line of works in Chapter 8 and show that the topology of the computation network plays a critical role in robust computation.

In Chapter 5, we introduce the elasticity problem in today's large scale systems and build a coded computation framework for this problem. We will mention a new computation platform on which our algorithms are implemented. The platform is called Apache REEF EGC (Elastic Group Communications), which is developed by practitioners at Microsoft. The new platform and the novel formulation of elastic computing bring out the necessity to understand multi-stage problems that are very unique in elastic systems, and we will address these problems in Chapter 6 and 7. In these two chapters, we mainly focus on utilizing the flexibility and critical properties of iterative computing and multi-stage problems to help design new coded computing techniques. These results show that when taking into consideration the multi-stage nature of the problem, we will be able to achieve scaling-sense improvements over coding-based methods for single-stage computation. We will also talk about how these techniques can be applied to numerical linear algebra algorithms such as spectrum analysis and singular value decomposition.

In Chapter 9 and Chapter 10, we show the case studies on two specific computation problems, namely distributed logistic regression and parallel convolutions using unreliable computation components. In Chapter 11, we will provide some discussions of the thesis, concluding remarks, and future directions. In the appendices, we mainly provide theoretical proofs. In Figure 1.3, we draw a *mental map* of the thesis.

Chapter 2

Technical background and preliminaries

2.1 Different computation systems and noise models

To make each chapter of the thesis self-contained, we will include a complete section of system models in each chapter. Here, in this background chapter, we provide a brief summary of different computation system models that we will consider in the thesis and some typical failure models.

Failures and robustness issues can happen in computation systems across various platforms and can have multiple forms. In large-scale commercial systems such as Amazon EC2, the failures usually occur at the node level, e.g., a machine failure, stragglers, and machine preemptions. These "system-level noises" or failure events can happen even more frequently if one wants to reduce the rental cost because preemptable and slow machines are often cheap. For circuits, many other gate-level or wire-level problems can happen, such as voltage variation, crosstalk, timing jitter, and thermal noise caused by increased power density. For energy-limited wireless sensor networks or in the edge-computing scenario, the system may cooperate to compute a specific function such as data summarization or average consensus. In this case, the communication between the nodes in the network is often limited in bandwidth and reliability because it is relatively hard to establish reliable and high-throughput communication links in a system with a large number of nodes. These type of constraints can either be in the form of noise imposed on the communication channels or imposed as rate constraints on the communication pipelines.

Therefore, we mainly study three types of failures. Note that we have mentioned these three models in Section 1.1. Here, we mainly focus on some details about mathematical assumptions.

- **Processor-level Failures:** Failures in this model happen at the processor-level of a distributed or parallel computing system. Failures may happen in the form of computation faults or delay at a node. For example, a straggler or a computation failure can be modeled as erasure in the computation result [149]. There may or may not be a master node. The network topology is usually simple, e.g., it can be a start network, a ring network, a binary tree network, or a high-dimensional mesh network, for which there are standard high-performance communication implementations. A common processor-level failure happens in a cluster formed by tens of or hundreds of machines. There can be soft errors as well, and the errors may not be detectable.

- **Gate-level Failures:** In this model, all logic gates and storage units are assumed unreliable. For example, these components may fail with a certain probability or may have a subset of components that always fail (caused by aging or stuck-at wire misalignment). This model is the hardest one among the three because every computation component is limited. Soft memory errors are typical in general-purpose commercial GPUs, and they can be modeled as undetectable memory noise. Soft errors in logic computation can also happen with low probability, and the probability increases when circuits get aged or some extreme design techniques are adopted to reduce energy consumption or increase computation speed. Therefore, it also necessary to establish a connection between the energy consumption on a gate and its output error probability, i.e., the gate error probability ϵ_g is some function $h(\cdot)$ evaluated at the amount of energy E_g.
- **Communication failures in distributed/multi-processor scenarios:** In this model, we consider an in-network computation setting and assume that communication links are limited: they may have noise or rate constraints. There can be a non-trivial network structure, such as a geometric graph or a general graph. A typical example of this model is a mobile or edge computation network. A computer cluster is also a good example of communication-bound tasks in which the overall communication overhead is the dominating cost compared to other costs like storage and computation. Compared to the processor-level failure model, the nodes in the network are often assumed reliable (noiseless). Two settings may be applicable here. The first one is to assume that each communication link has an independent communication noise or a bit-flipping event. The other one is to assume that there is a communication-rate constraint on each communication link because noiseless communication can be achieved on bit-pipes using channel coding when the communication rate is below the channel capacity.

In the thesis, we will present the results of our previous study on all of the above three models. In particular, we will study the following topics:

- the elastic events in the processor-level failure model, which can have a new node joining the computation (Chapter 5);
- the coded multi-stage and iterative problems, such as PageRank and gradient descent, in the processor-level failure model (Chapter 6 and Chapter 7);
- the undetectable soft errors in the processor-level failure model (Chapter 9 and 10);
- the soft errors in the gate-level failure model (Chapter 3), and the connection to energy-robustness tradeoff;
- the communication rate constraint in the communication-failure model, and its connection to error accumulation in computation systems (Chapter 4);

- the communication noise bottleneck for short-message transmissions in the communication failure model, when long-block codes are hard to apply (Chapter 8).

In each of these problems, we will see some or all of the issues mentioned above that arise. We hope to use this thesis as a chance to extend existing robust and system-aware computing techniques to different models and interesting applications and to explore deeper insights on some problems that have been addressed by current works.

2.2 Technical preliminaries and notation

Throughout the thesis, vectors are written in bold font, e.g., $\mathbf{x}$ and $\mathbf{y}$. Sets are written in calligraphic letters, such as $\mathcal{S}$. Scalar random variables are written in uppercase letters, e.g., U and V. A Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ is denoted by $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The all-zero vector with length N is denoted by $\mathbf{0}_N$, and the $N \times N$ identity matrix is denoted by $\mathbf{I}_N$. The calligraphic letter $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ represents a graph with a node (vertex) set $\mathcal{V}$ and an edge set $\mathcal{E}$. Each graph with N vertices has an N-by-N adjacency matrix $\mathbf{A} = (A_{m,n})$, which represents the edges or network connections, i.e., $A_{m,n} = 1$ if the node v_m has a directed edge to the node v_n.

We rely on the family of Bachmann-Landau notation [137] (i.e. "big-O" notation). For any two functions $f(x)$ and $g(x)$ defined on some subset of $\mathbb{R}$, asymptotically (as $x \to \infty$), $f(x) = \mathcal{O}(g(x))$ if $|f(x)| \leq c_2|g(x)|$; $f(x) = \Omega(g(x))$ if $|f(x)| \geq c_1|g(x)|$; and $f(x) = \Theta(g(x))$ if $c_3|g(x)| \leq |f(x)| \leq c_4|g(x)|$ for some positive real-valued constants c_1, c_2, c_3, c_4.

2.2.1 Preliminaries on error correcting codes

By $\mathbb{F}_2$, we denote the binary field $\{0, 1\}$. By $\mathbb{R}$, we denote the real field. We will use basic results from error control coding. From Section 2.2.4, we will consider codes in the real field. Here, we focus on *binary linear block codes*.

A binary linear block code with code length N and rate $R < 1$ is a set of 2^{NR} binary vectors (codewords) that form a linear subspace $\mathcal{C} \subset \mathbb{F}_2^N$. We always assume that NR is an integer. Each codeword $c \in \mathcal{C}$ can be written as the product of a binary row vector $\mathbf{m}$ with length NR, called the message vector, and an $NR \times N$ binary matrix $\mathbf{G}$, called the generator matrix. The coding matrix is usually written as

$$\mathbf{G} = \begin{bmatrix} \leftarrow & \mathbf{g}_1 & \rightarrow \\ \leftarrow & \mathbf{g}_2 & \rightarrow \\ \leftarrow & \dots & \rightarrow \\ \leftarrow & \mathbf{g}_K & \rightarrow \end{bmatrix}. \tag{2.1}$$

where each row $\mathbf{g}_k$ is a length-N codeword. If $\mathbf{G} = [\mathbf{I}, \mathbf{P}]$, where $\mathbf{I}$ denotes the $NR \times NR$ identity matrix, we say that the code with the generator matrix $\mathbf{G}$ is systematic. The matrix $\mathbf{P}$ is called the *parity* part of the generator matrix.

2.2.2 LDPC codes and the decoding algorithms

We will use different types of binary error correcting codes throughout the thesis. One special family of codes are the low-density parity-check (LDPC) codes [87]. The parity check matrix of an LDPC code can be represented by a bipartite graph, which is usually called the *Tanner graph*. In the Tanner graph, we denote the degree of a variable node v by d_v, and the degree of a check node c by d_c, i.e., a variable node v is connected to d_v parity check nodes in its neighborhood $\mathcal{N}_v$ and a parity check node c is connected to d_c variable nodes in its neighborhood $\mathcal{N}_c$. The followings are the formal definitions.

Definition: Bipartite graph($\mathcal{G}(N, M, d_v, d_c)$): This is a bipartite graph $\mathcal{G}(N, M)$ with N left nodes and M right nodes. In this thesis, we only consider regular bipartite graphs: each left node is connected to d_v right nodes and each right node is connected to d_c left nodes.

Definition: Expander($\mathcal{G}(N, M, d_v, d_c, \alpha, \delta)$): A bipartite graph $\mathcal{G}(N, M, d_v, d_c)$ is an expander graph with parameters $\alpha \in (0,1)$ and $\delta \in (0,1)$ if for any left node set $\mathcal{S}$ such that $|\mathcal{S}| \leq \alpha N$, the size of the neighborhood satisfies $|\mathcal{N}(\mathcal{S})| \geq \delta d_v |\mathcal{S}|$.

We will use the Gallager-B decoding algorithm defined in the following, which is a hard-decision iterative algorithm proposed in [87]. Suppose the received bits are $\mathbf{r} = (r_1, ...r_N)$. The Gallager-B algorithm works as follows:

- From variable node to check node:
  - Iteration 0: $m^{(0)}_{v\to c} = r_v$ is transmitted from v to every check node $c \in \mathcal{N}_v$.
  - Iteration i: $m^{(i)}_{v\to c}$ is transmitted from v to $c \in \mathcal{N}_v$,

$$m^{(i)}_{v\to c} = \begin{cases} x, & \text{if } |c' \in \mathcal{N}_v \setminus c : m^{(i-1)}_{c'\to v} = x| \geq b, \\ z, & \text{otherwise,} \end{cases} \tag{2.2}$$

    where $b = \left\lfloor \frac{d_v+1}{2} \right\rfloor$ and z is a randomly generated bit.
- From check node to variable node:
  - Iteration i: $m^{(i)}_{c\to v}$ is transmitted from check node c to variable node $v \in \mathcal{N}_c$,

$$m^{(i)}_{c\to v} = \bigoplus_{v' \in \mathcal{N}_c / v} m^{(i-1)}_{v'\to c}. \tag{2.3}$$

Remark 1. Note that the updating rule (2.2) involves the criterion to break ties. We use the updating rule $m^{(i)}_{v\to c} = z$ which is different from the original rule $m^{(i)}_{v\to c} = y_v$ in [87], in which y_v is the channel output associated with the variable node v.

Apart from the Gallager-B algorithm, We will use another simple parallel bit flipping algorithm [233] which is also a hard-decision algorithm. In particular, we use the modified parallel bit flipping algorithm defined in [39]. The PBF algorithm is often analyzed using the properties of expander graphs.

Definition: The PBF algorithm is defined as follows

- Flip each variable node that is connected to more than $\frac{d_v}{2}$ unsatisfied parity check nodes;

- Set the value of each variable node connected to exactly $d_v/2$ unsatisfied parity-check nodes to 0(or 1) with probability $1/2$;
- Update all parity check nodes;
- Repeat the first three steps for $c_e \log N$ times, where c_e is a constant.

The PBF algorithm can be used to correct a constant fraction of errors after $\Theta(\log N)$ decoding iterations when the computing components in the decoder are noiseless and the error fraction is small enough.

In Chapter 3, we may require the utilized LDPC code to satisfy some of the following conditions. The first bound comes from the need to study circuit implementations of noisy LDPC codes.

- **(A.1) Degree Bound**: The variable node degree d_v and the parity check node degree d_c are both less than or equal to some constant D, so that each majority or XOR-operation (which is used in the Gallager-B decoding algorithm [87]) can be carried out by a single unreliable gate. Moreover, we assume that the variable node degree $d_v \geq 4, \forall v$.
- **(A.2) Large Girth**: The girth $l_g = \Theta(\log N)$. An LDPC code with the following girth lower bound is obtained in [87, 152]:

$$l_g > \frac{2 \log N}{\log((d_v - 1)(d_c - 1))} - 2c_g, \tag{2.4}$$

  where $c_g = 1 - \frac{\log \frac{d_c d_v - d_c - d_v}{2d_c}}{\log((d_v-1)(d_c-1))}$ is a constant that does not depend on N.
- **(A.3) Worst-case Error Correcting**: One iteration of the PBF algorithm (see below) using a noiseless decoder can bring down the number of errors in the codeword from $\alpha_0 N$ to $(1-\theta)\alpha_0 N$ for two constants $\alpha_0, \theta \in (0, 1)$, for any possible patterns of $\alpha_0 N$ errors.

2.2.3 Random coding theory

First we state a lemma that we will use frequently.

Lemma 2.2.1 ([87], pp. 41, Lemma 4.1)**.** *Suppose $X_i, i = 1, \dots, L$, are independent Bernoulli random variables and* $\Pr(X_i = 1) = p_i, \forall i$. *Then*

$$\Pr(\sum_{i=1}^{L} X_i = 1) = \frac{1}{2}\left[1 - \prod_{i=1}^{L}(1 - 2p_i)\right], \tag{2.5}$$

where the summation is over $\mathbb{F}_2$, i.e., $1 + 1 = 0$.

A binary symmetric channel (BSC) with crossover probability ϵ is a channel that flips a bit with probability ϵ. A binary erasure channel (BEC) with erasure probability ϵ is a channel that outputs an erasure value 'e' with probability ϵ, no matter what value the input takes. We state two useful results from the theory of reliable communication [88]. The first one concerns repetition codes and the second one concerns linear block codes for reliable message transmission over noisy communication channels.

Lemma 2.2.2. *([88, Section 5.3]) Suppose we have a BSC with crossover probability ϵ. If one bit $x \in \mathbb{F}_2$ is repeatedly transmitted through the channel for j times and the receiver uses the majority rule to make a decision $\hat{x}$ the value of x, then, the bit error probability is upper bounded by*

$$P_e^{(bit)} = \Pr(\hat{x} \neq x) < [4\epsilon(1-\epsilon)]^{j/2}. \tag{2.6}$$

Remark 2. Lemma 2.2.2 states that $\mathcal{O}(\frac{\log 1/P_e}{\log 1/\epsilon})$ repeated transmissions are sufficient to achieve an error tolerance probability P_e at the destination, when the point-to-point source to destination channel is a BSC. One might also consider using adaptive schemes, such as sequential detection [257], to reduce the number of repetitions to achieve the same level of P_e. However, this does not change the number of transmissions in scaling sense.

Then, we provide an important result of binary random codes. Recall that we have a K-bit message vector $\mathbf{m}$ and a code $\mathcal{C}$ with length N and rate $R = \frac{K}{N}$. Then, we can encode the message $\mathbf{m}$ into N bits by multiplying $\mathbf{m}$ with the generator matrix $\mathbf{G}$, transmit these N bits over a channel and decode the received bits. The block error probability is defined as the probability that the decoding result $\hat{\mathbf{m}}$ is different from the original K-bit message at least in one bit. The next lemma characterizes the performance of using binary linear codes over a BSC.

Lemma 2.2.3. *([88, Theorem 5.6.2])(Random Coding Theorem) Suppose we have a K-bit message vector $\mathbf{m}$ to be transmitted through a BSC with crossover probability ϵ. Then, for each $R < C$, where C is the channel capacity, there exists a binary linear code with length N_R and rate R, such that $K < N_R R$ and the K-bit message can be encoded into N_R bits, transmitted through the BSC and decoded with block error probability upper bounded by*

$$P_e^{(blk)} = \Pr(\hat{\mathbf{m}} \neq \mathbf{m}) \leq \exp[-K E_r(\epsilon, R)/R], \tag{2.7}$$

where $E_r(\epsilon, R) > 0$ is the random coding exponent.

The random coding error exponent $E_r(\epsilon, R)$ for a BSC with crossover probability ϵ can be written as

$$E_r(\epsilon, R) = \max_{0 \leq \rho \leq 1} \left[-\rho R + E_0(\rho, \epsilon)\right],$$

where

$$E_0(\rho, \epsilon) = \rho \ln 2 - (1+\rho) \ln \left[\epsilon^{1/(1+\rho)} + (1-\epsilon)^{1/(1+\rho)}\right].$$

The random coding error exponent $E_r(\epsilon, R)$ is always positive for coding rate $R < C = 1 - H(\epsilon)$.

2.2.4 Real-number error correction

In this section, we present the genarlization of error-correcting codes to the real-domain, and provide results related to coded computing. We look at the problem of matrix-vector multiplication, and show that error-correcting codes can be used to protect the results from computation failures. Suppose we want to compute the matrix vector multiplication result $\mathbf{X}\mathbf{w}$. We partition the matrix $\mathbf{X}$ into K submatrices $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_K$ of equal size, i.e.,

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \\ \vdots \\ \mathbf{X}_K \end{bmatrix}. \tag{2.8}$$

We generate N *coded* data matrices $\mathbf{X}_i^{\text{coded}}, i = 1, 2, \ldots, N$, $(N > K)$, in which each matrix is a linear combination of the form

$$\mathbf{X}_i^{\text{coded}} = \sum_{j=1}^{K} g_{i,j}\mathbf{X}_j, \tag{2.9}$$

where each $g_{i,j}$ is a predetermined coefficient. Note that this real-number computing essentially treats each submatrix $\mathbf{X}_i$ as a symbol and codes the symbols using a (N, K)-code with the genertator matrix $\mathbf{G} = [g_{i,j}]$. This encoding process is similar with binary encoding.

The N coded data matrices are often distributed to N computation nodes. We may also use P to denote the number of computation nodes. In this way, the coded data $\mathbf{X}_i^{\text{coded}}, i = 1, 2, \ldots, N$ satisfy the following property with probability one for a variety of choices of the linear coefficients $g_{i,j}$'s, e.g., if $g_{i,j}$'s are i.i.d. Gaussian random variables:

Lemma 2.2.4. *Suppose we want to compute the matrix-vector product* $\mathbf{X}\mathbf{w}$. *Then, any K out of N coded computation results* $\mathbf{X}_i^{\text{coded}}\mathbf{w}, i = 1, 2, \ldots, N$ *are sufficient to recover the original (uncoded) computation results* $\mathbf{X}_i\mathbf{w}, i = 1, 2, \ldots, K$, *which are equivalent to recovering* $\mathbf{X}\mathbf{w}$.

This result appears in many coding-based linear computation techniques [109, 118, 149, 244]. The recovery of the results is through solving K linear systems of the form $\mathbf{X}_i^{\text{coded}}\mathbf{w} = \sum_{j=1}^{K} g_{i,j}\mathbf{X}_j\mathbf{w}$ for the K different computation nodes that successfully finish the computation. Lemma 2.2.4 essentially shows that no matter which computation nodes fail, as long as the number of remaining successful computation nodes is not smaller than K, the final result $\mathbf{X}\mathbf{w}$ can be obtained. We often call the parameter K the *recovery threshold* in coded computation.

Similar to binary error correcting codes, we can also have *systematic code* in the real domain, in which the linear coefficients satisfy

$$g_{i,j} = \mathbf{1}_{\{i=j\}}, \text{ if } i \le K. \tag{2.10}$$

In this case, some of the data data is the original data $\mathbf{X}_i, i = 1, 2, \ldots, L$.

2.2.5 Linear programming decoding

In Section 2.2.4, we focus on the case when the failed computation nodes can be detected. In what follows, we briefly review the linear programming decoding method in [44] for real-number coding with undetectable errors. These two types of failures are similar to erasures and undetectable bit flips respectively in communication theory.

We consider the case when a real-number vector $\mathbf{m} \in \mathbb{R}^K$ is encoded using a N-by-K generator matrix $\mathbf{G}$, and a vector $\mathbf{Gm} \in \mathbb{R}^N$ is generated. The encoded vector $\mathbf{Gm}$ is assumed to be corrupted by a sparse error vector $\mathbf{e} \in \mathbb{R}^N$, which satisfies

$$\|\mathbf{e}\|_0 := |\{i : e_i \neq 0\}| \leq \alpha \cdot N, \tag{2.11}$$

where $0 < \alpha < 1$ is a constant. The original message $\mathbf{m}$ is to be retrieved from the corrupted encoded message $\mathbf{r} = \mathbf{Gm} + \mathbf{e}$.

This problem can be readily connected to compressive sensing because, as shown in [44], if we multiply an $(N-K)$-by-N matrix $\mathbf{H}$ before the corrupted message $\mathbf{r}$, which is precalculated before communication/computation and satisfies $\mathbf{HG} = \mathbf{0}$, we get

$$\mathbf{Hr} = \mathbf{He}, \tag{2.12}$$

which is a classical compressive sensing problem, as the LHS is fixed and the vector $\mathbf{e}$ on the RHS is a sparse vector. The decoding can be implemented by solving an ℓ_1 minimization problem

$$(\mathbf{P.1}) \qquad \min_{\mathbf{m} \in \mathbb{R}^K} \|\mathbf{r} - \mathbf{Gm}\|_{\ell_1}, \tag{2.13}$$

which is equivalent to solving a linear programming problem

$$\min_{\mathbf{t} \in \mathbb{R}^N, \mathbf{m} \in \mathbb{R}^K} \mathbf{1}_N^\top \mathbf{t}, \qquad -\mathbf{t} \leq \mathbf{r} - \mathbf{Gm} \leq \mathbf{t}. \tag{2.14}$$

Suppose the matrix $\mathbf{H}$ satisfies the RIP constraints [44]:

$$(\mathbf{A.1}) \qquad (1-\delta_S)\|\mathbf{c}\|_2^2 \leq \|\mathbf{H}_{\mathcal{T}}\mathbf{c}\|_2^2 \leq (1+\delta_S)\|\mathbf{c}\|_2^2, \tag{2.15}$$

for all subsets $\mathcal{T} \subset \{1, 2, \ldots, N\}$ of cardinality at most S, and all vectors $\mathbf{c}$ with real coefficients $(c_j)_{j \in \mathcal{T}}$, where the constant $\delta_S \in [0,1]$ is called the S-restricted isometry constant, and

$$(\mathbf{A.2}) \qquad |\langle \mathbf{H}_{\mathcal{T}}\mathbf{c}, \mathbf{H}_{\mathcal{T}'}\mathbf{c}' \rangle| \leq \theta_{S,S'} \cdot \|\mathbf{c}\| \, \|\mathbf{c}'\|, \tag{2.16}$$

for all disjoint subsets $\mathcal{T}, \mathcal{T}' \subset \{1, 2, \ldots, N\}$ of cardinality $|\mathcal{T}| < S$ and $|\mathcal{T}'| < S'$, and for all vectors $\mathbf{c}$ and $\mathbf{c}'$ with real coefficients $(c_j)_{j \in \mathcal{T}}$ and $(c'_k)_{k \in \mathcal{T}'}$, where the constant $\theta_{S,S'}$ is called the S, S'-restricted orthogonality constant.

The following result shows that the ℓ_1 minimization problem recovers $\mathbf{m}$ exactly.

Lemma 2.2.5. *Suppose* $\mathbf{H}$ *is such that* $\mathbf{HG} = \mathbf{0}$ *and let* $S \geq 1$ *be an integer satisfying*

$$\delta_S + \theta_{S,S} + \theta_{S,2S} < 1, \tag{2.17}$$

where δ_S *and* $\theta_{S,2S}$ *are respectively defined in* (2.15) *and* (2.16). *Suppose* $\mathbf{r} = \mathbf{Gm} + \mathbf{e}$, *where* $\mathbf{e}$ *is supported on a sparse set of cardinality at most* S, *then, the minimization problem* (2.13) *recovers* $\mathbf{m}$ *exactly.*

In [44], it is shown that for random Gaussian matrices $\mathbf{H}$, as long as the ratio $\alpha = S/N$ is small enough while still being positive, the condition (2.17) holds with high probability in the limit of large N.

2.3 Related works

2.3.1 Common techniques in computing systems to address unreliability and heterogeneity

Check-pointing and replication: The built-in fault-tolerance techniques in distributed systems often respond in the following three ways to failure:

- *Stop-the-world*, or checkpointing [115, 197], which are deployed in the widely used Spark platform [292]. The entire system rolls back to the point when remaining machines have all data to recover the computation completely.
- Replication-based techniques, such as modular redundancy and state-machine replication [84, 163, 254]. The straggling effect of the slow workers [63, 259, 260] can often be addressed by replicating tasks across workers and using this redundancy to ignore some of the stragglers.
- Simply ignoring the failure. This method can lead to algorithm-level performance degradation for certain applications [180].

These techniques are based on the inherent assumption that machine failures are rare, while some robustness issues can happen on a much larger scale. For example, for elastic computing, the number of preempted machines may be more than half of all the machines if a greedy bidding strategy is used in a spot instance market (see Chapter 5).
Dynamic task allocation techniques: For task-level failures or stragglers, dynamic task allocation is useful [259, 260]. The task scheduler can choose which tasks to replicate, relaunch based on task profiles, and delay the relaunching to save time [9, 10, 12, 90, 242]. However, recovering from machine failures requires both restoring the machine states, downloading the data and installing individual software packages, which is usually time-consuming. The situation gets worse if failures events are frequent.
Resource-allocation in heterogeneous settings: Knowing task-specific properties can help improve resource allocation. For example, task replication may consume less resource if only small tasks are cloned [12]. When we have knowledge profiles about the functions, we can exploit it to achieve further gains beyond the general coding approaches (see Section 5.4.4 of Chapter 5).

2.3.2 Algorithm-based fault tolerance (ABFT)

Outside information theory, fault-tolerant linear transformations and related matrix operations have been studied extensively in algorithm-based fault tolerance [15, 37, 51, 68, 118, 125, 209, 238, 263]. In some of our works [273, 275, 278], we assume that the faults can happen at the gate-level, e.g., in AND gates and XOR gates. Instead, in ABFT, each functional block, e.g. a vector inner product, fails with a constant probability. Coding-inspired techniques for computing systems in recent years can be viewed as an advance to classical ABFT techniques, and often there are scaling-sense advantages by the use of sophisticated codes.

2.3.3 Coded computing

Coded computing is a candidate solution to deal with many robustness issues in distributed computing, as they have been shown to achieve low-redundancy fault-tolerance in both theory and practice. Coded computing is very general and can deal with machine failures [62, 92, 183], stragglers [63, 155], soft errors [1, 91, 112, 136, 159, 175, 297], communication bottlenecks [55, 296] and the respective solutions at exascale [26, 162]. Some problems of interest in coded computing include matrix multiplications [10, 19, 30, 74, 80, 83, 148, 150, 151, 168, 210, 223, 241, 261, 262, 273, 278, 281, 289, 290, 291], distributed regression [130, 131, 167], inverse problems and iterative solver of linear models [104, 272, 277, 280], Fourier transform [121, 288], covolution [75, 276], and deep neural networks [73, 139]. These techniques cover many common distributed computing primitives. Coded computing techniques can also reduce the communication time and shuffling time in distributed systems [18, 110, 156, 157, 208, 243, 243, 286]. In many cases, coded computing achieves scaling-sense speedups in average computation time compared to replication and other techniques.

2.3.4 Sparse coded computing

In the problem of coding for gradient-descent-type algorithms [47, 110, 130, 131, 208, 243, 286], the codes are often required to be sparse. For gradient coding on nonlinear gradient functions, a sparse code can reduce the storage overhead and the computational cost. Another reason to use sparse codes in practice is that the data is often sparse. If the coding matrix is dense, the encoded data becomes dense as well.

In Chapter 7, we focus on coded computing using a sparse code. Comparing to coding for gradient descent, Chapter 7 focuses on a more general framework of iterative computing which includes but is not limited to gradient descent. For example, power iterations and the more general Jacobi iterations and orthogonal iterations (see Section 7.4.1) are not gradient-descent-type methods and they are included in Chapter 7.

If the exact computation is required in coded computing, there is a tight lower bound on the number of non-zeros in a sparse code, and the bound is linear in the number of erasures. This suggests that computing using a sparse code can have severe limitations in dealing with a large number of failures. In Chapter 7, we focus on the regime where the encoding matrix is extremely sparse, which means the number of ones in each row of the encoding matrix is a constant (2 to 3) and does not increase with the number of erasures. Our technique is to utilize the intrinsic properties of iterative computation itself to provide good error correction ability and compensate for the weakness of sparse codes.

Some other works also focus on coded computing for graph analytics [201] and sparse matrix multiplications [261].

2.3.5 Iterative computation, inverse problems and spectral analysis

In Chapter 6 and Chapter 7, we show coding-inspired techniques to deal with computation failures and stragglers in distributed computing of linear inverse problems. We focus on iterative methods [217] that are designed to solve these inverse problems efficiently. The iterative problems that we consider include personalized PageRank [113, 190] and signal recovery on large graphs [49, 179, 264]. A typical example of iterative computing is the power-iteration method that repeatedly multiplies the intermediate result with the data matrix until convergence. Apart from PageRank, it is also applied in semi-supervised learning [299] and clustering [160]. Compared to single-shot computations, iterative problems have different optimization metrics and different properties for coded computing to utilize. For example, one may study the convergence rate of iteration computing, instead of the recovery threshold. The importance of the convergence-rate for iterative computing in the context of computing with stragglers has also been well-recognized by recent works [76, 131].

Spectral analysis on large and sparse matrices, such as computing eigenvectors and the singular value decomposition, can also be conducted in an iterative way [27, 94, 111, 122, 133, 215, 239]. For example, in Chapter 7, we show that coded computing can make the orthogonal-iteration method robust to erasure-type failures, which can be applied to eigendecompositions and singular value decompositions.

2.3.6 Communication avoiding distributed computing

Communication efficiency is one of the most import criteria in distributed computing. The research can be categorized into single-round schemes [114, 171, 173, 295, 300], multi-round schemes such as Disco [294], Dane [225], COCOA$^+$ [164] and accelerated gradient method [184], and coding-based approaches [105, 156, 243, 286]. There are also many works on communication avoiding linear algebra algorithms [21, 64, 248]. In our works, we aim to connect coding techniques with these techniques, and characterize the tradeoff between robustness, storage redundancy, and communication efficiency.

2.3.7 Noisy computing theory

The theory of robust computation with noisy components is started by the work of von Neumann's repetition-based construction [185] where an error-correction stage follows each computation stage to keep errors suppressed. Subsequent works [70, 194, 196] focus on minimizing the number of redundant gates, or the number of logic operations while making error probability below a small constant. In all of these prior works, it is usually assumed that both the computation units and the error-correction units (e.g., majority gates) are unreliable. However, the focus of these prior works is to study how a circuit is constructed to compute certain functions. It does not allow preprocessing steps to build some of the redundancy parts into the circuit in a reliable fashion. In Chapter 3, we will see that the difference in our work is that we allow (noiseless) precomputation based on the knowledge of the required function, which our scheme

(ENCODED) explicitly relies on. This difference in problem formulation is also why some of our achievable results on computational complexity might appear to beat the lower bounds of [70, 194, 196]. Therefore, our results are applicable when the same function needs to be computed multiple times for (possibly) different inputs, and thus the one-time cost of a precomputation is worth paying for. In [194, Theorem 4.4], Pippenger designed an algorithm to compute a binary linear transformation with noisy gates. The algorithm requires gates with unrealistically high fan-in and extremely low gate-error probability, which is too restricted for practical implementations.

Hadjicostis [108, 109] introduces finite-state fault-tolerant linear systems. The reliable computation in [109] uses coding to correct state errors after each state transition. Error control coding is also used in fault-tolerant parallel computing [238], AND-type one-step computing with unreliable components [202] and applied to error-resilient systems on chips (SoCs) [28]. The information-theoretic capacity of noisy computation is studied in [231, 232].

The study on noisy circuits and computing systems [109, 194, 253] often focus on finite fields. However, finite-field computing systems have limited applicability. It is therefore of immense practical interest to extend ideas of finite-field noisy computing, possibly with error control coding, to real-number settings. This need is widely recognized in the study of coded computing. For example, for soft-error correction in the real-domain fault-tolerant computing, we can use error control coding inspired from the field of compressive sensing [20, 43, 44, 158, 219, 270, 293].

2.3.8 Fault-tolerance in encoding, decoding, and storage systems

There are a large amount of works on applying error correcting codes to fault-tolerant encoders and decoders [71, 117, 244, 249, 249, 285]. Noisy decoders performing message-passing algorithms can be analyzed using the density evolution technique [152, 212] as well but the errors that happen during iterative decoding have to be take care of. In our earlier work [273], the problem of reliable communication with a noisy encoder is studied. In [273], noisy decoders are embedded in the noisy encoder to suppress errors repeatedly. The noisy encoding problem is a special case of computing matrix-vector multiplication when the matrix is the generator matrix of an error-correcting code, and the vector is the message. In [103], which considers a similar problem, errors are modelled as erasures on the encoding Tanner graph, and both worst-case and probabilistic analysis of noisy encoding is provided.

Closer in the spirit of robust computation with unreliable elements, in [53, 144, 245], the decoders (though not the encoders) for storage are assumed to be noisy as well. For storage, which can be viewed as computing the identity function, Low-Density Parity-Check (LDPC) codes [87] and Expander codes [233] have been used to correct errors.

2.3.9 Fault-tolerant signal processing and system processing

Researchers from the signal processing community have studied different problems regarding the design of fault-tolerant systems. For example, in [209], faults in a linear system are detected using Kalman-filtering based estimation. In [192, 203], adaptive filters with redundant tap coefficients were designed to deal with faults in these coefficients. Fault-detection algorithms on circuits and systems with unreliable computation units have also been studied extensively [8, 38, 54, 107, 204]. These algorithms often assume that the detection units are reliable. The field of fault-tolerant signal processing also includes deconvolution [145] and digital filtering [228].

2.3.10 Energy consumption in coding techniques

Energy consumption in decoding algorithms is an important field of study [31, 32, 99, 120]. While ignoring encoding and decoding costs is reasonable in long-range noisy communication problems [99], where the required transmit energy tends to dominate encoding/decoding computation energy, this can yield unrealistically positive results in short-range communication [97, 98, 99] and noisy computing [96], especially in the context of energy. The works cited above derive fundamental limits for simplistic implementation models that account for total energy consumption, including that of encoding and decoding. One can also use adaptive ways to reduce the energy consumption [120, 278].

2.3.11 Information dissipation in computing systems

Understanding information propagation is extremely useful in the understanding of robust computing systems. The aim of understanding how information flows in cascade channels and noisy circuits leads to the concept of *information dissipation* [79, 198, 279] which has been studied extensively from an information-theoretic viewpoint. These results often characterize and quantify the gradual loss of information as it is transmitted through noisy cascaded channels using information-theoretic lower bounds. In many classical network information theory problems, such as relay networks, the dissipation of information is not observed because it can be suppressed by use of asymptotically infinite blocklengths. The dissipation of information also cannot be quantified easily using classical information-theoretic tools that rely on the law of large numbers. This study has also yielded *strong* data processing inequalities that are essential for the understanding of information propagation [14, 42, 77, 205], and are beyond those commonly used in classical information theory.

Our works [273, 278] show that under some conditions, error-correcting codes can be used to overcome information dissipation and achieve reliable linear computation using unreliable circuit components. This is through combining information from multiple computation paths and mixing the information to compensate for the information loss on one path.

2.3.12 In-network function computation

From an information-theoretic and in particular rate-distortion viewpoint, information propagation is often studied from the perspective of distributed source coding for source reconstruction or function computation. The related works in this area can be categorized using the network structure, e.g., Gaussian multiple-access networks [237], three-node relay networks [221], CEO-type function computing networks [141, 255, 256], line or tree networks [17, 176, 222, 236, 251], and general lossless communication networks [128, 140]. The lossy function computation problem considered in Chapter 4 can be viewed as a generalization of these prior works for the specific purpose of understanding distortion accumulation in multi-stage computation networks.

The content in Chapter 4 is mainly inspired by the works by Su, Cuff and El Gamal [60, 240], which concerns outer bounds based on cut-set techniques in computation networks. This line of works often study the fundamental limits on the rate or the computation time that is required to meet specific fidelity requirements on function computation. However, our work is beyond cut-set bounds and shows that the outer bounds in [60, 240] can be significantly tightened by carefully examining the incremental distortions in multi-stage computing networks.

Problems of in-network computing have also been extensively studied for the goal of distributed data aggregation and distributed function computing [17, 60, 67, 93, 140, 141, 182, 222, 237, 247, 269, 279]. Some of the works consider the most communication-intensive function: the identity function (see, e.g. [86, 95, 154, 178]). The results are often obtained under specific assumptions on the network structure, including complete networks [86, 95, 143, 186, 258], grid networks [132], random geometric networks [72, 126, 129, 154, 287, 298] and tree networks [81, 274, 279]. There are three major computation models in the field of in-network computing: one-shot computation [86, 95, 126, 129, 132, 143, 154, 186], block computation [93, 140, 274, 279] and pipelined computation [16, 22, 124, 134, 170].

Chapter 3

Achievable result for the gate-level failure model: binary matrix-vector multiplication using entirely unreliable components

3.1 Introduction

Why do we study the gate-level failure model? One reason is that noise and variation issues in modern low-energy and low-area semiconductor devices require new design principles of circuits and systems [36]. For example, beyond 22nm is the region of extreme circuit component variations (see Figure 1.2). The saturation of "Dennard's scaling" of energy with smaller technology [65] also means that reducing CMOS transistor size no longer leads to a guaranteed reduction in energy consumption.

While most modern implementations use overwhelmingly reliable transistors, we will explore an appealing idea in this chapter, which is to allow errors in computation deliberately, and design circuits and systems that can utilize the random nature of these devices to reduce energy consumption. More specifically, in this Chapter, we investigate the problem of reliable computation of binary matrix-vector multiplications using circuits built entirely out of unreliable components, including the circuitry for introducing redundancy and correcting errors. The results of this chapter mainly appear in two papers [273, 278].

We will introduce the "ENCODED" technique (**En**coded **Co**mputation with **De**coders Embedde**D**), in which noisy decoders are embedded inside the noisy encoder to repeatedly suppress errors (see Section 3.4 for an explanation). The entire computation process is partitioned into multiple stages by utilizing the properties of an encoded form of the matrix to be multiplied (see Section 3.3.1 for details). In each stage, errors are introduced due to gate failures and then suppressed by embedded noisy decoders, preventing them from accumulating. Intuition on why embedded decoders are useful is provided in Section 3.3.2. The problem formulation and reliability models are detailed in Section 3.2.

We consider both probabilistic error models (transient gate errors) and permanent-errors models (defective gates).

In Section 3.3 and 3.4, we provably show that using ENCODED with LDPC decoders, an $L \times K$ matrix-vector multiplication can be computed with $\mathcal{O}(L)$ operations per output bit, while the output bit error probability is maintained below a small constant that is independent of L and K. In Section 3.4.1, we use expander LDPC codes to achieve worst-case error tolerance, while still using error-prone decoding circuitry. We show that ENCODED can tolerate defective gates errors as long as the fraction of defective gates is below a small constant. In Section 3.6, we use simulations to show that using exactly the same types of noisy gates (even with the same fan-in), the achieved bit error ratio and the number of iterations of ENCODED are both smaller than those of repetition-based schemes. Since computing energy is closely related to the number of operations, this shows an energy advantage of the ENCODED technique as well.

As we have mentioned in Section 1.2.2, energy consumption in circuit and systems is an important aspect to consider for the algorithm design. Therefore, in Section 3.5, we go a step further and systematically study the effect of the tunable supply voltage ("dynamic" voltage scaling) on the total energy consumption by modeling energy-reliability tradeoffs at the gate-level. For dynamic scaling, the gates are no longer homogeneous. We will talk about the theoretical modeling on the energy-error tradeoff of spintronic devices [41, 135, 169], which shows a possible application to tradeoff energy consumption and reliability of simple logic components. We introduce a two-phase algorithm in which the first phase is similar to ENCODED with homogeneous gates, but in the second phase, the voltage (and hence gate-energy) is tuned appropriately, which leads to orders of magnitude energy savings when compared with "static" voltage scaling (where the supply voltage is kept constant through the entire computation process). We show that, when the required output bit error probability is fixed, for polynomial decay of gate error probability ϵ with gate energy E (i.e., $\epsilon = \frac{1}{E^c}$), the energy consumption per output bit of ENCODED with dynamic voltage scaling is in scaling sense smaller than ENCODED with static voltage scaling (we note that energy for ENCODED with static voltage scaling is still smaller than *uncoded* with static voltage scaling).

3.2 System model and problem formulation

In this section, we provide detailed explanations on the noisy circuit computation model.

3.2.1 Circuit model

We first introduce unreliable gate models and circuit models that we will use in this chapter. We consider two types of unreliable gates: probabilistic gates and defective gates.

Definition: (Gate Model I (D, ϵ)) The gates in this model are probabilistically unreliable

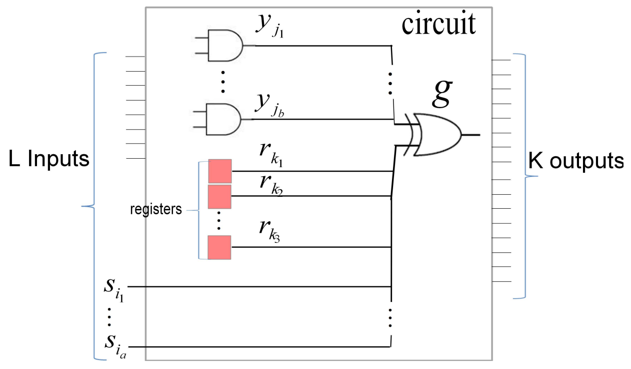


Figure 3.1: An unreliable gate g (Gate Model I or II)

in that they compute a deterministic boolean function g with additional noise z_g

$$y = g(u_1, u_2, ..., u_{d_g}) \oplus z_g, \tag{3.1}$$

where d_g denotes the number of inputs and is bounded above by a constant $D > 3$, $\oplus$ denotes the XOR-operation and z_g is a boolean random variable which takes the value 1 with probability ϵ which is assumed to be smaller than $\frac{1}{2}$. The event $z_g = 1$ means the gate g fails and flips the correct output. Furthermore, in this model, all gates fail independently of each other and the failure events during multiple uses of a single gate are also independent of each other. We allow different kinds of gates (e.g. XOR, majority, etc.) to fail with different probabilities. However, different gates of the same kind are assumed to fail with the same error probability.
This model is similar to the one studied in [253] and the failure event is often referred to as a transient fault. Our next model abstracts defective gates that suffer from permanent failures.
Definition: (Gate Model II (D, n, α)) In a set of n gates, each gate is either perfect or defective. A perfect gate always yields a correct output function

$$y = g(u_1, u_2, ..., u_{d_g}), \tag{3.2}$$

where d_g denotes the number of inputs and is bounded above by a constant $D > 3$. A defective gate outputs a deterministic boolean function of the correct output $\tilde{y} = f(g(\cdot))$. This function can be either $f(x) = \bar{x}$ (NOT function), $f(x) = 0$ or $f(x) = 1$ (also known as the "stuck-at error"when a circuit wire gets shorted). The fraction of defective gates in the set of n gates is denoted by α. Defective gates and perfect gates are not distinguishable.
The computation in a noisy circuit is assumed to proceed in discrete steps for which it is helpful to have circuits that have storage components.
Definition: (Register) A register is an error-free storage unit that outputs the stored binary value. A register has one input. At the end of a time slot, the stored value in a register is changed to its input value if this register is chosen to be *updated*.
Remark 3. It is relatively straightforward to incorporate in our analysis the case when registers fail probabilistically. A small increase in error probability of gates can absorb the error probability of registers.

Definition: (Noisy Circuit Model $(\mathcal{G}, \mathcal{R})$) A noisy circuit is a network of binary inputs $\mathbf{s} = (s_1, s_2, ...s_L)$, unreliable gates $\mathcal{G} = \{g_1, g_2, ..., g_{\mathscr{S}}\}$ and registers $\mathcal{R} = \{r_1, r_2, ..., r_{\mathscr{T}}\}$. Each unreliable gate $g \in \mathcal{G}$ can have inputs that are elements of **s**, or outputs of other gates, or from outputs of registers. That is, the inputs to an unreliable gate g are $s_{i_1}, \ldots, s_{i_a}, y_{j_1}, \ldots, y_{j_b}, r_{k_1}, \ldots, r_{k_c}$, where $a + b + c = d_g$, the total number of inputs to this gate. Each register $r \in \mathcal{R}$ can have its single input from the circuit inputs **s**, outputs of unreliable gates or outputs of other registers. For simplicity, wires in a noisy circuit are assumed to be noiseless.

Definition: (Noisy Computation Model $(L, K, \mathscr{N}_{\text{comp}})$) A computing scheme $\mathcal{F}$ employs a noisy circuit to compute a set of binary outputs $\mathbf{r} = (r_1, r_2, ...r_K)$ according to a set of binary inputs $\mathbf{s} = (s_1, s_2, ...s_L)$ in multiple stages. At each stage, a subset of all unreliable gates $\mathcal{G}$ are *activated* to perform a computation and a subset of all registers $\mathcal{R}$ are updated. At the completion of the final stage, the computation outputs are stored in a subset of $\mathcal{R}$. The number of *activated* unreliable gates in the t-th stage is denoted by $\mathscr{N}_{\text{comp}}^t$. Denote by $\mathscr{N}_{\text{comp}}$ the total number of unreliable operations (one unreliable operation means one activation of a single unreliable gate) executed in the noisy computation scheme, which is obtained by

$$\mathscr{N}_{\text{comp}} = \sum_{t=1}^{T} \mathscr{N}_{\text{comp}}^t, \tag{3.3}$$

where T is the total number of stages, which is predetermined.

The noisy computation model is the same as a sequential circuit with a clock. The number of stages T is the number of time slots that we use to compute the matrix-vector multiplication. In each time slot t, the circuit computes an intermediate function $f_t(x)$ using the computation units on the circuit, and the result $f_t(x)$ is stored in the registers for the computation in the next time slot $t + 1$. The overall number of stages T is predetermined (fixed before the computation starts). A computing scheme should be *feasible*, that is, in each time slot, all the gates that provide inputs to an activated gate, or a register to be updated, should be activated. In this chapter, we will only consider noisy circuits that are either composed entirely of probabilistic gates defined in Gate Model I or entirely of unreliable gates in Gate Model II.

3.2.2 Problem statement

The problem considered in this chapter is that of computing a binary matrix-vector multiplication $\mathbf{r} = \mathbf{s} \cdot \mathbf{A}$ using a noisy circuit, where the input vector $\mathbf{s} = (s_1, s_2, ...s_L)$, the output vector $\mathbf{r} = (r_1, r_2, ...r_K)$ and the L-by-K matrix $\mathbf{A}$ are all composed of binary entries. We consider the problem of designing a feasible computing scheme $\mathcal{F}$ for computing $\mathbf{r} = \mathbf{s} \cdot \mathbf{A}$ with respect to Definition 3.2.1. Suppose the correct output is $\mathbf{r}$. Denote by $\hat{\mathbf{r}} = (\hat{r}_1, \hat{r}_2, ...\hat{r}_K)$ the (random) output vector of the designed computing scheme $\mathcal{F}$. Note that the number of operations $\mathscr{N}_{\text{comp}}$ has been defined in Definition 3.2.1. The computational complexity per bit $\mathscr{N}_{\text{per-bit}}$ is defined as the total number of operations per output bit in the computing scheme. That is

$$\mathscr{N}_{\text{per-bit}} = \mathscr{N}_{\text{comp}}/K. \tag{3.4}$$

For gates from Gate Model I (Definition 3.2.1), we are interested in keeping the fraction of (output) errors bounded with high probability. This could be of interest, e.g., in approximate computing problems. To that end, we define another metric, δ_e^{frac}, the "bit-error fraction," which is simply the Hamming distortion between the computed output and the correct output (per output bit). That is, $\delta_e^{\text{frac}} = \max_{\mathbf{s}} \frac{1}{K} \sum_{k=1}^{K} \mathbb{1}_{\{\hat{r}_k \neq r_k\}}$, where $\mathbb{1}_{\{\cdot\}}$ is the indicator function. The bit-error fraction depends on the noise, which is random in Gate Model I. Thus, we will constrain it probabilistically (see *Problem 1*[1]). The resulting problems are stated as follows:
Problem 1.

$$\min_{\mathcal{F}} \quad \mathscr{N}_{\text{per-bit}}, \text{ s.t. } \Pr(\delta_e^{\text{frac}} < p_{\text{tar}}) > 1 - \delta, \tag{3.5}$$

where $p_{\text{tar}} > 0$ is the target block error fraction and δ is a small constant.

When we consider the Gate Model II (Definition 3.2.1), since all gates are deterministic functions, we are interested in the worst-case fraction of errors δ_e^{frac}. Thus, the optimization problem can be stated as follows:
Problem 2.

$$\min_{\mathcal{F}} \quad \mathscr{N}_{\text{per-bit}}, \text{ s.t. } \max_{\mathbf{s}, \mathcal{S}_{\text{def}}^i \text{ s.t.} |\mathcal{S}_{\text{def}}^i| < \alpha_i n_{\mathcal{F},i}, \forall i \in W} \delta_e^{\text{frac}} < p_{\text{tar}}, \tag{3.6}$$

where s is the input vector, $\mathcal{S}_{\text{def}}^i$ is the set of defective gates of type i, W is the set of indices of different types of noisy gates (such as AND gates, XOR gates and majority gates), α_i is the error fraction of the gates of type i, $n_{\mathcal{F},i}$ is the total number of gates of type i in the implementation of $\mathcal{F}$, and $p_{\text{tar}} > 0$ is the target fraction of errors. Note that $n_{\mathcal{F},i}$ is chosen by the designer as a part of choosing $\mathcal{F}$, while the error-fraction α_i is assumed to be known to the designer in advance.

3.2.3 Technical preliminaries

We will use error correcting codes to facilitate the computation of the binary matrix-vector multiplication. We will use a regular LDPC code [87] with code length N, dimension K and a $K \times N$ generator matrix $\mathbf{G}$. Please refer to Section 2.2.2 for more details on the LDPC codes and the decoding algorithms. The embedded decoders use either the Gallager-B decoding algorithm or the parallel bit flipping (PBF) algorithm.

Since we will consider noisy decoders, we will build on a more refined result, which concerns a single decoding iteration of the algorithm (see the requirement (A.3) in Section 2.2.2). The requirement in (A.3) can be met by using (d_v, d_c)-regular random code ensembles and using the analysis in [39], which we will show in details (see Lemma A.3.2). In particular, we show that almost all codes in the $(9, 18)$-regular code

[1]We will show that the bit-error fraction is constrained probabilistically (see *Problem 1*) for all input vector s.

ensemble of sufficiently large length N can reduce the number of errors by $\theta = 15\%$ after one iteration of the PBF algorithm, if the fraction of errors is upper-bounded by $\alpha_0 \le 5.1 \cdot 10^{-4}$. We also show that at least 4.86% of the $(9,18)$-regular codes of length $N = 50,000$ can reduce the number of errors by $\theta = 15\%$ after one iteration of the PBF algorithm, if the number of errors satisfies $\alpha_0 N \le 20$, which is equivalent to $\alpha_0 \le 0.0004$.

3.3 A simplified version of main results

In this section, we present the overview of the main scheme that we use for noisy computation of matrix-vector multiplications. We call this scheme "ENCODED" (**En**coded **Co**mputation with **De**coders Embedde**D**).

3.3.1 ENCODED: a multi-stage error-resilient computation scheme

Instead of computing a binary matrix-vector multiplication $\mathbf{r} = \mathbf{s} \cdot \mathbf{A}$ without using any redundancy, we will compute

$$\mathbf{x} = \mathbf{r} \cdot \mathbf{G} = \mathbf{s} \cdot \mathbf{AG}, \tag{3.7}$$

where $\mathbf{G} = [\mathbf{I}, \mathbf{P}] = [\mathbf{g}_1; \mathbf{g}_2; ...; \mathbf{g}_K]$ is the $K \times N$ generator matrix of the chosen systematic LDPC code. The matrix product $\mathbf{AG}$ is assumed to be computed offline in a noise-free fashion. An important observation is that since all rows in the matrix product $\mathbf{AG}$ are linear combinations of the rows in the generator matrix $\mathbf{G}$, the rows of $\mathbf{AG}$ are codewords as well. That is,

$$\tilde{\mathbf{G}} = \mathbf{AG} = \begin{bmatrix} \leftarrow & \tilde{\mathbf{g}}_1 & \rightarrow \\ \leftarrow & \tilde{\mathbf{g}}_2 & \rightarrow \\ \leftarrow & \ldots & \rightarrow \\ \leftarrow & \tilde{\mathbf{g}}_L & \rightarrow \end{bmatrix} \tag{3.8}$$

where each row $\tilde{\mathbf{g}}_l, l = 1, \ldots, L$ is a codeword. Then, if the computation were noiseless, the correct computation result $\mathbf{r} = \mathbf{s} \cdot \mathbf{A}$ could be obtained from the combined result

$$\mathbf{x} = [\mathbf{r}, \mathbf{r} \cdot \mathbf{P}] = \mathbf{r} \cdot \mathbf{G}. \tag{3.9}$$

Since $\mathbf{r} \cdot \mathbf{G} = \mathbf{s} \cdot \mathbf{AG} = \mathbf{s} \cdot \tilde{\mathbf{G}}$,

$$\mathbf{x} = \mathbf{s} \cdot \tilde{\mathbf{G}} = \sum_{l=1}^{L} s_l \tilde{\mathbf{g}}_l. \tag{3.10}$$

In the following sections, we will explain how error control coding can be used to reliably compute $\mathbf{x} = \sum_{l=1}^{L} s_l \tilde{\mathbf{g}}_l$. The basic idea is as follows: we break the computation into L stages, so that the noiseless intermediate result after the l-th stage would be $\mathbf{x}^{(l)} = \sum_{j=1}^{l} s_j \tilde{\mathbf{g}}_j$. When

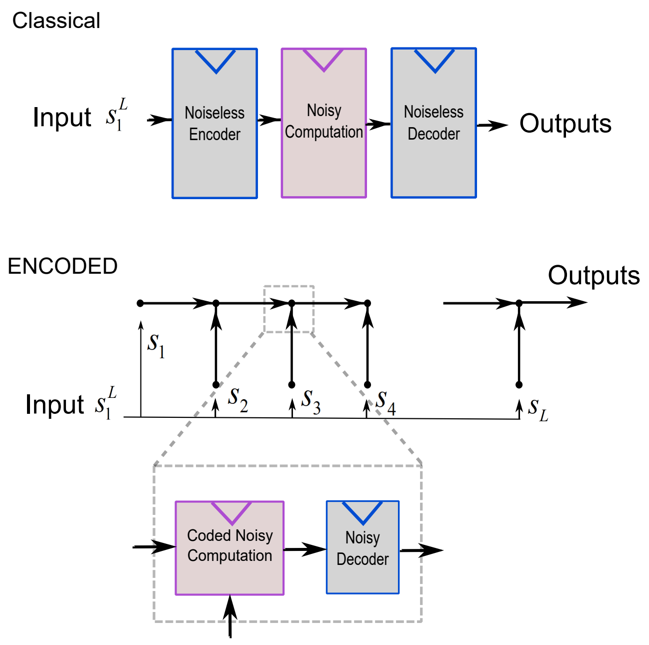


Figure 3.2: An illustration of the conceptual difference between classical noisy computing schemes and the ENCODED technique.

gates are noise-free, $\mathbf{x}^{(l)}$ is a codeword. When gates are noisy, during the l-th stage, we first compute $\mathbf{x}^{(l-1)} + s_l\tilde{\mathbf{g}}_l$ using noisy AND gates (binary multiplication) and noisy XOR gates (binary addition) and then correct errors (with high probability) using an LDPC decoder or an expander decoder to get $\mathbf{x}^{(l)}$. During the entire computing process, AND gates and XOR gates introduce errors, while the noisy decoders suppress errors. Finally, it will be proved in Theorem 3.4.2 that error probability is maintained below a small constant. We summarize the ENCODED technique in Algorithm 1.

Algorithm 1 ENCODED (**En**coded **Co**mputation with **De**coders Embedde**D**)

```
INPUT: A binary vector $\mathbf{s} = (s_1, s_2, ...s_L)$.
OUTPUT: A binary vector $\mathbf{x} = (x_1, x_2, ...x_N)$.
INITIALIZE
Compute $\tilde{\mathbf{G}} = \mathbf{AG} = [\tilde{\mathbf{g}}_1; \tilde{\mathbf{g}}_2; ...; \tilde{\mathbf{g}}_L]$. Store an all-zero vector $\mathbf{x}^{(0)}$ in an $N$-bit register.
FOR $l$ from 1 to $L$
    • Use $N$ unreliable AND gates to multiply $s_l$ with $\tilde{\mathbf{g}}_l$, the $l$-th row in $\tilde{\mathbf{G}}$, add this result
      to $\mathbf{x}^{(l-1)}$ using $N$ unreliable XOR gates, and store the result in the $N$-bit register.[2]
    • Use an unreliable decoder to correct errors and get $\mathbf{x}^{(l)}$.
END
Output $\mathbf{x}^{(L)}$ as the output x.
```

[7]These operations are assumed to be performed noiselessly, as discussed earlier.

3.3.2 Intuition underlying the embedded decoders

The basic idea of our proposed computing scheme is to split the computation into a multistage computation of $\mathbf{x} = \sum_{l=1}^{L} s_l \tilde{\mathbf{g}}_l$, and use embedded decoders inside the noisy circuit to repeatedly suppress errors as the computation proceeds. Since the noisy circuit can only be constructed using unreliable gates, the embedded decoders are also constituted by unreliable gates.

Why is such a multistage computation helpful? For instance, if "uncoded" matrix multiplication $\mathbf{r} = \mathbf{s}\mathbf{A}$ is carried out, each output bit is computed using an inner product, and $\mathcal{O}(L)$ unreliable AND and XOR-operations are required. Without repeated suppression, each output bit is erroneous with probability $\frac{1}{2}$ as $L \to \infty$. Intermediate and repeated error suppression alleviates this *error accumulation* problem.

Also note that due to the 'last-gate' effect in noisy circuits, error probability cannot approach zero. Thus, our goal is not to eliminate errors, but to suppress them so that the error probability (or the error fraction) is kept bounded below a target value that depends on the error probability of the last gate.

3.4 Main results on robust matrix-vector multiplication

In this section, we show that a matrix-vector multiplication can be computed 'reliably' (in accordance with the goals of Problems 1-2 in Section 3.2.2) even in presence of noise, using error correcting codes. We also compare resource requirements of this coding-based computation with repetition-based computation using simulations (in Section 3.6.) Note that for the purpose of clearly presenting the main intuition, we only include one algorithm called ENCODED-F. If the readers are interested, they can refer to the tree-based algorithm ENCODED-T for more details [278].

3.4.1 ENCODED-F: reliable computation of matrix-vector multiplications using noisy gates

We modify ENCODED as follows: we partition the entire computing process into $\left\lceil \frac{L}{d_s-1} \right\rceil$ stages, where d_s is called the *group size*. First, we store an all-zero codeword in the N-bit register. In the l-th stage, we first use $(d_s-1)N$ AND gates to obtain the d_s-1 scalar-vector multiplications $s_i \cdot \tilde{\mathbf{g}}_i$ for $i \in \{(d_s-1)(l-1)+1, (d_s-1)(l-1)+2, \ldots, (d_s-1)l\}$, where $\tilde{\mathbf{g}}_i$ is the i-th row of the combined matrix $\tilde{\mathbf{G}} = \mathbf{A}\mathbf{G}$. Then, we use N XOR gates to add the d_s-1 results to the N-bit register. The parameter d_s is chosen so that $d_s \leq D$, the maximum input to each noisy gate. After that, we use one iteration of the PBF algorithm (see Section 3.3) to correct errors. We use P XOR gates and N majority gates in one iteration of the PBF algorithm.

In what follows, we prove Theorem 3.4.1, which quantifies the error-tolerance of ENCODED-F. The basic tool used to prove Theorem 3.4.1 is a modified version of the worst-case error correcting result in the requirement (A.3), which provides the worst-case error correcting capability of regular LDPC codes using one iteration of noisy PBF decoding.

Theorem 3.4.1 (Error Suppression Using the PBF algorithm for Problem 3). *Using unreliable AND gates, XOR gates and majority gates from Gate Model II* (D, n, α) *with respective error fractions* α_{and}*,* α_{xor} *and* α_{maj} *respectively, and using an* (d_v, d_c)*-regular LDPC code which satisfies (A.3) to implement ENCODED-F with group size* d_s*, as long as*

$$(d_s-1)\alpha_{and} + [D(1-R)+1]\alpha_{xor} + \alpha_{maj} < \theta\alpha_0, \tag{3.11}$$

the binary matrix-vector multiplication $\mathbf{r} = \mathbf{sA}$ *can be computed using* N *AND gates,* $(N+P)$ *XOR gates, and* N *majority gates, and the number of operations per bit is at most* $\frac{2N+P}{K}\left\lceil \frac{L}{d_s-1}\right\rceil + \frac{NL}{K} = \Theta(\frac{LN}{K})$*. Further, the error fraction of the final output is at most* α_0*.*

Proof of Theorem 3.4.1. We use induction on the stage index l to derive an upper bound on the number of errors. In the first stage, $N(d_s-1)$ AND gates and N XOR gates introduce at most $N[(d_s-1)\alpha_{\text{and}} + \alpha_{\text{xor}}]$ errors, which is upper bounded by

$$(d_s-1)\alpha_{\text{and}} + \alpha_{\text{xor}} < \theta\alpha_0, \tag{3.12}$$

which can be obtained by combining (3.11). Suppose in the $(l-1)$-th stage, after adding a set of (d_s-1) codewords $s_i \cdot \tilde{\mathbf{g}}_i, i \in \{(d_s-1)(l-2)+1, (d_s-1)(l-2)+2, \ldots, (d_s-1)(l-1)\}$ to the N-bit register, the number of errors is strictly less than $N\alpha_0$.

Then, according to condition (A.3), if no computation errors occur during execution of one iteration of PBF algorithm, the fraction of errors can be reduced to $N\alpha_0(1-\theta)$. Whenever an XOR gate flips the corresponding parity check value during the PBF algorithm, it affects at most d_c majority gates. In total, there are P XOR gates used in one iteration of the PBF algorithm, so there are at most $\alpha_{\text{xor}}Pd_c$ errors due to XOR errors in the PBF algorithm. There are at most $\alpha_{\text{maj}}N$ errors due to majority gate failures. After this iteration of bit flipping, another set of (d_s-1) codewords $s_i\mathbf{g}_i$ is added to the N-bit register with $N(d_s-1)$ AND gates and N XOR gates. These two operations introduce $(\alpha_{\text{xor}} + \alpha_{\text{and}}(d_s-1))N$ errors. Therefore, the total error fraction before the next PBF algorithm is upper bounded (using the union bound) by

$$\begin{aligned} \alpha_{\text{PBF}} \leq N\alpha_0(1-\theta) + &[d_c(1-R)+1]\alpha_{\text{xor}} \\ &+\alpha_{\text{maj}} + (d_s-1)\alpha_{\text{and}}, \end{aligned} \tag{3.13}$$

where R is the code rate ($R = \frac{N-P}{N}$). As long as (3.11) holds and $d_c \leq D$, before the next bit flipping, it holds that

$$\alpha_{\text{PBF}} \leq N\alpha_0(1-\theta) + \theta N\alpha_0 = \alpha_0 N. \tag{3.14}$$

Therefore, the induction can proceed.

In each stage, we need $N+P$ XOR-operations and N majority-operations. During the entire computation, we need NL AND-operations. Therefore, the computational complexity per output bit, which is the total number of operations in $\lceil L/(d_s-1)\rceil$ stages divided by K bits, is $(2N+P)\lceil L/(d_s-1)\rceil/K + \frac{NL}{K}$. □

ENCODED-F can be applied to Gate Model I as well, which is characterized in the following theorem.

Theorem 3.4.2 (Error Suppression Using PBF algorithm for Problem 2)**.** *Using unreliable AND gates, majority gates and XOR gates from Gate Model I with respective gate error probabilities p_{and}, p_{xor} and p_{maj}, and using an (d_v, d_c)-regular LDPC code which satisfies (A.3) to implement ENCODED-F with group size d_s, as long as*

$$\begin{aligned}&\max\{p_{and}, p_{xor}, p_{maj}\}\\ &< \lambda := \frac{\theta\alpha_0/2}{(d_s-1)+[d_c(1-R)+1]+1},\end{aligned} \tag{3.15}$$

the binary matrix-vector multiplication $\mathbf{r} = \mathbf{s}\cdot\mathbf{A}$ can be computed using $\frac{2N+P}{K}\left\lceil\frac{L}{d_s-1}\right\rceil + \frac{NL}{K} = \Theta(\frac{LN}{K})$ operations per bit. Further, the final error fraction δ_e^{frac} satisfies

$$\Pr(\delta_e^{frac} < \alpha_0) > 1 - P_e^{blk}, \tag{3.16}$$

where the probability P_e^{blk} satisfies

$$P_e^{blk} < 3L\exp\left(-\lambda^* N\right), \tag{3.17}$$

where

$$\lambda^* = D(2\lambda\|\lambda) = (2\log 2 - 1)\,\lambda + \mathcal{O}(\lambda^2). \tag{3.18}$$

Proof. See Appendix A.1. □

Remark 4. The analysis of the PBF algorithm requires finding codes that satisfy Assumption (A.3) (see Appendix A.3), which still requires randomized code constructions. Another method to analyze the bit flipping algorithm is to use Expander codes (also see Appendix A.3). However, hardware-friendly expander codes tend to be hard to construct and use in practice, while many hardware-friendly LDPC codes have been designed. In fact, we included a tree-based algorithm ENCODED-T in [278] because ENCODED-T works for all regular LDPC codes and does not require the assumption that a code satisfying (A.3) exists. For a comparison between ENCODED-T and ENCODED-F, see [278].

The following converse result holds for all computation schemes. Although this converse result does not match with any of the achievable results listed above, it matches with an achievable result when a "noiseless decoder" is available (details will be provided in [278]) in the scaling of the target error probability p_{tar}. Thus, we believe the converse result captures the required computational complexity for the beginning stages of the matrix-vector multiplication computation.

Theorem 3.4.3 (Converse result)**.** *For Gate Model I with error probability ϵ, maximum fan-in D, and the matrix-vector multiplication $\mathbf{r} = \mathbf{s}\cdot\mathbf{A}$ with $\mathbf{A}$ having full row rank, in order to achieve P_e^{blk} smaller than p_{tar}, the number of operations required per bit is lower bounded as $\mathcal{N}_{\text{per-bit}} \geq \frac{L\log 1/p_{tar}}{KD\log D/\epsilon} = \Omega(\frac{L\log 1/p_{tar}}{K\log 1/\epsilon})$.*

Proof. See Appendix A.2. □

3.4.2 Theoretical comparison with repetition coding

In this chapter, although we obtain results on the number of operations for ENCODED in Theorem 3.4.2, the results are biased for the comparison between ENCODED and repetition-based schemes, because the number of operations do not take into account the gate fan-in. Therefore, to compare the complexity of operations with different fan-in, we define a new concept called "effective number of operations". We assume that the "effective number of operations" for an operation with fan-in c is $\mathscr{N}_{\text{c-fan-in}} = c$ (the analysis for a different $\mathscr{N}_{\text{c-fan-in}}$ can be done similarly). We show that if we consider the problem of **find a binary matrix-vector multiplication scheme that achieves target error probability $p_{\textbf{tar}} = 5.1 \cdot 10^{-4}$ using only noisy gates with $\max(p_{\textbf{xor}}, p_{\textbf{maj}}, p_{\textbf{and}}) < 1.3 \cdot 10^{-6}$, the effective number of operations of ENCODED-F is smaller than that of distributed majority voting, provided that the size of the matrix-vector multiplication satisfies $N = 2K > 9.85 \cdot 10^{7}$, and $L > \frac{p_{\textbf{tar}}}{p_{\textbf{and}}}$**. We choose these parameters only to show that ENCODED can provably beat repetition-based schemes in situations when the parameters are not absurdly large, and hence the theoretical analysis here has potential to provide practical insight. Here $\max(p_{\text{xor}}, p_{\text{maj}}, p_{\text{and}})$ is interpreted as the maximum error probability over all types of different gates, which allows the same type of gates (i.e., MAJ-gates) with different fan-in to have different error probabilities.

Counting the effective number of operations

First, we compare the effective number of operations in both schemes. For ENCODED-F, we use a $(9, 18)$ code. To make the comparison fair, we allow the distributed majority voting scheme to group several stages into one stage as well. Recall that we use d_s to denote the number of stages that ENCODED-F groups into one stage. Therefore, we use d_s' to denote the number of stages that distributed majority voting groups into one stage. In general, $d_s \neq d_s'$.

We compare ENCODED-F using (9,18) LDPC codes with distributed majority voting with three-time repetition. We show when

$$d_s > 14, \tag{3.19}$$

the "effective" number of operations in ENCODED-F is less than that of distributed majority voting. **Note that d_s' can be arbitrary**.

The number of compute-and-correct stages in ENCODED-F is $\left\lceil \frac{L}{d_s - 1} \right\rceil$, and that of distributed majority voting is $\left\lceil \frac{L}{d_s' - 1} \right\rceil$. For the ease of analysis, assume L is a multiple of both $d_s - 1$ and $d_s' - 1$. For ENCODED-F, in each compute-and-correct stage, we need N XOR-operations of fan-in d_s for binary addition, P XOR-operations of fan-in d_c and N MAJ-operations of fan-in d_v for LDPC decoding. In all compute-and-correct stages, the overall number of AND-operations of fan-in 2 is NL. Then,

$$\mathscr{N}^{\text{ENC}}_{\text{XOR}-\text{d}_\text{s},\text{per-bit}} = \frac{N}{K} \left\lceil \frac{L}{d_s - 1} \right\rceil = \frac{2}{d_s - 1} L, \tag{3.20}$$

$$\mathscr{N}^{\text{ENC}}_{\text{XOR}-\text{d}_\text{c},\text{per-bit}} = \frac{P}{K}\left\lceil \frac{L}{d_s-1}\right\rceil = \frac{1}{d_s-1}L, \tag{3.21}$$

$$\mathscr{N}^{\text{ENC}}_{\text{MAJ}-\text{d}_\text{v},\text{per-bit}} = \frac{N}{K}\left\lceil \frac{L}{d_s-1}\right\rceil = \frac{2}{d_s-1}L, \tag{3.22}$$

$$\mathscr{N}^{\text{ENC}}_{\text{AND}-2,\text{per-bit}} = \frac{NL}{K} = 2L. \tag{3.23}$$

In the distributed majority voting scheme with repetition time 3, the number of operations per output bit is

$$\mathscr{N}^{\text{Rep}}_{\text{XOR}-\text{d}'_\text{s},\text{per-bit}} = 3\left\lceil \frac{L}{d'_s-1}\right\rceil = \frac{3}{d'_s-1}L, \tag{3.24}$$

$$\mathscr{N}^{\text{Rep}}_{\text{MAJ}-3,\text{per-bit}} = 3\left\lceil \frac{L}{d'_s-1}\right\rceil = \frac{3}{d'_s-1}L, \tag{3.25}$$

$$\mathscr{N}^{\text{Rep}}_{\text{AND}-2,\text{per-bit}} = 3L. \tag{3.26}$$

Therefore, from (3.20)-(3.23), for ENCODED-F with $d_c = 18$ and $d_v = 9$, the effective number of operations is

$$\begin{aligned}\mathscr{N}^{\text{ENC}}_{\text{eff}} &= d_s \cdot \frac{2}{d_s-1}L + d_c \cdot \frac{1}{d_s-1}L + d_v \cdot \frac{2}{d_s-1}L + 2\cdot 2L \\ &= \frac{36+2d_s}{d_s-1}L + 4L = \frac{38}{d_s-1}L + 6L.\end{aligned} \tag{3.27}$$

From (3.24) to (3.26), for distributed majority voting, the effective number of operations is

$$\begin{aligned}\mathscr{N}^{\text{rep}}_{\text{eff}} &= d'_s \cdot \frac{3}{d'_s-1}L + 3 \cdot \frac{3}{d'_s-1}L + 2\cdot 3L \\ &= \frac{3d'_s+9}{d'_s-1}L + 6L = \frac{12}{d'_s-1}L + 9L > 9L.\end{aligned} \tag{3.28}$$

Therefore, when $d_s > 14$,

$$\mathscr{N}^{\text{ENC}}_{\text{eff}} < \frac{38}{13}L + 6L < 3L + 6L = 9L < \mathscr{N}^{\text{rep}}_{\text{eff}}. \tag{3.29}$$

Analyzing the probability of error

Now, we analyze the error probability of ENCODED-F for $d_s = 14$, $d_c = 18$ and $d_v = 9$. From Lemma A.3.1 in Appendix A.3, using almost all codes in a (d_v, d_c)-regular LDPC random code ensemble with $d_v > 4$ and N large enough, after one iteration of the PBF algorithm, one can reduce the number of errors by at least $\theta\alpha_0 N$ for any $\alpha_0 N$ worst-case errors if α_0 and θ are small enough. That is, using a (d_v, d_c)-regular LDPC code, the number of errors after one iteration of noiseless PBF algorithm will be smaller than

$\alpha_0 \cdot (1-\theta)$. Recall that this is the condition (A.3) that we require on the utilized LDPC code. In Example 1 in Appendix D, for the $(9,18)$-regular LDPC code, we computed numerically the threshold value of α_0 for $\theta = 0.15$ and obtained $\alpha_0 = 5.1 \cdot 10^{-4}$. We also obtained finite-length bounds which state that there exist $(9,18)$-regular LDPC codes with length $N = 50,000$ that can reduce the number of errors by 15% for an arbitrary pattern of at most 20 errors, which corresponds to the case when $\alpha_0 = 4 \cdot 10^{-4}$ and $\theta = 0.15$.

From Theorem 3, using the (9,18) code, when the maximum gate error probability $\epsilon = \max(p_{\text{xor}}, p_{\text{maj}}, p_{\text{and}})$ satisfies the condition

$$\begin{aligned} \epsilon < \lambda = & \frac{\theta\alpha_0/2}{(d_s-1)+[d_c(1-R)+1]+1} \\ = & \frac{\theta\alpha_0/2}{(14-1)+\left[18(1-\frac{1}{2})+1\right]+1} = \frac{\theta\alpha_0}{54}, \end{aligned} \tag{3.30}$$

ENCODED-F has bounded final error fraction with high probability, which is

$$1 - P_e^{\text{blk}} > 1 - 3L\exp\left(-D(2\lambda\|\lambda)N\right), \tag{3.31}$$

where $\lambda = \frac{\theta\alpha_0}{54}$ and $D(2\lambda\|\lambda) = (2\log 2 - 1)\,\lambda + \mathcal{O}(\lambda^2)$.

In particular, if we choose $\epsilon = \frac{1}{60}\theta\alpha_0 < \frac{1}{54}\theta\alpha_0 = \lambda$, the final error fraction satisfies $\delta_e^{\text{frac}} < \alpha_0 = \frac{60}{\theta}\cdot\epsilon$ with probability $1 - 3L\exp\left(-D(2\lambda\|\lambda)N\right)$. As we have mentioned, for $\theta = 0.15$, we obtain $\alpha_0 = 5.1 \cdot 10^{-4}$. Therefore, when the gate error probabilities satisfy $\max(p_{\text{xor}}, p_{\text{maj}}, p_{\text{and}}) = \epsilon = \frac{1}{60}\theta\alpha_0 = 0.0043\alpha_0 = 1.3 \cdot 10^{-6}$, the obtained error probability is smaller than $\alpha_0 = 60/\theta_0 \cdot \epsilon = 400\epsilon = 5.1 \cdot 10^{-4}$ with probability $1 - 3L\exp\left(-D(2\lambda\|\lambda)N\right)$, which is approximately 1 with reasonably large N, which can be guaranteed[3] if $N > \frac{50}{D(2\lambda\|\lambda)} \approx \frac{50}{(2\log 2-1)\cdot\lambda} = \frac{50}{(2\log 2-1)\cdot\frac{1}{60}\theta\alpha_0} = \frac{5.02\cdot 10^4}{\alpha_0} = 9.85 \cdot 10^7$.

Therefore, if we consider the problem "find a binary matrix-vector multiplication scheme that achieves target error probability $p_{\text{tar}} = \alpha_0 = 5.1 \cdot 10^{-4}$ using only noisy gates with $\max(p_{\text{xor}}, p_{\text{maj}}, p_{\text{and}}) < 1.3 \cdot 10^{-6}$", ENCODED-F has smaller "effective number of operations" than that of distributed majority voting. Additionally, one-time repetition or two-time repetition cannot obtain $p_{\text{tar}} = \alpha_0 = 5.1 \cdot 10^{-4}$ when L is reasonably large so that $\frac{1}{2}[1 - (1 - 2p_{\text{and}})^L] \approx Lp_{\text{and}} > \alpha_0$. Thus, we conclude that ENCODED-F beats repetition-based schemes under this circumstance. Here, we acknowledge that the problem parameters (such as $\max(p_{\text{xor}}, p_{\text{maj}}, p_{\text{and}}) < 1.3 \cdot 10^{-6}$ and $N > 9.85 \cdot 10^7$) are chosen to show that the theoretical analysis works even when the parameter sizes are not extremely large, and thus the theoretical analysis technique has the potential to provide practical insight.

[3]We believe that further optimization in code design can provide techniques for error suppression for even smaller value of N.

3.5 Main results on energy efficient matrix vector multiplication

In this section, we consider unreliable gates with tunable failure probability [78] when supply voltage, and hence energy consumed by gates, can be adjusted to attain a desired gate-reliability. To model this property within Gate Model I in (3.1), we assume that the added noise $z_g \sim$ Bernoulli($\epsilon_g(E_g)$), in which $\epsilon_g(E_v)$ is a function that depends on the supply energy E_v. We assume that E_v is identical for all gates at any stage of the computation, while it can vary across stages. Intuitively, $\epsilon_g(\cdot)$ should be a monotonically decreasing function, since the error probability should be smaller if more energy is used. Suppose the energy-reliability tradeoff functions of AND-gates, XOR-gates and majority-gates are $\epsilon_{\text{and}}(\cdot)$, $\epsilon_{\text{xor}}(\cdot)$ and $\epsilon_{\text{maj}}(\cdot)$ respectively. Then, the failure probability of these three types of gates are $p_{\text{and}} = \epsilon_{\text{and}}(E_v)$, $p_{\text{xor}} = \epsilon_{\text{xor}}(E_v)$ and $p_{\text{maj}} = \epsilon_{\text{maj}}(E_v)$.

3.5.1 ENCODED-V: low-energy matrix-vector multiplications using dynamic voltage scaling

We modify the ENCODED-F technique in Section 3.4.1 with 'dynamic' voltage scaling to obtain arbitrarily small output error fraction. The gate model here is Model I. The original ENCODED-F technique has $\lceil L/(d_s - 1)\rceil$ stages, where in each stage, a noisy decoder of the utilized LDPC code is used to carry out one (noisy) iteration of PBF decoding. In the original ENCODED-F technique, we assumed that gate failure probability is constant (and equal for all gates) throughout the duration of the computation process. Here, we partition the entire ENCODED-F technique into two phases. In the first phase, we use constant supply energy, while in the second phase, we increase the supply energy as the computation proceeds, so that the gate failure probability decreases during the computation process, in order to achieve the required output error fraction with high probability.

For ease of presentation, we consider the case when $d_s = 2$, i.e., we only add $d_s - 1 = 1$ codeword to the N-bit storage at each stage. The extension to general d_s is straightforward. We partition the entire ENCODED-F so that there are $L - L_{\text{vs}}$ stages in the first phase and L_{vs} stages in the second phase, where L_{vs} is defined as

$$L_{\text{vs}} = \left\lceil \frac{\log \frac{1}{p_{\text{tar}}} + \log \alpha_0}{\log \frac{1}{1-\frac{1}{2}\theta}} \right\rceil, \tag{3.32}$$

where p_{tar} is the required final output error fraction. In the i-th stage of the last L_{vs} stages, we assume that the supply energy is increased to some value to ensure that

$$[d_c(1-R)+1]p_{\text{xor}}^{(i+1)} + p_{\text{maj}}^{(i+1)} + p_{\text{and}}^{(i+1)} \leq \frac{1}{4}\theta\alpha_0\left(1 - \frac{1}{2}\theta\right)^i. \tag{3.33}$$

We call this (dynamic) voltage-scaling scheme the ENCODED-V technique.

Theorem 3.5.1. *(Using dynamic voltage scaling for Problem 1) Using unreliable AND gates, majority gates and XOR gates defined from Gate Model I (D,ϵ) with maximum fan-in D and error probability p_{and}, p_{xor} and p_{maj}, and using a regular LDPC code that satisfies assumption (A.3), the binary matrix-vector multiplication $\mathbf{r}=\mathbf{s}\cdot\mathbf{A}$ can be computed using the ENCODED-F technique with dynamic voltage scaling, with per-bit energy consumption*

$$\begin{aligned}
&E_{per\text{-}bit} = \\
&\frac{L-L_{vs}}{K}\left[N\epsilon_{and}^{-1}\left(p_{and}\right)+N\epsilon_{maj}^{-1}\left(p_{maj}\right)+(N+P)\epsilon_{xor}^{-1}\left(p_{xor}\right)\right] \\
&+\frac{N}{K}\sum_{i=1}^{L_{vs}}\epsilon_{and}^{-1}\left(p_{and}^{(i)}\right)+\frac{N}{K}\sum_{i=1}^{L_{vs}}\epsilon_{maj}^{-1}\left(p_{maj}^{(i)}\right) \\
&+\frac{N+P}{K}\sum_{i=1}^{L_{vs}}\epsilon_{xor}^{-1}\left(p_{xor}^{(i)}\right),
\end{aligned} \tag{3.34}$$

where L_{vs}, which is a function of p_{tar}, is defined in (3.32). *Further, the output error fraction is below p_{tar} with probability at least $1-P_e^{blk}$, where the probability P_e^{blk} satisfies*

$$P_e^{blk} < 3(L-L_{vs})\exp\left(-\lambda^* N\right)+3\sum_{i=1}^{L_{vs}}\exp\left(-\widetilde{\lambda}^{(i+1)}N\right), \tag{3.35}$$

where

$$\begin{aligned}
&\lambda^* = D(2\lambda\|\lambda) = \left(2\log 2-1\right)\lambda+\mathcal{O}(\lambda^2), \\
&\widetilde{\lambda}^{(i+1)} = D(2\lambda^{(i+1)}\|\lambda^{(i+1)}) \\
&\qquad\quad = \left(2\log 2-1\right)\lambda^{(i+1)}+\mathcal{O}((\lambda^{(i+1)})^2), \\
&\lambda = \tfrac{\theta\alpha_0/2}{[d_c(1-R)+1]+2}, \\
&\lambda^{(i+1)} = \tfrac{\theta\alpha_0\left(1-\frac{1}{2}\theta\right)^i/4}{[d_c(1-R)+1]+2}.
\end{aligned} \tag{3.36}$$

Proof. See Appendix A.4. □

We consider three specific cases of energy-reliability tradeoff: exponential decay model $\epsilon_{\text{and}}(u)=\epsilon_{\text{xor}}(u)=\epsilon_{\text{maj}}(u)=\exp(-cu), c>0$, polynomial decay model $\epsilon_{\text{and}}(u)=\epsilon_{\text{xor}}(u)=\epsilon_{\text{maj}}(u)=(\frac{1}{u})^c, c>0$ or sub-exponential decay model $\epsilon_{\text{and}}(u)=\epsilon_{\text{xor}}(u)=\epsilon_{\text{maj}}(u)=\exp(-c\sqrt{u}), c>0$. We evaluate the total energy consumption per output bit under a specific choice of supply energy that ensures the condition (3.33).

Corollary 3.5.2. *Using a (d_v,d_c) regular LDPC code that satisfies assumption (A.3) (with parameters α_0 and θ) and has length $N>\frac{1}{\theta^*}\log\left(\frac{6L}{p_{tar}}\right)$, where*

$$\begin{aligned}
&\theta^* = \min\{\lambda^*, \\
&D\left(2\frac{\theta p_{tar}\left(1-\frac{1}{2}\theta\right)/4}{[d_c(1-R)+1]+2}\middle\|\frac{\theta p_{tar}\left(1-\frac{1}{2}\theta\right)/4}{[d_c(1-R)+1]+2}\right)\Bigg\},
\end{aligned} \tag{3.37}$$

and λ^* is defined in (3.36), *the ENCODED-V technique can achieve output bit error probability p_{tar} with total energy consumption pet bit $E_{per\text{-}bit}$: When the energy-reliability tradeoff function*

Table 3.1: This table shows the energy-reliability tradeoffs of different computing schemes under different gate error probability models.

	uncoded	ENCODED-T	ENCODED-V
$\epsilon = \exp(-cu)$	$\Omega\left(L \log \frac{L}{p_{\text{tar}}}\right)$	$\Theta\left(\frac{LN}{K} \log \frac{1}{p_{\text{tar}}}\right)$	$\mathcal{O}\left(\frac{N}{K} \max\{L, \log^2 \frac{1}{p_{\text{tar}}}\}\right)$
$\epsilon = (\frac{1}{u})^c$	$\Omega\left(L(\frac{L}{p_{\text{tar}}})^{\frac{1}{c}}\right)$	$\Theta\left(\frac{LN}{K}(\frac{1}{p_{\text{tar}}})^{\frac{1}{c}}\right)$	$\mathcal{O}\left(\frac{N}{K} \max\left\{L, \left(\frac{1}{p_{\text{tar}}}\right)^{\frac{1}{c}}\right\}\right)$
$\epsilon = \exp(-c\sqrt{u})$	$\Omega\left(L \log^2 \frac{L}{p_{\text{tar}}}\right)$	$\Theta\left(\frac{LN}{K} \log^2 \frac{1}{p_{\text{tar}}}\right)$	$\mathcal{O}\left(\frac{N}{K} \max\{L, \log^3 \frac{1}{p_{\text{tar}}}\}\right)$

$\epsilon_{and}(u) = \epsilon_{xor}(u) = \epsilon_{maj}(u) = (\frac{1}{u})^c, c > 0$, $E_{per\text{-}bit} = \mathcal{O}\left(\frac{N}{K} \max\left\{L, \left(\frac{1}{p_{tar}}\right)^{\frac{1}{c}}\right\}\right)$; *when the energy-reliability tradeoff function* $\epsilon_{and}(u) = \epsilon_{xor}(u) = \epsilon_{maj}(u) = \exp(-cu), c > 0$, $E_{per\text{-}bit} = \mathcal{O}\left(\frac{N}{K} \max\{L, \log^2 \frac{1}{p_{tar}}\}\right)$; *when the energy-reliability tradeoff function* $\epsilon_{and}(u) = \epsilon_{xor}(u) = \epsilon_{maj}(u) = \exp(-c\sqrt{u}), c > 0$, $E_{per\text{-}bit} = \mathcal{O}\left(\frac{N}{K} \max\{L, \log^3 \frac{1}{p_{tar}}\}\right)$.

Proof. See Appendix A.5. □

We use Table 3.1 to show the energy-reliability tradeoff of "uncoded" matrix multiplication, ENCODED-T and ENCODED-V. The resutls of ENCODED-T are from [278].

3.6 Simulation results

We use simulations to compare ENCODED and repetition-based schemes. In particular, we provide a comparison between ENCODED-F and a particular repetition-based scheme called "distributed voting scheme" (see [109] for details on the distributed voting scheme), that is designed for $p_{\text{maj}} > 0$. This method repeats not only the computation part, but also the majority voting part of the repetition-based circuit. The illustration of the distributed voting scheme is shown in Fig. 3.3. In this way, we can compare the (repetition-coding based) distributed voting scheme with ENCODED that both use noisy gates.

The performance comparison is shown in Fig. 3.4. In the distributed majority scheme, we use three-time repetition or four-time repetition. For ENCODED-F, we set $d_v = 4$, $d_c = 8$, $d_s = 8$, $K = 2000$, $L = 2100$, $N = 4000$. We set $p_{\text{and}} = 0.000125$, $p_{\text{maj}} = 0.0005$ and $p_{\text{xor}} = 0.001$. We set these error parameters because we assume that the error probability of each gate is proportional to its fan-in number (we use 2-input AND-gates, 4-input MAJ-gates and 8-input XOR gates). Note that the number of compute-and-correct stages in ENCODED-F should be $\left\lceil \frac{L}{d_s - 1} \right\rceil = 300$. In one compute-and-correct stage, we need N XOR-operations of fan-in $d_s = 8$ for binary addition, P XOR-operations of fan-in $d_c = 8$ for parity computation and N MAJ-operations of fan-in $d_v = 4$ for majority computation. In all 300 stages, we also need NL AND-operations of fan-in 2. Therefore the number of

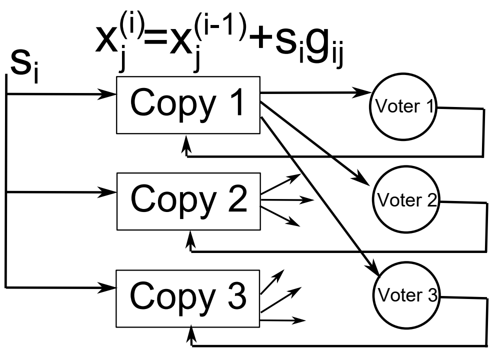


Figure 3.3: This is the illustration of the 3-time distributed voting scheme for computing an inner product $\mathbf{s} \cdot \mathbf{a}_j = s_1a_{1j} + s_2a_{2j} + \ldots s_La_{Lj}$, where s is the input to the matrix-vector multiplication $\mathbf{sA}$, and $\mathbf{a}_j$ is the j-th column of $\mathbf{A}$. The computation is divided into L stages. In the i-th stage, the distributed voting scheme computes $x_j^{(i)} = x_j^{(i-1)} + s_ig_{ij}$ for three times using three sets of AND-gates and XOR-gates, uses three noisy majority-gates to compute three copies of the majority votes. Then, the output of each majority value is sent to the corresponding copy for the computation in the next stage.

operations per output bit for ENCODED-F is

$$\mathscr{N}^{\text{ENC}}_{\text{XOR}-8,\text{per-bit}} = \frac{N+P}{K}\left\lceil \frac{L}{d_s-1} \right\rceil = \frac{3}{7}L, \tag{3.38}$$

$$\mathscr{N}^{\text{ENC}}_{\text{MAJ}-4,\text{per-bit}} = \frac{N}{K}\left\lceil \frac{L}{d_s-1} \right\rceil = \frac{2}{7}L, \tag{3.39}$$

$$\mathscr{N}^{\text{ENC}}_{\text{AND}-2,\text{per-bit}} = \frac{NL}{K} = 2L. \tag{3.40}$$

In the distributed majority voting scheme with repetition time t_m (t_m can be 3 or 4 when the majority gate with fan-in 4 is used), the number of operations per output bit is

$$\mathscr{N}^{\text{Rep}}_{\text{XOR}-8,\text{per-bit}} = t_m\left\lceil \frac{L}{d_s-1} \right\rceil = \frac{t_m}{7}L, \tag{3.41}$$

$$\mathscr{N}^{\text{Rep}}_{\text{MAJ}-\text{t}_\text{m},\text{per-bit}} = t_m\left\lceil \frac{L}{d_s-1} \right\rceil = \frac{t_m}{7}L, \tag{3.42}$$

$$\mathscr{N}^{\text{Rep}}_{\text{AND}-2,\text{per-bit}} = t_mL. \tag{3.43}$$

Therefore, when the repetition time t_m is 3 or 4, the number of operations per output bit for ENCODED-F is always smaller than the number of operations per output bit for the distributed majority voting scheme.

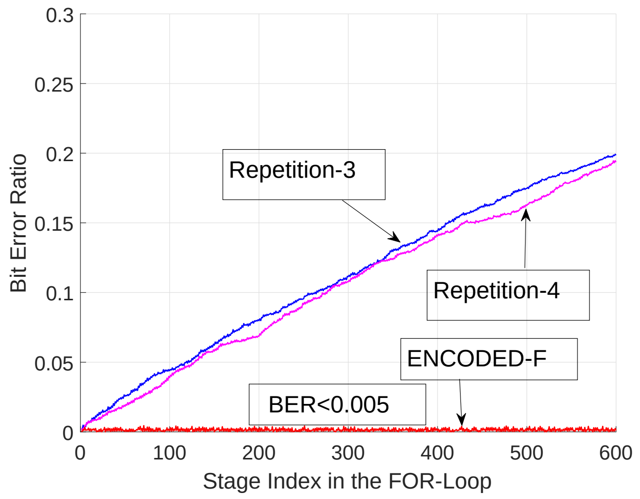


Figure 3.4: In this figure, a simulation result of ENCODED-F using a (4,8) regular LDPC with $d_s = 8$ is shown. The code length $N = 4000$, the size of the matrix-vector multiplication satisfies $L = 2100$ and $K = 2000$. A comparison with the distributed majority voting schemes with repetition time 3 and 4 is also shown. The gate error probabilities are set to $p_{\text{and}} = 0.000125$, $p_{\text{maj}} = 0.0005$ and $p_{\text{xor}} = 0.001$ in both ENCODED-F and the distributed majority voting scheme.

3.7 Conclusions and future directions

In this chapter, we presented the ENCODED technique that can compute reliable matrix-vector multiplication using entirely unreliable logic gates. The key idea that ENCODED relies on is to repeatedly suppress errors in computation process by, in a sense, encoding the computation matrix of the matrix-vector multiplication, instead of encoding inputs (as is done in traditional communication). Using ENCODED, both probabilistic errors and worst-case errors can be kept suppressed. Further, we used thorough numerical analysis to show that ENCODED outperform repetition-based strategies that are commonly used today. Inspired by voltage-scaling techniques commonly used to reduce power in circuit design, we also analyzed possible gains attainable using 'static' and 'dynamic' voltage scaling in conjunction with our ENCODED technique.

One limitation of the technique in this chapter is that it is limited to finite fields instead of real number coding. It turns out that the extension to real-number coding can indeed be made. One can refer to the two case studies in Chapter 9 and 10 respectively for real-number coded computing with either noisy and noiseless decoding. In Chapter 9, we also use LDPC-type coding techniques for error-correction over reals.

Chapter 4

Fundamental limit: quantifying distortion accumulation for measuring error accumulation in computing systems

4.1 Introduction

As we have mentioned in Section 1.2.3, when examining computational problems, a fundamental question to ask is "what is the correct measure of error accumulation". For example, in the computation with noisy or unreliable components, the inaccuracy can accumulate. In this chapter, we will see one possible way to measure and quantify error that we believe is fundamental. Specifically, we look at the problem of error accumulation in multi-stage linear computation problems, and we call the measure of accumulation of error the *distortion accumulation* phenomenon. Note that a similar problem has been studied in the field of *information dissipation*, and here, we present results on the dissipation of information in multi-stage computation problems due to successive quantizations. The dissipation of information usually cannot be quantified easily using classical information-theoretic tools that rely on the law of large numbers, because the dissipation of information is often due to finite-length of codewords and power constraints on the channel inputs. In many classical network information theory problems, such as relay networks, the dissipation of information is not observed because it can be suppressed by use of asymptotically infinite block length. Quantifying dissipation of information requires tools that go beyond those commonly used in classical information theory, e.g., cut-set techniques and the data processing inequality.

In this chapter, we show that in distributed lossy *computation*, information does dissipate. This chapter is a review of the papers [274, 279]. We first study the problem of lossily computing a weighted sum of Gaussian vectors (i.e., a matrix-vector multiplication) over a tree network at an arbitrarily determined sink node. We prove that distortion must accumulate, and hence information, if measured in the way of mean-square distortion, must dissipate, along the way from leaves to the sink node due to repeated lossy quantization of distributed data scattered in the network. In contrast with dissipation results in communication, this information loss, measured in mean-square distortion, happens even at infinite block length. Moreover, by quantifying *incremental distortion*, i.e., an incremental information loss on each link of the tree network, we derive an information-theoretic outer bound on the rate-distortion function that is tighter than classical cut-set bounds obtained for this problem in the work of Cuff, Su and El Gamal [60]. Using the same technique, we improve the classical outer bound on the sum rate of network consensus (all nodes compute the same matrix-vector multiplication) for tree networks from $\mathcal{O}\left(n\log_2\frac{1}{n^{3/2}D}\right)$ (see [240, Proposition 4]) to $\mathcal{O}\left(n\log_2\frac{1}{D}\right)$, where n is the number of nodes in the tree network and D is the required overall distortion. In Remark 6, we provide the intuition underlying the difference between our bound and the cut-set bound for lossy in-network computation. Note that although our definition of information loss (measured in terms of distortion accumulation) is different from that of [198], this definition provides a new perspective in this line of study.

In Section 4.3 and Section 4.4, we provide information-theoretic bounds on the rate-distortion function for matrix-vector multiplication in a tree network, where the function is computed at an arbitrarily predetermined sink node. For simplicity, we restrict our attention to independent Gaussian vectors. In Section 4.5, we extend our results to the problem of network consensus, in which all nodes compute the same matrix-vector multiplication result. In both cases, the difference between the inner and outer bounds is shown to approach zero in the high-resolution (i.e., zero distortion) limit. Note that in [60, Section V], the authors show a constant difference between their lower and the inner bounds in the Gaussian case. Using our improved outer bound, we can upper-bound the difference by $\mathcal{O}(D^{1/2})$, where D is the required distortion. Therefore, the inner bound and the outer bound match in the asymptotic zero-distortion limit. In the special case of a line network, we show that the rate-distortion function is very similar to the reverse water-filling result for parallel Gaussian sources [59, Theorem 10.3.3].

The achievable scheme obtained in this chapter is based on random Gaussian codebooks. The main difficulty here is to bound the overall distortion for random coding in matrix-vector multiplication computation by generalizing classical random coding to multi-stage computing problems. To generalize classical random coding on point-to-point channels, and compute the overall distortion, we quantify a non-trivial equivalence between random-coding-based estimates and MMSE estimates. Relying on the distortion accumulation result for MMSE estimates, we equivalently obtain the distortion accumulation result for Gaussian random codebooks, and hence obtain the overall distortion. This equivalence between random coding and MMSE is easy to obtain for point-to-point channels, but hard for network function computation, due to information loss about the exact source distribution after successive quantization. The essential technique is to bound this information loss using bounds on associated KL-divergences, and hence to show the equivalence between network computation and point-to-point communications. (See also Remark 8 for details on why our analysis is conceptually different from classical techniques such as Wyner-Ziv coding and why such new proof techniques are needed.)

4.1.1 Applications

Since the primary motivation for this chapter is to provide a thorough understanding of error accumulation in computing systems, the content is mostly theoretical. Therefore, we think it is useful to comment on some applications of the proof techniques developed in this chapter. Apart from data aggregation and network consensus, which are the two main problems in this chapter, the proof techniques on distortion accumulation are also useful for understanding the fundamental limits of many other problems when communication overhead dominates the overall distributed computing cost. One such example application is of current interest is "distributed data summarization" often done through the computation of histograms. A good example is distributed word counting: document data are stored distributedly in many workers organized in a data-center network, and the goal is to compute an overall histogram of the word counts. The general problem of data summarization aims to obtain the final summarization of distributed data by computing a linear combination of the summarizations of data at different workers. Another exciting application [176] is when data comes in a *streaming* fashion, and the storage minimization problem in a timeline network in [176] is related to the computation rate minimization problem in a line network in our framework.

4.1.2 Notation and preliminary results

Quantities that measure mean-square distortions are denoted by D or d with subscripts and superscripts. The calligraphic letter $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ is used to represent a tree graph with a node set $\mathcal{V} = \{v_i\}_{i=0}^n$ with cardinality $n+1$ and an edge set $\mathcal{E}$. In this chapter, an edge is always undirected[1]. The neighborhood $\mathcal{N}(v_i)$ of a node v_i is defined as all the nodes that are connected with v_i. A root node v_0 is specified for the tree graph. For an arbitrary node $v_i \neq v_0$, a unique parent of v_i on the path from v_i to v_0 can be determined, which is denoted as $v_{\mathrm{PN}(i)}$. The *children* of v_i are defined as the set of nodes $\{v_j \in \mathcal{V} \mid v_i = v_{PN(j)}\}$. The *descendants* of v_i are defined as the set of nodes that includes all nodes v_j that have v_i on the unique path from v_j to the root v_0. The set $\mathcal{S}_i$ is used to denote the set that is constituted by node v_i and all the descendants of v_i. As shown in Fig. 4.1, the set $\mathcal{S}$ is constituted by a node v_b and its descendants. Thus, in Fig. 4.1, $\mathcal{S} = \mathcal{S}_b$ and $v_a = v_{\mathrm{PN}(b)}$. When there is no ambiguity, we use $v_1, v_2, \ldots v_d$ to denote the children of a particular node v_b.

First, we state the orthogonality principle and the statisticians' Pythagoras theorem, which we will use frequently in this chapter.

Lemma 4.1.1. *(Pythagoras theorem, [265, Theorem 9.4], [220, Section 8.1]) For a random (vector) variable X such that $\mathbb{E}[X^\top X] < \infty$ and a σ-algebra $\mathcal{G}$, the conditional expectation $\mathbb{E}[X|\mathcal{G}]$ is a version of the orthogonal projection of X onto the probability space $\mathcal{L}^2(\Omega, \mathcal{G}, \mathbf{P})$: for all $\mathcal{G}$-measurable (vector) functions Y, it holds that $Y \perp (X - \mathbb{E}[X|\mathcal{G}])$, or equivalently*

$$\mathbb{E}\left[Y\left(X - \mathbb{E}[X|\mathcal{G}]\right)^\top\right] = 0. \tag{4.1}$$

[1]Although we consider an undirected tree graph, we specify a unique root node, which makes the subsequent definitions on descendants and children valid.

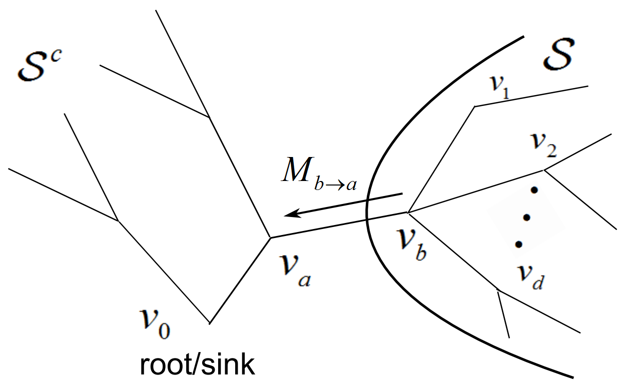


Figure 4.1: This is an illustration of matrix-vector multiplication considered in this chapter. The goal is to compute a weighted sum of distributed Gaussian vectors (i.e., matrix-vector multiplication) over a tree-network. The notation $M_{b\to a}$ denotes the message, or the set of bits, transmitted from v_b to v_a. The set $\mathcal{S}$ in this figure can also be written as $\mathcal{S}_b$, which denotes the set that contains v_b and all its descendants in the network.

Second, we provide a lemma that describes the relationship between the Kullback-Leibler divergence and the mean-square error under Gaussian smoothing.

Lemma 4.1.2. *([268][206, Lemma 3.4.2]) Let $\mathbf{x}$ and $\mathbf{y}$ be a pair of N-dimensional real-valued random vectors, and let $\mathbf{z} \sim \mathcal{N}(\mathbf{0}_N, \mathbf{I}_N)$ be independent of $(\mathbf{x}, \mathbf{y})$. Then, for any $t > 0$,*

$$D\left(P_{\mathbf{x}+\sqrt{t}\mathbf{z}}||P_{\mathbf{y}+\sqrt{t}\mathbf{z}}\right) \leq \frac{1}{2t}\mathbb{E}\left[\|\mathbf{x}-\mathbf{y}\|_2^2\right]. \tag{4.2}$$

Proof. See the paper [279] for a detailed proof. □

4.2 System model and problem formulation

We consider a matrix-vector multiplication problem in a tree network $\mathcal{T} = (\mathcal{V}, \mathcal{E})$. Suppose each node $v_i \in \mathcal{V}$ has an independent random vector $\mathbf{x}_i \sim \mathcal{N}(\mathbf{0}_N, \mathbf{I}_N)$. We assume that only bits can be sent on the network links. The objective is to obtain a weighted sum $\mathbf{y} = \sum\limits_{i=1}^{n} w_i\mathbf{x}_i$, i.e. matrix-vector multiplication result, at the pre-assigned sink node v_0. In Section 4.5, we will also consider an extension of the problem where the same matrix-vector multiplication is computed at all nodes.

There are T time slots. In each time slot, only one node transmits along only one edge. At each time slot t, the transmitting node $v(t)$ computes a mapping f_t (whose arguments are to be made precise below) and transmits an encoded version $g_t(f_t)$ to one of its neighbors through the edge $e(t)$. Each encoding mapping g_t outputs a binary sequence of a finite length. The arguments of f_t may consist of all the information available at the transmitting node $v(t)$ up to time t, including its observation $\mathbf{x}_{v(t)}$, randomly generated data, and information obtained from its neighborhood up to time t. Note that the total number of time slots T can be greater than number of vertices n in general, i.e., nodes may be allowed to transmit multiple times. For an arbitrary link $v_i \to v_j$, define $M_{i\to j}$ as all the bits transmitted on the link $v_i \to v_j$ (see Fig. 4.1). Denote by $R_{i\to j}$ the number of bits in $M_{i\to j}$ normalized by N. Note that $R_{i\to j}$ is the (normalized) total number of bits transmitted possibly over multiple time slots to node v_j. Also note that $R_{i\to j} > 0$ only if v_i and v_j are connected. We consider the minization of the *sum rate* defined as the following.

$$R = \frac{1}{N}\sum_{i=1}^{n}(NR_{i\to \mathrm{PN}(i)} + NR_{\mathrm{PN}(i)\to i}) = \sum_{i=1}^{n}(R_{i\to \mathrm{PN}(i)} + R_{\mathrm{PN}(i)\to i}). \tag{4.3}$$

We only consider oblivious and fixed distributed computation schemes which do not change with inputs. A scheme must be feasible, i.e., all arguments of f_t should be available in $v(t)$ before time t.

Since the goal is to compute $\mathbf{y} = \sum\limits_{i=1}^{n} w_i\mathbf{x}_i$ at the sink node v_0, without loss of generality, we assume $v(T) = v_0$ and the output of the mapping $f(T)$ computed at $v(T)$ is the final estimate $\widehat{\mathbf{y}}$. Denote by D the overall (normalized) mean-square distortion

$$D = \frac{1}{N}\mathbb{E}\left[\|\mathbf{y} - \widehat{\mathbf{y}}\|_2^2\right]. \tag{4.4}$$

The objective is to compute the minimum value of the sum rate R (defined in (4.3)) such that the overall distortion is smaller than D^{tar}.

$$\begin{aligned} \min \quad & R, \\ \text{s.t. } & D \le D^{\text{tar}}. \end{aligned} \tag{4.5}$$

In what follows, we define some quantities associated with the "incremental distortion" that we mentioned in Section 4.1. For an arbitrary set $\mathcal{S} \subset \mathcal{V}$, define $\mathbf{y}_{\mathcal{S}} = \sum\limits_{v_j\in\mathcal{S}} w_j\mathbf{x}_j$ as the partial sum in $\mathcal{S}$. We use $\sigma_{\mathcal{S}}^2 = \sum\limits_{v_j\in\mathcal{S}} w_j^2$ to denote the variance of each entry of $\mathbf{y}_{\mathcal{S}}$. Suppose at the final time slot T, all the available information (observations of random variables) at a node $v_i \in \mathcal{V}$ is I_i. Denote by $\widehat{\mathbf{y}}_{\mathcal{S},i}^{\text{mmse}}$ the MMSE estimate of $\mathbf{y}_{\mathcal{S}}$ at any node v_i, given the information I_i, which can be written as

$$\widehat{\mathbf{y}}_{\mathcal{S},i}^{\text{mmse}} = \mathbb{E}\left[\mathbf{y}_{\mathcal{S}}|I_i\right]. \tag{4.6}$$

For an arbitrary (non-sink) node v_i and its parent node $v_{\mathrm{PN}(i)}$, denote by D_i^{Tx} and D_i^{Rx} the MMSE distortions of estimating $\mathbf{y}_{\mathcal{S}_i}$, respectively at v_i and $v_{\mathrm{PN}(i)}$, where, recall, $\mathcal{S}_i$ denotes the set of descendants of node v_i (including itself). The information about $\mathbf{y}_{\mathcal{S}_i}$ should be transmitted from v_i to its parent $v_{\mathrm{PN}(i)}$. Therefore, the superscript $^{\text{Tx}}$ means that the distortion is defined for the transmitting node v_i, and the superscript $^{\text{Rx}}$ means the receiving node $v_{\mathrm{PN}(i)}$. Define D_i^{Inc} to be the mean-square difference between the two estimates $\widehat{\mathbf{y}}_{\mathcal{S}_i,i}^{\mathrm{mmse}}$ and $\widehat{\mathbf{y}}_{\mathcal{S}_i,\mathrm{PN}(i)}^{\mathrm{mmse}}$. Thus,

$$D_i^{\mathrm{Tx}} = \frac{1}{N}\mathbb{E}\left[\left\|\mathbf{y}_{\mathcal{S}_i} - \widehat{\mathbf{y}}_{\mathcal{S}_i,i}^{\mathrm{mmse}}\right\|_2^2\right], \tag{4.7}$$

$$D_i^{\mathrm{Rx}} = \frac{1}{N}\mathbb{E}\left[\left\|\mathbf{y}_{\mathcal{S}_i} - \widehat{\mathbf{y}}_{\mathcal{S}_i,\mathrm{PN}(i)}^{\mathrm{mmse}}\right\|_2^2\right], \tag{4.8}$$

$$D_i^{\mathrm{Inc}} = \frac{1}{N}\mathbb{E}\left[\left\|\widehat{\mathbf{y}}_{\mathcal{S}_i,\mathrm{PN}(i)}^{\mathrm{mmse}} - \widehat{\mathbf{y}}_{\mathcal{S}_i,i}^{\mathrm{mmse}}\right\|_2^2\right]. \tag{4.9}$$

Denote the MMSE distortion in estimating $\mathbf{y} = \sum_{i=1}^{n} w_i \mathbf{x}_i$ at v_0 by D_0^{mmse}. Because for the same distributed computation scheme, the overall distortion D cannot be less than D_0^{mmse}, the overall distortion with MMSE estimate at the sink v_0,

$$D \geq D_0^{\mathrm{mmse}}. \tag{4.10}$$

In Section 4.3.1, we will show that $D_i^{\mathrm{Inc}} = D_i^{\mathrm{Rx}} - D_i^{\mathrm{Tx}}$ (for all feasible distributed computation schemes) and the overall MMSE distortion D_0^{mmse} can be written as the summation of D_i^{Inc} on all links. Therefore, we call D_i^{Inc} the incremental distortion.

4.3 Main results: outer bounds based on incremental distortion

4.3.1 Distortion accumulation

Our first result shows that the overall MMSE distortion can be written as the summation of the distortion on all the tree links. It asserts that the distortion for in-network computing must accumulate along the way from all the leaves to the sink node.

Theorem 4.3.1 (Distortion Accumulation). *For any feasible distributed computation scheme (see the model of Section 4.2) and for each node $v_i \in \mathcal{V} \setminus \{v_0\}$, the incremental distortion D_i^{Inc} and the MMSE distortions D_i^{Tx} and D_i^{Rx} satisfy*

$$D_i^{Rx} = D_i^{Tx} + D_i^{Inc}. \tag{4.11}$$

Thus, we also have

$$D_i^{Tx} = \sum_{v_j \in \mathcal{S}_i \setminus \{v_i\}} D_j^{Inc}, \tag{4.12}$$

$$D_0^{mmse} = \sum_{i=1}^{n} D_i^{Inc}. \tag{4.13}$$

Proof. See Appendix B.1.1. □

Remark 5. In some of the proofs in this chapter, we adopt an 'induction method in the tree network', which we often briefly refer to as *induction in the tree*. The idea is that, to prove that some property P holds for each node $v_i \in \mathcal{V}$, firstly, we prove that P holds at all leaves. Secondly, we prove that, for an arbitrary node v_b, if P holds at v_b, then P also holds at its parent-node v_a. It is obvious that these two arguments lead to the conclusion that P holds for all nodes in the tree network.

4.3.2 Rate-distortion outer bound

Our second result provides an outer bound on the rate-distortion function for matrix-vector multiplication over a tree network using incremental distortions.

Theorem 4.3.2 (Incremental-Distortion-Based Outer Bound). *For the model of Section 4.2, given a feasible distributed computation scheme, the sum rate is lower-bounded by*

$$\begin{aligned} R &\geq \frac{1}{2}\sum_{i=1}^{n}\left[\log_2\frac{\sigma^2_{\mathcal{S}_i}}{D_i^{Inc}} - \frac{D_i^{Tx}}{2w_i^2} - \frac{\log_2 e}{2\sigma^2_{\mathcal{S}_i}}\sqrt{2D_i^{Tx}\left(4\sigma^2_{\mathcal{S}_i} + D_i^{Tx}\right)}\right] \\ &= \frac{1}{2}\sum_{i=1}^{n}\left[\log_2\frac{\sigma^2_{\mathcal{S}_i}}{D_i^{Rx} - D_i^{Tx}} - \mathcal{O}\left((D_i^{Tx})^{1/2}\right)\right], \end{aligned} \tag{4.14}$$

where w_i is the weight of the observation $\mathbf{x}_i$, $\mathcal{S}_i$ is the node set that contains node v_i and its descendants, $\sigma^2_{\mathcal{S}_i}$ is the variance of each entry of the partial sum $\mathbf{y}_{\mathcal{S}_i} = \sum_{v_j \in \mathcal{S}_i} w_j \mathbf{x}_j$, D_i^{Tx} and D_i^{Inc} are the MMSE distortion and the incremental distortion at the node v_i, which are respectively defined in (4.7) *and* (4.9). *By optimizing over the incremental distortions D_i^{Inc}, one obtains the following scheme-independent bound stated in an optimization form*

$$\begin{aligned} &\min_{D_i^{Inc}, 1\leq i\leq n} \frac{1}{2}\sum_{i=1}^{n}\left[\log_2\frac{\sigma^2_{\mathcal{S}_i}}{D_i^{Inc}} - \frac{D_i^{Tx}}{2w_i^2} - \frac{\log_2 e}{2\sigma^2_{\mathcal{S}_i}}\sqrt{2D_i^{Tx}\left(4\sigma^2_{\mathcal{S}_i} + D_i^{Tx}\right)}\right], \\ &\quad s.t. \begin{cases} D_i^{Tx} = \sum_{v_j \in \mathcal{S}_i \setminus \{v_i\}} D_j^{Inc}, \forall i \neq 0, \\ \sum_{i=1}^{n} D_i^{Inc} = D_0^{mmse} \leq D. \end{cases} \end{aligned} \tag{4.15}$$

Define the function $\psi_i(\cdot)$ as

$$\psi_i(x) = \frac{x}{2w_i^2} + \frac{\log_2 e}{2\sigma^2_{\mathcal{S}_i}}\sqrt{2x\left(4\sigma^2_{\mathcal{S}_i} + x\right)}. \tag{4.16}$$

Then, a lower bound on R can be obtained from the optimization in (4.15)*:*

$$R \geq \frac{1}{2}\log_2\frac{\prod_{i=1}^{n}\sigma^2_{\mathcal{S}_i}}{(D/n)^n} - \frac{1}{2}\sum_{i=1}^{n}\psi_i(D), \tag{4.17}$$

which means that in the limit of small distortion D, the optimization problem (4.15) *provides the following lower bound in order sense*

$$R \geq \frac{1}{2}\log_2 \frac{\prod_{i=1}^{n} \sigma_{\mathcal{S}_i}^2}{(D/n)^n} - n\mathcal{O}(D^{1/2}). \tag{4.18}$$

Proof Sketch: The complete proof is in Appendix B.1.2. The first step is to prove, on an arbitrary link $v_b \to v_a$ towards the root (see Fig. 4.1), $NR_{b\to a} \geq h(\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}}) - \frac{N}{2}\log_2 2\pi e D_b^{\text{Inc}}$, where $h(\cdot)$ denotes differential entropy, and hence the rate $R_{b\to a}$ is related to the incremental distortion D_b^{Inc}.

Then, we prove that $h(\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}}) > h(\mathbf{y}_{\mathcal{S}}) - \mathcal{O}\left(N(D_b^{\text{Tx}})^{1/2}\right)$, using inequality (4.2). Thus, using $h(\mathbf{y}_{\mathcal{S}}) = \frac{N}{2}\log 2\pi e \sigma_{\mathcal{S}_b}^2$ (note that $\mathcal{S}$ and $\mathcal{S}_b$ here denote the same set), we get $R_{b\to a} \geq \frac{1}{2}\log_2 \frac{\sigma_{\mathcal{S}_b}^2}{D_b^{\text{Inc}}} - \mathcal{O}\left((D_b^{\text{Tx}})^{1/2}\right)$. Inequality (4.14) can be obtained by summing over all links towards the root. The optimization form obtained in (4.15) only requires the minimization of the scheme-dependent bound over the choices of D_i^{Inc}. The proof of the last inequality (4.17) and its order-sense form (4.18) can be obtained by lower-bounding the optimization problem (4.15). □

This outer bound is obtained when all incremental distortions are equal, which is very similar to the reverse water-filling solution for the parallel Gaussian lossy source coding problem [59, Theorem 10.3.3] in the limit of large rate (zero distortion). We will prove that this rate (in the small distortion regime) is also achievable using Gaussian random codebooks (see Section 4.4). To achieve the optimal sum rate, the rate on the link $v_i \to v_{\text{PN}(i)}$ should be approximately equal to $\frac{1}{2}\log_2 \frac{\sigma_{\mathcal{S}_i}^2}{D/n}$, where $\sigma_{\mathcal{S}_i}^2$ is the variance of each entry of the partial sum $\mathbf{y}_{\mathcal{S}_i}$.

4.3.3 Comparison with the cut-set bound

Using the classical cut-set bound technique [240, Thm. 1], we can obtain another bound different from the one in Theorem 4.3.2. This bound is in the same mathematical form as the sum rate expression in [60, Sec. V-A.3].

Theorem 4.3.3 (Cut-Set Outer Bound)**.** *For the model of Section 4.2, the sum rate is lower-bounded by*

$$R \geq \frac{1}{2}\sum_{i=1}^{n} \log_2 \frac{\sigma_{\mathcal{S}_i}^2}{D_i^{\text{Rx}}}. \tag{4.19}$$

Proof. See Appendix B.1.3. □

Denote by R_1 the outer bound obtained by the classical cut-set bound (Theorem 4.3.3) and by R_2 the outer bound obtained by Theorem 4.3.2. From (4.14) and (4.19)

$$\Delta_R := R_2 - R_1 = \frac{1}{2}\sum_{i=1}^{n}\left[\log_2 \frac{D_i^{\text{Rx}}}{D_i^{\text{Rx}} - D_i^{\text{Tx}}} - \mathcal{O}\left((D_i^{\text{Tx}})^{1/2}\right)\right]. \tag{4.20}$$

In order to illustrate the improvement on the outer bound R_2, we consider the case when $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ is a line network, connected as $v_0 \leftrightarrow v_1 \leftrightarrow \ldots \leftrightarrow v_n$. Then,

$$\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S}_{i-1},i-1} \overset{(a)}{=} \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S}_{i-1},\text{PN}(i)} \overset{(b)}{=} \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S}_{i},\text{PN}(i)} + w_{i-1}\mathbf{x}_{i-1}, \tag{4.21}$$

where (a) holds because v_{i-1} is the parent-node of v_i, and (b) follows from $\mathbf{y}_{\mathcal{S}_{i-1}} = \mathbf{y}_{\mathcal{S}_i} + w_{i-1}\mathbf{x}_{i-1}$. Therefore, $\mathbf{y}_{\mathcal{S}_{i-1}} - \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S}_{i-1},i-1} = \mathbf{y}_{\mathcal{S}_i} - \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S}_i,\text{PN}(i)}$. Using (4.7), (4.8), we obtain $D^{\text{Tx}}_{i-1} = D^{\text{Rx}}_i$. Thus, (4.20) changes to

$$\Delta_R = \frac{1}{2}\sum_{i=1}^{n}\left[\log_2 \frac{D^{\text{Tx}}_{i-1}}{D^{\text{Tx}}_{i-1} - D^{\text{Tx}}_i} - \mathcal{O}\left((D^{\text{Tx}}_i)^{1/2}\right)\right], \tag{4.22}$$

where $0 = D^{\text{Tx}}_n < D^{\text{Tx}}_{n-1} < \ldots < D^{\text{Tx}}_1 < D^{\text{mmse}}_0 \leq D$.

Then, we consider a typical choice of D^{Tx}_i, which minimizes the rate outer bound. In (4.18), we can show that, when D is required to be small enough, the way to minimize the RHS of (4.14) is to make $D^{\text{Rx}}_i - D^{\text{Tx}}_i$ to be a constant for all i. This strategy yields a lower bound on the minimum possible rate. In the case of a line network, this strategy becomes $D^{\text{Tx}}_i = \frac{n-i}{n} D^{\text{mmse}}_0, \forall i$. Then

$$\Delta_R = \sum_{i=1}^{n}\left[\frac{\log_2(n-i+1)}{2} - \mathcal{O}\left((D^{\text{Tx}}_i)^{1/2}\right)\right] \approx \frac{1}{2}\log_2(n!) = \Theta(n\log_2 n), \tag{4.23}$$

when the overall distortion D is small, i.e., the gap between the two bounds can be arbitrarily large.

Remark 6. Here, we point out the intuition underlying the difference between the proofs of the incremental-distortion-based bound (Theorem 4.3.2) and the cut-set bound (Theorem 4.3.3). The classical proofs of cut-set bounds for lossy computation often rely on the following key steps:

$$\begin{aligned} &\text{Rate} \geq I\,(\text{Computed Result; True Result}) \\ &\quad \geq h(\text{True Result}) - h(\text{True Result}|\text{Computed Result}), \end{aligned} \tag{4.24}$$

where $h(\text{True Result}|\text{Computed Result})$ can be upper-bounded by a function of overall distortion and the expression $h(\text{True Result})$ can be obtained explicitly. However, the proof of the incremental-distortion-based bound is based on the following key steps (see Appendix B.1.2):

$$\begin{aligned} &\text{Rate on Link } e = (v_1, v_2) \\ &\geq I\,(\text{Computed Result 1; Computed Result 2}) \\ &\geq h(\text{Computed Result 1}) - h(\text{Computed Result 1}|\text{Computed Result 2}), \end{aligned} \tag{4.25}$$

where "Computed Result 1" denotes the MMSE estimate at the parent-node v_1 on link $e = (v_1, v_2)$ and "Computed Result 2" denotes the MMSE estimate at the child-node v_2 on link e. The term $h(\text{Computed Result 1}|\text{Computed Result 2})$ leads to a function of incremental distortion between two estimates, which yields a tighter bound than cut-set bounds for lossy in-network computing. However, the distribution of "Computed Result 1", the MMSE estimate, is unknown, and hence h(Computed Result 1) cannot be obtained directly. To solve this problem, we lower-bound h(Computed Result 1) by upper-bounding the difference between h(Computed Result 1) and h(True Result), using the inequality in Lemma 4.1.2.

4.4 Main results: achievable rates with random Gaussian codebooks

In this section, we use random Gaussian codebooks to give an incremental-distortion based sum rate inner bound. The main achievable result in this chapter is as follows.

Theorem 4.4.1 (Inner Bound). *Using random Gaussian codebooks, we can find a distributed computation scheme, such that the sum rate R is upper-bounded by*

$$R \leq \frac{1}{2}\sum_{i=1}^{n} \log_2 \frac{\sigma_{\mathcal{S}_i}^2}{d_i} + n\delta_N, \tag{4.26}$$

where $\lim_{N\to\infty} \delta_N = 0$ is a parameter defined in (4.39), *and d_i's are tunable distortion parameters, and $\sigma_{\mathcal{S}}^2 = \sum\limits_{v_j \in \mathcal{S}} w_j^2$. Further, the overall distortion D satisfies*

$$D \leq \sum_{i=1}^{n} d_i + \epsilon_N, \tag{4.27}$$

where $\lim_{N\to\infty} \epsilon_N = 0$ is a parameter defined in (4.55)[2]. *The limit sum rate $\lim_{N\to\infty} R$ exists, and can be upper-bounded by*

$$\lim_{N\to\infty} R \leq \frac{1}{2}\log_2 \frac{\prod\limits_{i=1}^{n} \sigma_{\mathcal{S}_i}^2}{(D/n)^n}. \tag{4.28}$$

Proof. See Section 4.4.2. □

We rely on typicality-based arguments to prove the inner bound. Therefore, before we elaborate on the main distributed computation scheme in Section 4.4.2, we first review some notation and techniques on typicality.

[2]The parameter δ_N is used for providing a slight excess rate of the rate defined by mutual information in (4.39), and the parameter ϵ_N upper-bounds the deviation of the overall sum distortion D from the summation of the tunable distortion parameters $\sum\limits_{i=1}^{n} d_i$. Note that here N denotes the code length of the random Gaussian codebooks.

4.4.1 Notation on typicality-based coding

We first define some random variables, the pdfs of which we will use in the distributed computation scheme. (We will clarify the absolute continuity and hence existence of densities with respect to the appropriate Lebesgue measure of the various random objects used in our proofs.) At each node v_i, we define an *estimate random variable* U_i^{TC} and a *description random variable* V_i^{TC}. The superscript $^{\text{TC}}$ represents the Gaussian test channel, which we will use to define these scalar random variables. Denote the variance of U_i^{TC} by $\widehat{\sigma}_i^2$. The estimate random variables U_i^{TC}'s are defined from the leaves to the root v_0 in the tree. For an arbitrary leaf v_l, define

$$U_l^{\text{TC}} = w_l X_l, \tag{4.29}$$

where $X_l \sim \mathcal{N}(0,1)$ is a scalar random variable, and w_l is the weight at node v_l in the weighted sum $\mathbf{y} = \sum\limits_{i=1}^{n} w_i \mathbf{x}_i$. For non-leaf nodes, without loss of generality, we use v_1, v_2, $\ldots v_d$ to denote the children of an arbitrary node v_b (see Fig. 4.1). Suppose the description random variables $\{V_i^{\text{TC}}\}_{i=1}^{d}$ at the children of node v_b have been defined (the formal definitions of the description random variables are provided later in equation (4.31)). Then, define the estimate random variable for the non-leaf node v_b as

$$U_b^{\text{TC}} = \sum_{k=1}^{d} V_k^{\text{TC}} + w_b X_b, \tag{4.30}$$

where $X_b \sim \mathcal{N}(0,1)$ is a scalar random variable, and w_b is the weight at v_b. At each node v_i, the description random variable V_i^{TC} is now defined based on the estimate random variable using a Gaussian test channel

$$U_i^{\text{TC}} = V_i^{\text{TC}} + Z_i, \tag{4.31}$$

where $Z_i \sim \mathcal{N}(0, d_i)$ is independent of V_i^{TC} and d_i is a variable that will be chosen later. From the definition of Gaussian test channels, $\text{var}[V_i^{\text{TC}}] = \widehat{\sigma}_i^2 - d_i$. Readers are referred to Appendix B.2.1 for details on the definition of Gaussian test channels. Then, using (4.31), we have that

$$\widehat{\sigma}_b^2 = \sum_{k=1}^{d} \text{var}[V_i^{\text{TC}}] + w_b^2 = \sum_{k=1}^{d} (\widehat{\sigma}_k^2 - d_k) + w_b^2. \tag{4.32}$$

Note that the estimate random variables and the description random variables are both defined from leaves to the root. However, we have different definitions of the estimate random variables for leaves and non-leaf nodes ((4.29) and (4.30)) but the same definition of description random variables. Note that the Gaussian test channel (4.31) and the definitions in (4.29) and (4.30) involve linear transformations. Therefore, all estimate random variables U_i^{TC}'s and description random variables V_i^{TC}'s are scalar Gaussian random variables with zero mean. We will not directly use the random variables U_i^{TC} and V_i^{TC} in the achievability proof (because they are scalars and cannot be directly used for coding). However, we use the pdfs of these random variables. We use $\phi_{U_i^{\text{TC}}}$ and

$\phi_{V_i^{\rm TC}}$ to denote the pdfs of $U_i^{\rm TC}$ and $V_i^{\rm TC}$. We also use joint pdfs, where the meanings are always clear from the context. Note that the variance of $U_i^{\rm TC}$ and $V_i^{\rm TC}$ are tunable, since the parameter d_i, which is related to the variance of the added Gaussian noise Z_i, is a tuning parameter.

Remark 7. In fact, the way in which we define the description random variables and estimate random variables in Section 4.4.1 essentially implies the basic idea of our distributed computation scheme. Although we consider block computation in the entire chapter, we can view these description random variables and estimate random variables as the 'typical' intermediate results during the computation. In particular, the estimate random variable $U_i^{\rm TC}$ represents the typical properties of the estimate $\widehat{\mathbf{s}}_i$ of the partial sum $\mathbf{y}_{\mathcal{S}_i}$ at the node v_i (by representing the typical properties, we mean the typical sets that the estimate $\widehat{\mathbf{s}}_i$ belongs to are defined based on the distributions of the estimate random variable $U_i^{\rm TC}$), while the description random variable $V_i^{\rm TC}$ represents the typical properties of the descriptions $\widehat{\mathbf{r}}_i$. Note that the messages to be further transmitted from the node v_i to its parent node is the description sequence $\widehat{\mathbf{r}}_i$. The estimate $U_0^{\rm TC}$ represents the properties of the estimate of Y at the sink v_0. Based on this intuition, we can provide an intuitive explanation of the formula in Theorem 4.4.1: suppose $U_i^{\rm TC}$ and $V_i^{\rm TC}$ are length-N vectors (this is of course technically incorrect, and we only try to provide some intuition on Theorem 4.4.1 here), then, since $U_i^{\rm TC}$ and $V_i^{\rm TC}$ are all Gaussian, it can be proved that $V_i^{\rm TC}$ is just the MMSE estimate $\widehat{\mathbf{y}}^{\rm mmse}_{\mathcal{S}_i,{\rm PN}(i)} = \mathbb{E}\left[\mathbf{y}_{\mathcal{S}_i}|I_{{\rm PN}(i)}\right] = \mathbb{E}\left[\mathbf{y}_{\mathcal{S}_i}|V_i^{\rm TC}\right]$ of the required partial sum $\mathbf{y}_{\mathcal{S}_i}$ at node $v_{{\rm PN}(i)}$, the parent node of v_i. Then, we can apply the distortion accumulation result ((4.13) in Theorem 4.3.1) to $U_i^{\rm TC}$ and $V_i^{\rm TC}$, and obtain $D = \sum\limits_{i=1}^{n} d_i$, since $d_i = \mathbb{E}[(U_i^{\rm TC})^2 - (V_i^{\rm TC})^2]$ is the counterpart of the incremental distortion $D_i^{\rm Inc}$. In Section 4.4.2, we will formalize this intuitive argument using Gaussian random codes.

Denote by q_U, q_V and $q_{U,V}$ the N-fold product distribution of the scalar distributions $\phi_{U_i^{\rm TC}}$, $\phi_{V_i^{\rm TC}}$ and $\phi_{U_i^{\rm TC},V_i^{\rm TC}}$. Denote by $\mathcal{T}_{U,\varepsilon}^N$ and $\mathcal{T}_{V,\varepsilon}^N$ the two sets of N-length sequences s^N and r^N that are respectively typical with respect to $\phi_U^{\rm TC}$ and $\phi_V^{\rm TC}$. Denote by $\mathcal{J}_\varepsilon^{2N}$ the set of all $2N$-length sequences $\left(s^N, r^N\right)$ that are jointly typical with respect to $\phi_{U^{\rm TC},V^{\rm TC}}$. Denote by $\mathcal{T}_{V,\varepsilon}^N(s^N)$ the set of sequences r^N that are jointly typical with a particular typical sequence s^N. The formal definitions of these typical sets are provided in the following equations (note that we will use a general definition of typical sets from [119], and we will show that the definitions below are special cases of the general definition):

$$\mathcal{T}_{U,\varepsilon}^N = \left\{ s^N : \left| -\frac{1}{N}\log q_U(s^N) - h(U_i^{\rm TC}) \right| < \varepsilon_N, \left| \frac{1}{N}||s^N||_2^2 - \hat{\sigma}_i^2 \right| < \varepsilon_N \right\}, \tag{4.33}$$

$$\mathcal{T}_{V,\varepsilon}^N = \left\{ r^N : \left| -\frac{1}{N}\log q_V(r^N) - h(V_i^{\rm TC}) \right| < \varepsilon_N, \left| \frac{1}{N}||r^N||_2^2 - (\hat{\sigma}_i^2 - d_i) \right| < \varepsilon_N \right\}, \tag{4.34}$$

$$\mathcal{J}_\epsilon^{2N} = \left\{ (s^N, r^N) : s^N \in \mathcal{T}_{U,\varepsilon}^N, r^N \in \mathcal{T}_{V,\varepsilon}^N, \left| -\frac{1}{N} \log q_{U,V}(s^N, r^N) - h(U_i^{\mathrm{TC}}, V_i^{\mathrm{TC}}) \right| < \varepsilon_N, \right.$$
$$\left. \left| \frac{1}{N} ||s^N - r^N||_2^2 - d_i \right| < \varepsilon_N \right\}, \tag{4.35}$$

$$\mathcal{T}_{V,\varepsilon}^N(s^N) = \left\{ r^N : (s^N, r^N) \in \mathcal{J}_\epsilon^{2N} \right\}. \tag{4.36}$$

4.4.2 Algorithm: applying Gaussian codes in function computing

The illustrative explanation in Remark 7 relies on Gaussian test channels, which is a heuristic to provide insights into the design of the achievability strategy. In this part, we rigorously prove the achievability using explicit random Gaussian codebooks.

Note that all computations are block computations. According to the system model, each node v_i has a random vector $\mathbf{x}_i$, where each coordinate is generated by $\mathcal{N}(0,1)$. The sink v_0 has the goal to compute the weighted sum $\mathbf{y} = \sum_{i=1}^{n} w_i \mathbf{x}_i$, i.e., a matrix-vector multiplication. Recall that $\mathbf{y}_\mathcal{S} = \sum_{v_j \in \mathcal{S}} w_j \mathbf{x}_j$ and $\sigma_\mathcal{S}^2 = \sum_{v_j \in \mathcal{S}} w_j^2$.

Before the computation starts, each node v_i generates a codebook[3] $\mathcal{C}_i = \{\mathbf{c}_i(w) : w \in \{0, 1, \dots 2^{NR_i}\}\}$, where each codeword is generated i.i.d. according to distribution $p_{V_i^{\mathrm{TC}}}$. The rate is chosen such that

$$R_i = I(U_i^{\mathrm{TC}}; V_i^{\mathrm{TC}}) + \delta_N = \frac{1}{2} \log \frac{\widehat{\sigma}_i^2}{d_i} + \delta_N, \tag{4.37}$$

where U_i^{TC} and V_i^{TC} are scalar test-channel random variables defined in Section 4.4.1 and $\lim_{N\to\infty} \delta_N = 0$. We claim that, for each node $v_i \in \mathcal{V}$,

$$\widehat{\sigma}_i^2 \leq \sigma_{\mathcal{S}_i}^2. \tag{4.38}$$

Proof. See Appendix B.2.2. □

This leads to

$$R_i \leq \frac{1}{2} \log \frac{\sigma_{\mathcal{S}_i}^2}{d_i} + \delta_N. \tag{4.39}$$

Summing up (4.39) over all links, we obtain the first inequality (4.26) in Theorem 4.4.1.

The codebook $\mathcal{C}_i$ is revealed to v_i's parent-node $v_{\mathrm{PN}(i)}$. At the beginning of the distributed computation scheme, each leaf v_l uses $w_l \mathbf{x}_l$ as the estimate $\widehat{\mathbf{s}}_l$. During the distributed computation scheme, as shown in Fig. 4.1, each non-leaf node v_b, upon

[3]Notice that the rate of this code should be $\log_2(2^{NR_i} + 1) \approx R_i$. However, when $N \to \infty$ (which is the case considered in this section), the code rate converges to R_i. In other words, a single codeword $\mathbf{c}_i(0)$ has asymptotically no effect on the coding rate.

receiving description indices $M_{1b}, M_{2b}, \ldots M_{db}$ from the d children $v_1, \ldots v_d$, decodes these description indices, computes the sum of these descriptions and the data vector generated at v_b as follows

$$\widehat{\mathbf{s}}_b = \sum_{k=1}^{d} \mathbf{c}_k(M_{k\to b}) + w_b \mathbf{x}_b, \tag{4.40}$$

and re-encodes $\widehat{\mathbf{s}}_b$ into a new description index $M_{b\to a} \in [1 : 2^{NR_b}]$ and sends the description index to the parent-node v_a using rate R_b. We denote the reconstructed description by $\widehat{\mathbf{r}}_b = \mathbf{c}_b(M_{b\to a})$. The decoding and encoding at the node v_b are defined as follows. Note that the leaves only encode and the root v_0 only decodes.

- **Decoding:** In each codebook $\mathcal{C}_k, k = 1, \ldots d$, use the codeword $\mathbf{c}_k(M_{k\to b})$ as the description $\widehat{\mathbf{r}}_k$. If $v_b = v_0$ is the root, it computes the sum of all codewords $\mathbf{c}_k(M_{k\to 0})$ as the estimate of $\mathbf{y}$:

$$\widehat{\mathbf{y}} = \sum_{v_k \in \mathcal{N}(v_0)} \mathbf{c}_k(M_{k\to 0}) = \sum_{v_k \in \mathcal{N}(v_0)} \widehat{\mathbf{r}}_k. \tag{4.41}$$

- **Encoding:** Find a codeword $\mathbf{c}_b(M_{b\to a}) \in \mathcal{C}_b \setminus \{\mathbf{c}_b(0)\}$ such that the two vectors $\widehat{\mathbf{s}}_b = \sum\limits_{k=1}^{d} \mathbf{c}_k(M_{k\to b}) + w_b\mathbf{x}_b$ and $\widehat{\mathbf{r}}_b = \mathbf{c}_b(M_{b\to a})$ are jointly typical with respect to the test-channel distribution $\phi_{U_b^{\mathrm{TC}}, V_b^{\mathrm{TC}}}$ (in $\mathcal{J}_\epsilon^{2N}$). If there are more than one codewords that satisfy this condition, arbitrarily choose one of them. However, if $\widehat{\mathbf{s}}_b = \sum\limits_{k=1}^{d} \mathbf{c}_k(M_{k\to b}) + w_b\mathbf{x}_b$ is not typical with respect to the test-channel distribution $\phi_{U_b^{\mathrm{TC}}}$ (not in $\mathcal{T}_{U,\varepsilon}^{N}$), or if there is no codeword in $\mathcal{C}_b \setminus \{\mathbf{c}_b(0)\}$ that satisfies the joint typicality condition, send description index $M_{b\to a} = 0$ (note that this means the index of the 0-th random codeword $\mathbf{c}_b(0)$, instead of a vector $\mathbf{0}^N$).

Since all codebooks $\mathcal{C}_k, k = 1, \ldots d$, have been revealed to v_b, the decoding is always successful, in that the decoding process is simply the mapping from the description index $M_{k\to b}$ to the description $\mathbf{c}_k(M_{k\to b})$. However, the encoding may fail. In this case, the description index $M_{b\to a} = 0$ is sent and this description index is decoded to a predetermined random sequence $\mathbf{c}_b(0)$ on the receiver side. Note that the rate R_i is the same as $R_{i\to \mathrm{PN}(i)}$ in (4.3), and the notation R_i is used here for simplicity. We still use the notation $R_{i\to \mathrm{PN}(i)}$ for the results on network consensus, where each node may have to send descriptions to different nodes, and $R_{i\to \mathrm{PN}(i)}$ can usefully indicate that the direction of information transmission is from the node v_i to its parent node $v_{\mathrm{PN}(i)}$.

4.4.3 The proof of Theorem 4.4.1: analysis of the Gaussian random codes

In this part, we analyze the expected distortion of the Gaussian random codes. Note that, unless specifically clarified, all results in this part are stated for the random coding ensemble, i.e., the expectation $\mathbb{E}[\cdot]$ and the probability $\Pr(\cdot)$ are taken over random data sampling, codeword selection and random codebook generation. The result in Theorem 4.4.1 holds for at least one code in this random coding ensemble.

The following Lemma 4.4.2 states that the estimate $\widehat{\mathbf{s}}_b$ and the description $\widehat{\mathbf{r}}_b$ are jointly typical for all b with high probability.

Lemma 4.4.2 (Covering Lemma for Lossy In-network Matrix-vector Multiplication)**.** *For the encoding and decoding schemes as described in this section, denote by $E_i = 1$ the event that the encoding at the node v_i is not successful. Then*

$$\lim_{N\to\infty} \sup_{1\le i\le n} \Pr(E_i = 1) = 0, \tag{4.42}$$

where the probability is taken over random data sampling and random codebook generation.

Proof. See Appendix C-C in [279]. □

In Lemma 4.4.3, we provide bounds on the variances of $\widehat{\mathbf{s}}_b$ and $\widehat{\mathbf{r}}_b$. Note that the inequalities in Lemma 4.4.3 do not trivially follow from the typicality of $\widehat{\mathbf{s}}_b$ and $\widehat{\mathbf{r}}_b$ because the typicality of $\widehat{\mathbf{s}}_b$ only ensures that $\frac{1}{N}\|\widehat{\mathbf{s}}_b\|_2^2 - \widehat{\sigma}_b^2$ converges to zero in probability, while (4.43) requires convergence in mean value to zero. This is a standard issue. In [279], we use a standard technique to overcome this issue. The key idea is that, for non-typical case (when encoding fails), we send a predetermined random sequence, on the variance of which we can provide a bound.

Lemma 4.4.3. *At each node v_b, the description $\widehat{\mathbf{r}}_b = \mathbf{c}_b(M_{b\to a})$ and the estimate $\widehat{\mathbf{s}}_b$ defined in* (4.40) *satisfy*

$$\left|\mathbb{E}\left[\frac{1}{N}\|\widehat{\mathbf{s}}_b\|_2^2\right] - \widehat{\sigma}_b^2\right| < \varepsilon_N, \tag{4.43}$$

$$\left|\mathbb{E}\left[\frac{1}{N}\|\widehat{\mathbf{r}}_b\|_2^2\right] - (\widehat{\sigma}_b^2 - d_b)\right| < \varepsilon_N, \tag{4.44}$$

$$\left|\mathbb{E}\left[\frac{1}{N}\|\widehat{\mathbf{r}}_b - \widehat{\mathbf{s}}_b\|_2^2\right] - d_b\right| < \varepsilon_N, \tag{4.45}$$

where $\lim_{N\to\infty}\varepsilon_N = 0$.

Proof. See Appendix C-D in [279]. □

Lemma 4.4.4. *At each node v_b, the description $\widehat{\mathbf{r}}_b = \mathbf{c}_b(M_{b\to a})$ and the estimate $\widehat{\mathbf{s}}_b$ defined in* (4.40) *satisfy*

$$h(\widehat{\mathbf{s}}_b) > \frac{N}{2}\log_2 2\pi e\widehat{\sigma}_b^2 - N\beta_N, \tag{4.46}$$

$$h(\widehat{\mathbf{r}}_b) > \frac{N}{2}\log_2 2\pi e(\widehat{\sigma}_b^2 - d_b) - N\beta_N, \tag{4.47}$$

where $\lim_{N\to\infty}\beta_N = 0$, $h(\cdot)$ is the differential entropy function, and the random vectors $\widehat{\mathbf{s}}_b$ and $\widehat{\mathbf{r}}_b$ are defined in the probability space that contains the random codebook generation[4]*.*

[4]To define differential entropy for the two random vectors $\widehat{\mathbf{s}}_b$ and $\widehat{\mathbf{r}}_b$, we need to first define the densities (with respect to the Lebesgue measure) of the two random vectors. The estimate $\widehat{\mathbf{s}}_b$ is certainly absolutely

Proof. See Appendix C-E in [279]. □

Lemma 4.4.4 indicates that $\widehat{\mathbf{s}}_b$ and $\widehat{\mathbf{r}}_b$ are close to Gaussian-distributed random variables in differential entropy sense. We will use Lemma 4.4.3 and Lemma 4.4.4 to show a non-trivial relationship between the Gaussian-code-based distortion d_i and the MMSE-based incremental distortion D_i^{Inc}. This relationship is characterized in Lemma 4.4.5. The proof is based on an observation that, when the true distribution of the source is close (in the sense of differential entropy) to the expected distribution, the estimate based on random coding can provide a distortion that is approximately equal to the MMSE estimate.

Remark 8. If we try to directly obtain the overall distortion bound in (4.27) using some classical coding schemes such as Wyner-Ziv coding [59, Chapter 15.9], we have to prove that the incremental errors (the term d_i) due to successive quantizations along the network are 'approximately uncorrelated' (so that d_i for different i can be summed up to obtain the bound on the overall distortion (4.27)). While we do not pursue this direction, the above may be achieved by obtaining a non-trivial generalization of the "Markov Lemma" [89, Lecture Notes 13] to Gaussian vectors. To bypass this difficulty, we directly relate the Gaussian-code-based distortion d_i and the MMSE-based distortion D_i^{Tx}, which simultaneously shows some nontrivial connections between Gaussian random codes and MMSE estimates. This is why the proof of the inner bound is conceptually different from existing literature.

Recall that the MMSE estimate of the sum $\mathbf{y}_{\mathcal{S}_i}$ at the node v_j is denoted by $\widehat{\mathbf{y}}_{\mathcal{S}_i,j}^{\text{mmse}} = \mathbb{E}_{\mathcal{C}_i}\left[\mathbf{y}_{\mathcal{S}_i}|I_j\right]$, where I_j, as before, denotes the information available to the node v_j. Define D_i^{Tx}, D_i^{Rx} and D_i^{Inc} similar to (4.7), (4.8) and (4.9). That is, $D_i^{\text{Tx}} = \mathbb{E}\left[\frac{1}{N}\left\|\mathbf{y}_{\mathcal{S}_i} - \widehat{\mathbf{y}}_{\mathcal{S}_i,i}^{\text{mmse}}\right\|_2^2\right]$, $D_i^{\text{Rx}} = \mathbb{E}\left[\frac{1}{N}\left\|\mathbf{y}_{\mathcal{S}_i} - \widehat{\mathbf{y}}_{\mathcal{S}_i,\text{PN}(i)}^{\text{mmse}}\right\|_2^2\right]$ and $D_i^{\text{Inc}} = \frac{1}{N}\mathbb{E}\left[\left\|\widehat{\mathbf{y}}_{\mathcal{S}_i,\text{PN}(i)}^{\text{mmse}} - \widehat{\mathbf{y}}_{\mathcal{S}_i,i}^{\text{mmse}}\right\|_2^2\right]$. Notice that the inner $\mathbb{E}[\cdot]$ (for the MMSE estimate $\widehat{\mathbf{y}}_{\mathcal{S}_i,j}^{\text{mmse}} = \mathbb{E}_{\mathcal{C}_i}\left[\mathbf{y}_{\mathcal{S}_i}|I_j\right]$) is for a given codebook $\mathcal{C}_i$ at v_i, because both v_i and its parent $v_{\text{PN}(i)}$ know the codebook $\mathcal{C}_i$ (see the codebook construction in Section 4.4.2). However, the outer $\mathbb{E}[\cdot]$ (for $D_i^{\text{Tx}} = \mathbb{E}\left[\frac{1}{N}\left\|\mathbf{y}_{\mathcal{S}_i} - \widehat{\mathbf{y}}_{\mathcal{S}_i,i}^{\text{mmse}}\right\|_2^2\right]$) is still taken over both the codeword selection and the random codebook generation. In this subsection, the quantities D_i^{Tx}, D_i^{Rx} and D_i^{Inc} are all averaged over the random codebook ensemble.

Lemma 4.4.5. *For an arbitrary node v_i*

$$\sqrt{d_i - \varepsilon_N} - \eta_N \leq \sqrt{D_i^{\mathit{Inc}}} \leq \sqrt{d_i + \varepsilon_N} + \eta_N, \tag{4.48}$$

where $\lim_{N\to\infty} \eta_N = 0$ *and* ε_N *is the same as in* (4.45). *Further, the mean-square difference between*

continuous, because it is smoothed by the Gaussian random variable $\mathbf{x}_b$ (see (4.40)). However, conditioned on a specific instance of the Gaussian codebooks, the random vector $\widehat{\mathbf{r}}_b$ has a finite support, and the (conditional) differential entropy of $\widehat{\mathbf{r}}_b$ is $-\infty$. To overcome this difficulty, we cast the analysis on the unconditional distribution of $\widehat{\mathbf{r}}_b$, i.e., taking into account the code generation randomness. In this way, $\widehat{\mathbf{r}}_b$ is also absolutely continuous.

the MMSE estimate $\widehat{\mathbf{y}}_{\mathcal{S}_i,i}^{mmse}$ and the estimate $\widehat{\mathbf{s}}_i$ based on Gaussian random codes satisfies

$$\frac{1}{N}\mathbb{E}\left[\left\|\widehat{\mathbf{s}}_i - \widehat{\mathbf{y}}_{\mathcal{S}_i,i}^{mmse}\right\|_2^2\right] \leq \Delta_N, \tag{4.49}$$

where $\lim\limits_{N\to\infty} \Delta_N = 0.$

Proof. See Appendix B.2.3. □

Since the distributed computation scheme using Gaussian random codes in Theorem 4.4.1 (see Section 4.4.2) satisfies the model in Section 4.2, the distortion accumulation result in Theorem 4.3.1 holds, i.e.,

$$\frac{1}{N}\mathbb{E}\left[\left\|\mathbf{y} - \widehat{\mathbf{y}}_{\mathcal{S}_0,0}^{\text{mmse}}\right\|_2^2\right] = \sum_{i=1}^{n} D_i^{\text{Inc}}, \tag{4.50}$$

where $\mathbf{y}$ is the overall weighted sum, $\widehat{\mathbf{y}}_{\mathcal{S}_0,0}$ is the MMSE estimate of $\mathbf{y}$ at the sink v_0, and all expectation operations are taken over the random codebook ensemble. Using (4.49) in Lemma 4.4.5, we have that

$$\frac{1}{N}\mathbb{E}\left[\left\|\widehat{\mathbf{y}} - \widehat{\mathbf{y}}_{\mathcal{S}_0,0}^{\text{mmse}}\right\|_2^2\right] \leq \Delta_N, \tag{4.51}$$

where $\lim_{N\to\infty} \Delta_N = 0$ and $\widehat{\mathbf{y}}$ is the estimate of the overall sum $\mathbf{y}$ at the sink using random Gaussian code. From Lemma 4.1.1, we have that

$$\mathbb{E}\left[\|\widehat{\mathbf{y}} - \mathbf{y}\|_2^2\right] = \mathbb{E}\left[\left\|\mathbf{y} - \widehat{\mathbf{y}}_{\mathcal{S}_0,0}^{\text{mmse}}\right\|_2^2\right] + \mathbb{E}\left[\left\|\widehat{\mathbf{y}} - \widehat{\mathbf{y}}_{\mathcal{S}_0,0}^{\text{mmse}}\right\|_2^2\right]. \tag{4.52}$$

Plugging in (4.50), (4.51) and using the triangle inequality, we get

$$D = \frac{1}{N}\mathbb{E}\left[\|\widehat{\mathbf{y}} - \mathbf{y}\|_2^2\right] \leq \sum_{i=1}^{n} D_i^{\text{Inc}} + \Delta_N. \tag{4.53}$$

Using (4.48) in Lemma 4.4.5, we get

$$\begin{aligned} D \leq &\sum_{i=1}^{n}\left(\sqrt{d_i + \varepsilon_N} + \eta_N\right)^2 \\ = &\sum_{i=1}^{n} d_i + \sum_{i=1}^{n} \varepsilon_N + \eta_N^2 + 2\eta_N\sqrt{d_i + \varepsilon_N}. \end{aligned} \tag{4.54}$$

By defining $\epsilon_N = \sum\limits_{i=1}^{n} \varepsilon_N + \eta_N^2 + 2\eta_N\sqrt{d_i + \varepsilon_N}$, we get

$$D \leq \sum_{i=1}^{n} d_i + \epsilon_N, \tag{4.55}$$

where $\lim_{N\to\infty}\epsilon_N = 0$. Finally, noticing that (4.55) holds for the random code ensemble, we can find at least one code in the ensemble such that the distortion bound (4.27) holds.

Since we can tune the distortion parameter d_i directly, we can set $d_1 = d_2 = \ldots = d_n = d$. Then, in the limit of large N, $D = nd$, which means that $d_1 = d_2 = \ldots = d_n = D/n$. Thus, we can obtain the minimized achievable result $R = \frac{1}{2}\log_2 \frac{\prod_{i=1}^{n} \sigma^2_{\mathcal{S}_i}}{(D/n)^n}$, which is (4.28) in Theorem 4.4.1.

4.5 Extension to network consensus

The results in the preceding sections can be extended to the case when each node in the network $\mathcal{T}$ wants to obtain an estimate of $\mathbf{y} = \sum_{i=1}^{n} w_i \mathbf{x}_i$. Note that the network consensus problem considered in this chapter is a generalization of average consensus, which is the case where $w_i = \frac{1}{n}, \forall i$.

Define $\mathcal{S}_{i\to j} \subset \mathcal{V}$ as the set that contains node v_i and all its descendants when neighboring node v_j is defined to be the parent-node of v_i. As in (4.7)-(4.9), define

$$D^{\mathrm{Tx}}_{i\to j} = \frac{1}{N}\mathbb{E}\left[\left\|\mathbf{y}_{\mathcal{S}_{i\to j}} - \widehat{\mathbf{y}}^{\mathrm{mmse}}_{\mathcal{S}_{i\to j},i}\right\|_2^2\right], \tag{4.56}$$

$$D^{\mathrm{Rx}}_{i\to j} = \frac{1}{N}\mathbb{E}\left[\left\|\mathbf{y}_{\mathcal{S}_{i\to j}} - \widehat{\mathbf{y}}^{\mathrm{mmse}}_{\mathcal{S}_{i\to j},j}\right\|_2^2\right], \tag{4.57}$$

and

$$D^{\mathrm{Inc}}_{i\to j} = \frac{1}{N}\mathbb{E}\left[\left\|\widehat{\mathbf{y}}^{\mathrm{mmse}}_{\mathcal{S}_{i\to j},i} - \widehat{\mathbf{y}}^{\mathrm{mmse}}_{\mathcal{S}_{i\to j},j}\right\|_2^2\right], \tag{4.58}$$

where $\widehat{\mathbf{y}}^{\mathrm{mmse}}_{\mathcal{S}_{i\to j},i}$ and $\widehat{\mathbf{y}}^{\mathrm{mmse}}_{\mathcal{S}_{i\to j},j}$ are defined by (4.6), i.e., the MMSE estimates of $\mathbf{y}_{\mathcal{S}_{i\to j}}$ with information at v_i or at v_j. Since each node v_i makes an estimate of the matrix-vector multiplication result $\mathbf{y}$, for a given distributed computation scheme, we define the overall distortion of the MMSE estimate $\widehat{\mathbf{y}}^{\mathrm{mmse}}_i$ of $\mathbf{y}$ at the node v_i as

$$D^{\mathrm{mmse}}_i = \frac{1}{N}\mathbb{E}\left[\|\widehat{\mathbf{y}}^{\mathrm{mmse}}_i - \mathbf{y}\|_2^2\right]. \tag{4.59}$$

For the same distributed computation scheme, define the overall distortion of the estimate $\widehat{\mathbf{y}}_i$ of $\mathbf{y}$ at the node v_i as

$$D^{\mathrm{Total}}_i = \frac{1}{N}\mathbb{E}\left[\|\widehat{\mathbf{y}}_i - \mathbf{y}\|_2^2\right]. \tag{4.60}$$

Then, we have that $D^{\mathrm{Total}}_i \geq D^{\mathrm{mmse}}_i$. For a feasible and oblivious distributed computation scheme $(T, \mathscr{S}, \mathscr{G}, \mathbf{v}, \mathbf{e}) \in \mathcal{F}$ (see the distributed computation model in Section 4.2), the sum rate R is defined in the same way as in the problem of matrix-vector multiplication:

$$R = \sum_{i=1}^{n} \sum_{v_j \in \mathcal{N}(i)} R_{i\to j}. \tag{4.61}$$

The distortion is defined as the sum distortion

$$D = \sum_{i=1}^{n} D_i^{\text{Total}}. \tag{4.62}$$

Thus, the problem to be considered is

$$\begin{aligned} \min_{(T,\mathscr{S},\mathscr{G},\mathbf{v},\mathbf{e})\in\mathcal{F}} \quad & R, \\ \text{s.t. } D \le D^{\text{tar}}. \end{aligned} \tag{4.63}$$

We define $\overrightarrow{\mathcal{T}}_k$ as the edge set of the directed tree towards the root v_k. The set $\overrightarrow{\mathcal{T}}_k$ can be written as

$$\begin{aligned} \overrightarrow{\mathcal{T}}_k = \{&\text{directed edges } (v_i, v_j) : \ (v_i, v_j) \in \mathcal{E}, \text{ and } v_j \\ &\text{is the parent node of } v_i \text{ when } v_k \text{ is defined as the root}\}. \end{aligned}$$

In all, we define n different directed edge sets of directed trees towards n different roots. These directed trees are all defined based on the original tree $\mathcal{T}$. The only difference is that the edges are directed. We use $(i,j) \in \overrightarrow{\mathcal{T}}_k$ to represent that the ordered pair (i,j) is a directed edge in the directed edge set $\overrightarrow{\mathcal{T}}_k$.

Theorem 4.5.1 (Distortion Accumulation for Network Consensus). *For the network consensus problem, the overall distortion of estimating Y at the node v_k satisfies*

$$D_k^{mmse} = \sum_{(i,j)\in\overrightarrow{\mathcal{T}}_k} D_{i\to j}^{Inc}, \tag{4.64}$$

where $\overrightarrow{\mathcal{T}}_k$ is the directed edge set of the directed tree towards the root v_k, and $D_{i\to j}^{Inc}$ is as defined in (4.58).

Proof. See Appendix B.3.1. □

4.5.1 Inner and outer bounds based on incremental-distortion

Recall that $\sigma_{\mathcal{S}}^2$ is the variance of $Y_{\mathcal{S}}$. The counterpart of Theorem 4.3.2 is stated as follows.

Theorem 4.5.2 (Incremental-Distortion-Based Outer Bound for Network Consensus). *For the network consensus problem, given a feasible distributed computation scheme, the sum rate is lower-bounded by*

$$\begin{aligned} R &= \sum_{i=1}^{n} \sum_{v_j\in\mathcal{N}(i)} R_{i\to j} \\ &\ge \frac{1}{2}\sum_{i=1}^{n} \sum_{v_j\in\mathcal{N}(i)} \left[\log_2 \frac{\sigma_{\mathcal{S}_{i\to j}}^2}{D_{i\to j}^{Inc}} - \frac{D_{i\to j}^{Tx}}{2w_i^2} - \frac{\log_2 e}{2\sigma_{\mathcal{S}_{i\to j}}^2}\sqrt{2D_{i\to j}^{Tx}\left(4\sigma_{\mathcal{S}_{i\to j}}^2 + D_{i\to j}^{Tx}\right)}\right] \\ &= \frac{1}{2}\sum_{i=1}^{n} \sum_{v_j\in\mathcal{N}(i)} \left[\log_2 \frac{\sigma_{\mathcal{S}_{i\to j}}^2}{D_{i\to j}^{Rx} - D_{i\to j}^{Tx}} - \mathcal{O}\left((D_{i\to j}^{Tx})^{1/2}\right)\right], \end{aligned} \tag{4.65}$$

where $\sigma^2_{\mathcal{S}_{i\to j}}$ is the variance of each entry of the partial sum $\mathbf{y}_{\mathcal{S}_{i\to j}} = \sum_{v_k\in\mathcal{S}_{i\to j}} w_k\mathbf{x}_k$, and $D^{Tx}_{i\to j}$, $D^{Rx}_{i\to j}$ and $D^{Inc}_{i\to j}$ are respectively defined in (4.56), (4.57) *and* (4.58). *By optimizing over the incremental distortions $D^{Inc}_{i\to j}$, one obtains the following scheme-independent bound stated in an optimization form*

$$\min_{D^{Inc}_{i\to j},\forall(i,j)\in\mathcal{E}} \frac{1}{2}\sum_{i=1}^{n}\sum_{v_j\in\mathcal{N}(i)}\left[\log_2\frac{\sigma^2_{\mathcal{S}_{i\to j}}}{D^{Inc}_{i\to j}} - \frac{D^{Tx}_{i\to j}}{2w_i^2} - \frac{\log_2 e}{2\sigma^2_{\mathcal{S}_{i\to j}}}\sqrt{2D^{Tx}_{i\to j}\left(4\sigma^2_{\mathcal{S}_{i\to j}} + D^{Tx}_{i\to j}\right)}\right],$$
$$s.t.\begin{cases} D^{Tx}_{i\to j} = \sum_{v_k\in\mathcal{S}_{i\to j}\setminus\{v_i\},(k,l)\in\vec{\mathcal{T}}_j} D^{Inc}_{k\to l}, \forall(i,j)\in\mathcal{E}, \\ D \geq D^{mmse}_k = \sum_{(i,j)\in\vec{\mathcal{T}}_k} D^{Inc}_{i\to j}. \end{cases} \tag{4.66}$$

Proof. See Appendix B.3.2. □

Then, we present an achievable result using Gaussian codes to show that the outer bound in Theorem 4.5.2 is tight in the low distortion regime.

Theorem 4.5.3 (Inner Bound for Network Consensus)**.** *Using Gaussian random codebooks, we can find a distributed computation scheme, such that the sum rate R satisfies*

$$R \leq \frac{1}{2}\sum_{i=1}^{n}\sum_{v_j\in\mathcal{N}(i)}\log_2\frac{\sigma^2_{\mathcal{S}_{i\to j}}}{d_{i\to j}} + (2n-2)\delta_N, \tag{4.67}$$

where $\lim_{N\to\infty}\delta_N = 0$, and the $d_{i\to j}$'s are distortion parameters. Further, the overall distortion D in all nodes v_i satisfies

$$D < \sum_{k=1}^{n}\sum_{(i,j)\in\vec{\mathcal{T}}_k} d_{i\to j} + n\epsilon_N, \tag{4.68}$$

where $\lim_{N\to\infty}\epsilon_N = 0$.

Proof. See Appendix B.3.3. □

If we ignore the small gap between the inner bound (4.67) and the outer bound (4.65) when the resolution level D is fine enough, the optimal rate can be obtained by solving the following convex optimization problem:

$$\min_{D^{\text{Inc}}_{i\to j},\forall(i,j)\in\mathcal{E}} \frac{1}{2}\sum_{i=1}^{n}\sum_{v_j\in\mathcal{N}(i)}\log_2\frac{\sigma^2_{\mathcal{S}_{i\to j}}}{D^{\text{Inc}}_{i\to j}},$$
$$s.t.\quad \sum_{k=1}^{n}\sum_{(i,j)\in\vec{\mathcal{T}}_k} D^{\text{Inc}}_{i\to j} \leq D. \tag{4.69}$$

Remark 9. The rate-distortion outer bound in (4.65) depends on the distributed computation scheme. Using convex optimization techniques, we can minimize over all incremental distortions $D^{\text{Inc}}_{i\to j}$ with the linear constrains specified by (4.64) to obtain a fundamental outer bound on the rate-distortion function of distributed consensus. The outer bound is essentially obtained by rate allocation in the network. If the $\mathcal{O}(D^{1/2})$ gap between the inner and outer bound is neglected, the rate (measured in number of bits) allocated to the link $v_i \to v_j$ is $\frac{1}{2}\log_2\frac{\sigma^2_{\mathcal{S}_{i\to j}}}{D^{\text{Inc}}_{i\to j}}$.

We consider a special case when $w_i = \frac{1}{n}, \forall i$. This is the classical case of lossy distributed network consensus with the same distortion requirement at all nodes [240]. We again consider the line network as shown in Section 4.3.3. In this case, it can be shown that the optimal solution is

$$D^{\text{Inc}}_{i\to j} = \frac{D}{2(n-1)}, \forall (i,j) \in \mathcal{E},$$

if all $\mathcal{O}\left((D^{\text{Tx}}_{i\to j})^{1/2}\right)$ terms are neglected, in the limit of zero-distortion (high resolution)[5]. Similar with the data-aggregation case, this solution for network consensus is also very similar to the reverse water-filling solution for parallel Gaussian lossy source coding problem [59, Theorem 10.3.3] in the limit of large rate (zero distortion). This solution yields a sum rate of $\mathcal{O}\left(n\log_2\frac{1}{D}\right)$. The classical outer bound [240, Prop. 4] about the distributed network consensus in a tree network is $\mathcal{O}\left(n\log_2\frac{1}{n^{3/2}D}\right)$. This means that our result is certainly tighter than the classical result in a line network in the zero-distortion limit. Moreover, this $\mathcal{O}(n\log n)$ gap is also consistent with the $\log(n!)$ gap in Section 4.3.3.

4.6 Conclusions and future directions

In this chapter, we have considered the lossy matrix-vector multiplication problem in tree network, where each node has a Gaussian random vector. Our results show that the phenomenon of information dissipation (error accumulation) exists in this problem, and by quantifying the information dissipation, we obtain an information-theoretic outer bound on the rate-distortion function that is tighter than classical cut-set bounds for lossy matrix-vector multiplication for both data aggregation and network consensus problems. The improvement on the classical cut-set bounds can be made arbitrarily large when the diameter of the network is large enough. The results also show that linear Gaussian codes can achieve within $\mathcal{O}(\sqrt{D})$ of the obtained outer bound, which means that our outer bound is tight when the required distortion is small (high resolution scenario). A meaningful future direction is to investigate tighter outer bound for all values of D, and investigate compression algorithms, e.g., lattice codes, that achieve the outer bound for all values of D.

[5]We can neglect the $\mathcal{O}\left((D^{\text{Tx}}_{i\to j})^{1/2}\right)$ terms, because in the zero-distortion limit, $\log\frac{1}{D^{\text{Inc}}_{i\to j}} > \log\frac{1}{D^{\text{Tx}}_{i\to j}} >> (D^{\text{Tx}}_{i\to j})^{1/2}$.

Chapter 5

Exploiting the multi-stage computing I: new platforms and coded elastic computing

5.1 Introduction

Starting from this chapter, we will review three coding-based techniques that are tightly related to the problem of multi-stage computing, i.e., a computing problem that is implemented in several communication rounds and (maybe) using similar or repeated computation patterns. This chapter will introduce a new computation platform on elastic computing where multi-stage computing problems are critical, which motivates the study of coding techniques for multi-stage problems. In Chapter 6 and Chapter 7, we provide other two techniques that intentionally use properties of multi-stage computing to improve the performance of coded computing. The results of these three chapters are from a series of papers on coding techniques for multi-stage computing problems [272, 277, 280, 281]. The results on elastic computing will be updated in the online report [281].

First, we provide the background of the study on elastic computing. New offerings from cloud-service providers allow exploiting under-utilized Virtual Machines (VMs) at a fraction of the original cost. Such offerings, however, have the drawback that machines can be preempted at any time if a high-priority job appears. This, in turn, will surface as a computation failure at the application level. While common distributed machine learning frameworks are already built with fault-tolerance [6, 174], they often assume that failures are transient and rare. Due to this assumption, machine failures are often recovered by a "stop-the-world" scheme whereby the entire system is forced to wait until regular execution on the failure machines is restored from the previous state (eventually on new machines). The above assumptions, however, do not necessarily hold for failures due to machines being preempted because (1) these failures are permanent and local data may not be accessible anymore; (2) several machines can be preempted altogether in an elastic event; (3) these failures add up to transient failures, therefore leading to more frequent disruptions during computation; and (4) the computational framework may need to acquire additional machines to compensate, meaning that data has to be copied on the new machines which will likely become stragglers for the running computation. In practice, we observed situations at scale where the stop-the-world scheme results in zero computation progress because, by the time a failure is recovered, a new failure occurs. This results in the necessity to build an elastic run-time framework and related failure-aware algorithms which can continue the computation and flexibly adapt in the presence of failures. Another possible technique to deal with preemption is to view the preempted machines as erasure-type faults and ignore them. Although machine learning algorithms are robust to small transient faults, merely ignoring the computational results in these permanently-failed preempted machines may result in algorithmic-level performance loss. The influence of ignoring the computation results for the preemption type of faults is also more severe than usual computation faults because the number of failures can be huge. Similarly, even if the data are redundant in some applications, ignoring partial results may still lead to reduced confidence levels on the prediction accuracy. It may also not be desirable from a customer's perspective who often requires the full dataset to be present during the entire training process in order to achieve the highest accuracy.

In order to deal with the aforementioned problems, in this chapter, we present *coded elastic computing*: a novel distributed learning framework allowing users to train their machine learning models over preemptable machines. In our coded elastic computing framework machines are allowed to arbitrarily join or leave during a distributed iterative learning task thanks to the introduction of redundancy in the computation. Coded elastic computing can flexibly change the workload of each machine at runtime based on the number of available machines by selecting to use only a subset of the encoded data in a cyclic fashion. Apart from providing fault-tolerance when machines are preempted, coded elastic computing is also *positively-elastic* in that it can utilize the properties of the coded data to reduce the workload at existing machines flexibly when new machines join the computation. We will show that the coded elastic computing framework can make the computational cost at each machine scale inversely with the number of machines, which leads to linear scaling of theoretical computational cost. The proposed technique is also useful in other applications besides elastic computation when the number of machines needs to be dynamically adjusted during a learning task, such as when the number of machines is tuned as a hyper-parameter, or when the machines have to be reallocated to achieve fairness [116] between users or the specific need of some users at runtime.

The coded elastic computing technique is tested in the multi-tenancy cluster at Microsoft as an example of the Apache REEF EGC (Elastic Group Communication) framework. Apache REEF is a library that helps develop distributed high-performance applications on top of cluster resource managers such as YARN [250] and Mesos [116]. The Apache REEF project provides a set of abstractions and reusable functional blocks to ease the process of building cloud-scale applications. EGC is a distributed communication framework which extends Apache REEF by providing an API allowing to implement elastic computations by chaining fault-tolerant MPI-like primitives. Based on the EGC framework, we test the proposed technique for a coded implementation of linear regression on a real dataset when machines can leave and join the computation. We show that the current technique can obtain the same convergence behavior as ordinary gradient-descent-based algorithms but can elastically allocate the work load based on the number of available machines without moving data at the existing machines. We also compare with other baselines, such as ignore, replication and an existing algorithm called Elastic Distr-BGD [180] to show the improvement of the proposed technique in terms of the model generalization error.

In this chapter, we first present a coded elastic matrix-vector multiplication algorithm to illustrate the main idea. Then, we present the generalizations of the proposed technique in broader applications, namely matrix-matrix multiplication, linear regression, and master-free fully-distributed computing. Finally, we validate the approach with a set of experiments.

5.2 System model and problem formulation

We consider the case when a data matrix $\mathbf{X}$ is stored distributedly at several machines. Note that this is the most typical case for large-scale data training. Note that the results in this chapter are different from Chapter 4 and 8 in that there is no underlying graph structure.

Elastic events can happen during which existing machines can be preempted and new machines can be added to the computation. We use n to denote the number of existing machines, and m to denote the memory cost at each machine. Note that n and m can both change over time in the elastic computing settings, and n can even become greater than P when new machines join. In [281], we identified some properties of the typical elastic events that can happen in large-scale distributed systems. We now quote them in the following.

Property 1. When a preemption failure happens, which machine(s) to be preempted is decided by the resource allocator and is not known in advance.

Property 2. The preemption is permanent, meaning that the preempted machines are going to be removed forever. However, new machines may join after an unknown time.

Property 3. After an elastic event happens, the entire system gets the information. That is, if some machines leave or join, the other machines know immediately about which machines leave or join.

The second and the third properties of the properties mentioned above differentiate the elastic events from common machine failures and stragglers because (1) new machines can join the computation, and (2) one may adapt the computation scheme instantly after an elastic event and possibly utilize the newly available resources. This is because, although we do not know which machines are going to be taken away, and we do not know how many machines are going to be taken away, we know that, **the network configuration does not change before the next elastic event**. In Section 5.3.1 and Section 5.3.2, we provide the elastic data partitioning scheme which can actively utilize resource elasticity.

5.2.1 Definition of computation elasticity

In this section, we review the formal definitions of elasticity provided in [281]. The main intuition behind these definitions is that we would like to achieve flexible and fast transitions between different configuration points (which we will define below) when elastic events happen, without introducing communication overhead.

Denote by P the initial number of machines. A configuration point is a tuple (n, m) that uses n machines, with the memory cost at each machine not exceeding m. Note that n means the number of the currently available machines, and can change over time. An achievable computation policy at a configuration point is said to be *optimal* if it obtains the optimal tuple (e, u), where e is the number of machine preemptions that the policy can tolerate (i.e., the system can continue to compute the exact result as if there is no machine preemption), and u is the size of data that a machine actually *selects to use* ($u \leq m$). Note that this is an important property of coded elastic computing: even though we use the redundancy for preemption tolerance, we do not need to utilize the redundancy during the computation when we have the knowledge of the failures. For a configuration point (n, m), we would like to minimize the actually selected data u to reduce the memory access and maximize the number of tolerable machine failures e to provide the best preemption tolerance.

For a given configuration (n, m), i.e., n machines where each machine has memory cost m, denote by $\mathcal{A}_{n,m}$ the set of computing policies that use n machines where each machine has memory cost no more than m. Denote by $\mathcal{A}^*_{n,m} \subset \mathcal{A}_{n,m}$ the set of optimal computing policies, i.e., they obtain the optimal tuple (e^*, u^*).

Definition: (transition compatibility) A pair of policies (a, a') with $a \in \mathcal{A}_{n,m}$ and $a' \in \mathcal{A}_{n',m'}$ is said to be transition compatible if the policy a can be transitioned to policy a' **without** having to move or modify the data in the existing machines in the event of a configuration transition (n,m)$\rightarrow$(n',m').

As we have mentioned in the beginning of Section 5.2.1, the key to the above definition is that we discourage inter-machine data movement in order to make the elastic configuration transitions non-disruptive to ongoing computation tasks.

Definition: (fully transition compatibility) A family of policies $\mathcal{F} = \{a_{n,m}\}$, $n \in \mathcal{N}$, $m \in \mathcal{M}$, is said to be fully transition compatible if every pair ($a_{n,m}$, $a_{n',m'}$) in $\mathcal{F}$ are transition compatible.

Definition: (optimal fully transition compatibility) A family of fully transition compatible policies $\mathcal{F} = \{a_{n,m}\}$, $n \in \mathcal{N}$, $m \in \mathcal{M}$ is said to be optimal if all policies are optimal, i.e., each policy $a_{n,m} \in \mathcal{F}$ is in $\mathcal{A}^*_{n,m}$ and obtains the optimal tuple (e^*, u^*) for the number of tolerable machine preemptions and the size of the selected data to use.

Our goal is two-fold. First, we want to find optimal fully transition compatible families and the conditions under which they exist. Second, we want to study typical computation primitives and try to see if fully-compatible family of policies exist these problems. In Section 5.3 present matrix-vector multiplication techniques that can provide a fully transition compatible family of optimal computation policies with fixed memory cost at each machine, i.e., when m is fixed in different transition compatible policies $a_{n,m}$.

We will generalize these results to other computation problems in Section 5.4. In fact, we will keep updating the online document [278] to include more computation primitives.

5.3 Main results: elastic data partitioning combined with codes

In this section, we initially focus on the problem of matrix-vector multiplication for having a better theoretical understanding of coded elastic computing. Note however that the proposed technique is general enough to be applicable in many machine learning algorithms beyond matrix-vector multiplications.

Before presenting the algorithm in Section 5.3.3, we introduce the main idea of coded elastic computing in Section 5.3.1 and Section 5.3.2. Then, in Section 5.3.4 and Section 5.3.4, we analyze the proposed techniques and prove that they are indeed elastic according to our definition.

5.3.1 Coded data partitioning in the presence of preempted machines

Assume that in the worst-case of preemption failures, there are at least L machines that remain[1]. We will show that the parameter L is also equal to the *recovery threshold* (see Section 2.2.4). In this chapter, we consider repeatedly using the same data but with different input vectors. For matrix multiplications, it means that we compute $\mathbf{X}\mathbf{w}_t$ for $t = 1, 2, \ldots$ for the same $\mathbf{X}$. This computation primitive is applicable in a variety of scenarios, including training linear models, PageRank, model-parallel deep neural networks and many machine learning algorithms at the inference stage.

We partition the data matrix $\mathbf{X}$ into L subsets (or equivalently, submatrices) $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_L$ of equal size. If the total number of data points is not divisible by L, we can use zero-padding. We generate P *coded* data matrices $\mathbf{X}_s^{\text{coded}}, s = 1, 2, \ldots, P$, $(P > L)$, in which each matrix is a linear combination of the form

$$\mathbf{X}_s^{\text{coded}} = \sum_{k=1}^{L} g_{s,k}\mathbf{X}_k \tag{5.1}$$

where each $g_{s,k}$ is a random but predetermined coefficient. Note that this is the same setting as Lemma 2.2.4, so we can use the coded matrices to tolerate $P - L$ failures. Lemma 2.2.4 is critical for the failure recovery. It essentially shows that no matter which machines are preempted, as long as the number of remaining machines is not smaller than L, the *whole information* of the original data is preserved in the remaining machines. This is why we call the parameter L the *recovery threshold*. The parameter L is limited by the storage constraint at each machine. The more redundancy we can add to the data,

[1]The parameter L, or a lower bound of L, is needed for exact computation. However, in many machine learning tasks, one can often optimize with a subset of data due to data redundancy. In that case, knowledge of L is not necessary.

the lower recovery threshold we need, and hence more failures we can tolerate. In our experiments, we use a redundancy factor of P/L = 2, and hence we can at maximum tolerate failures when half of the machines are preempted. We use systematic codes in the experiment. This can provide backward-compatibility to switch between coded computing and uncoded ordinary computing, and at the same time significantly reduce the cost of encoding the data at the preprocessing stage (5.1).

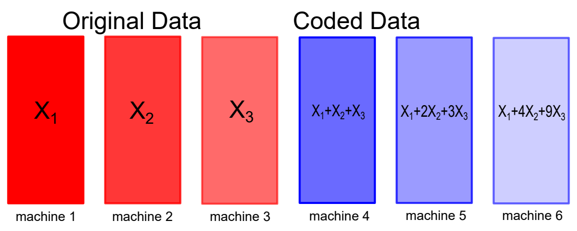
5.3.2 Elastic data partitioning for elastic computation by using data in a cyclic way

According to Lemma 2.2.4, as long as the number of machines that are not preempted is greater or equal to L, the remaining data using the coded data partitioning can preserve the whole information of the original data. However, when the number of machines is strictly larger than L, it becomes redundant to use all the coded data because the data at L machines already preserve the whole information. One may think that this amount of waste is not significant.

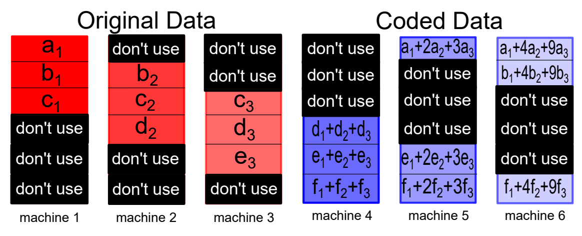
To positively utilize all the remaining machines and achieve the parallel computing capabilities of the extra machines, we select to use data in a cyclic fashion as shown in Figure 5.1. We use a systematic code (see (2.10)) by which the first L of the P coded blocks $\mathbf{X}_s^{\text{coded}}, s = 1, 2, \ldots, P$ are the original data $\mathbf{X}_k, k = 1, 2, \ldots, L$. In Figure 5.1(a) we use red to denote original data and blue to denote the remaining coded data. In this example, the initial number of machines is $P = 6$ and the recover threshold is $L = 3$. From figure 5.1(b) to 5.1(e), we show how to continue the computation when machines are gradually preempted from 6 to 3 (machines correspond to the columns). The stored data remains fixed i.e., the same way as in Figure 5.1(a), but we further partition the data into smaller blocks and only select part of the data to use. Each machine is initially allocated a single subset of coded data $\mathbf{X}_s^{\text{coded}}, s = 1, 2, \ldots, P$, among which L subsets are the original data. Each subset of data is represented as a column in any subfigure of Figure 5.1.

If no failures occur (see Figure 5.1(b)), to remove redundancy from the data, we partition each data block (column) into P *sub-blocks*, and let each machine only use L out of P sub-blocks. By a sub-block of data, we mean one small rectangle in Figure 5.1(b). If $M \geq 1$ machines are preempted (see Figure 5.1(c)-5.1(e)), we partition each data block into $P - M$ sub-blocks, and still let each node only use L out of $P - M$ sub-blocks. If new machines join, they download the coded data previously used in some failed machines or some new linearly combined data, and all the machines, including the ones that just join, use elastic data partitioning based on the current number of available machines.

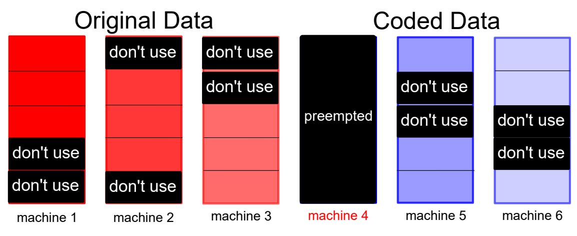
There are two advantages of this type of data usage (1) the overall selected data to use is of the same size as the original data and the selected data across all remaining machines have the same size; and (2) the selected data preserve the whole information of the original data. Thus, we can exactly recover the results while removing the redundancy in the way of using data. These two properties will be formally introduced and proved in Theorem 5.3.1.



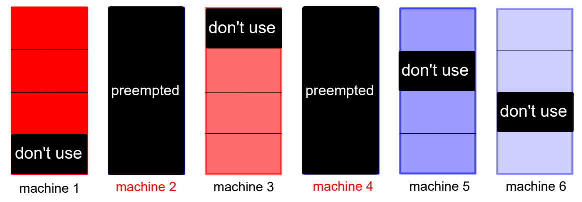
(a) Data encoding without further partitioning



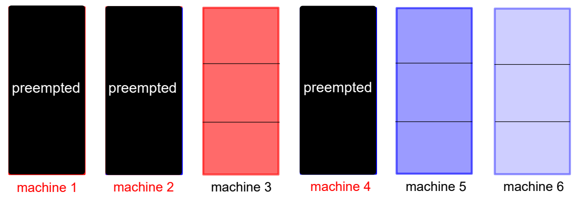
(b) No preempted machine



(c) One preempted machine



(d) Two preempted machines



(e) Three preempted machines

Figure 5.1: The main idea of elastic data partitioning is to use the data in a cyclic way. Each column of data is stored at one machine. For each group (i.e., row block) of data at different machines, there are enough number of sub-blocks that contain all the information. This cyclic way of using data leads to linear scaling of the per-machine computational cost in the number of machines.

5.3.3 Coded elastic computing for matrix-vector multiplications

We provide the detailed procedures of the coded elastic computing algorithm for the repeated matrix-vector multiplication problem $\mathbf{X}\mathbf{w}_t, t = 1, 2, \ldots$ in Algorithm 2. We use $\mathbf{X}_{k,j}^{\text{coded}}$ to represent the j-th sub-block of the data at the k-th machine. We will call $\mathbf{X}_{k,j}^{\text{coded}}$'s with the same j "the j-th group" of sub-blocks which correspond to the j-th row block in any subfigure of Figure 5.1. Note that the number of row blocks changes with the number of preempted machines. We use $\mathbf{G}_j$ to represent the collection of linear combination coefficients for the j-th group (row block) that are selected to use. For example, for the first group (row block) in Figure 5.1(b), we have

$$\mathbf{G}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 2 & 3 \\ 1 & 4 & 9 \end{bmatrix} \tag{5.2}$$

because the three selected sub-blocks are $\mathbf{a}_1$, $\mathbf{a}_1 + 2\mathbf{a}_2 + 3\mathbf{a}_3$ and $\mathbf{a}_1 + 4\mathbf{a}_2 + 9\mathbf{a}_3$. And for the last group (row block) in Figure 5.1(b),

$$\mathbf{G}_6 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 9 \end{bmatrix} \tag{5.3}$$

because the three selected sub-blocks are $\mathbf{f}_1 + \mathbf{f}_2 + \mathbf{f}_3$, $\mathbf{f}_1 + 2\mathbf{f}_2 + 3\mathbf{f}_3$, and $\mathbf{f}_1 + 4\mathbf{f}_2 + 9\mathbf{f}_3$.

Algorithm 2 Coded Elastic Computing for Matrix-Vector Multiplication

```
Input: The data matrix $\mathbf{X}$, the number of machines $P$, the recovery threshold $L$, the linear combination coefficients $g_{s,k}$'s in equation (5.1) and the sequence of input vectors $\mathbf{w}_t$ $t = 1, 2, \ldots$.
Preprocessing: Partition the data $\mathbf{X}$ into $L$ subsets and compute the coded subsets as in (5.1).
Online computation:
FOR each computation with input $\mathbf{w}_t$:
    Broadcast: The master node sends $\mathbf{w}_t$ to each worker.
    FOR each group index $j$:
        Gather: The $k$-th worker computes $\mathbf{u}_{t,k,j} = \mathbf{X}^{\text{coded}}_{k,j}\mathbf{w}_t$ and sends $\mathbf{u}_{t,k,j}$ to the master.
        The master gathers vectors $\mathbf{u}_{t,k,j}$ for all workers that use the $j$-th sub-block and obtains
        the matrix $\mathbf{u}_{t,j}$ which contains the results for the $j$-th group (row block).
        Decode: The master node computes $\mathbf{u}_{t,j}\mathbf{G}_j^{-1}$ to obtain the results for the $j$-th group.
    Output: The master node outputs $\mathbf{X}\mathbf{w}_t$.
    IF Preemption/New Machines: Change the selected data to use based on the current number of machines.
```

5.3.4 Analysis of coded elastic computing: achieving optimal fully transition compatibility

According to Definition 5.2.1, a fully compatible family of policies can support seamless transitions between any pairs of policies in the family. The coded elastic computing scheme provided in Algorithm 2 for matrix-vector multiplication gives a family of fully-compatible policies for fixed memory cost at each machine. In Section 5.3.4, we analyze the memory cost and the number of data points to be used at each existing machine. Then, in Section 5.3.4, we provide lower bounds on these two quantities and show that Algorithm 2 achieves the optimal memory cost and the size of the selected data. Thus, the coded computing policies in Algorithm 2 is a compatible family of *optimal* policies, in that for the fixed storage size, each policy obtains the largest number of tolerable failures and smallest size of the selected data to use (e^*,u^*), and provides seamless transitions between each other without moving data at existing machines.

Upper bounds on the storage cost and the size of the selected data to use

Suppose the original data has N data points and all data points are in $\mathbb{R}^d$. In the following theorem, by the *size* of the data we mean the number of data points.

Theorem 5.3.1. *The coded elastic computing algorithm achieves the exact computation result of $\mathbf{X}\mathbf{w}_t$ for all t. The size of the data stored at each machine is N/L. The size of the selected data to use at each machine is $N/(P-M)$ and is the same across different machines. The overall size of the selected data is the same as the size of the original data.*

Proof. See Appendix C.1. □

Theorem 5.3.1 shows that our technique uses the same size of data as the original (uncoded) case. This is desirable for memory-bound applications.

Remark 10. (Cost analysis) Recall that P is the number of workers, $N \times d$ is the size of the matrix $\mathbf{X}$, L is the recovery threshold, and $P-M$ is the number of currently available machines. The encoding (preprocessing step) is a one-time cost for online matrix-vector multiplications. The decoding by solving a linear system at the master node has computational cost $\mathcal{O}(LN)$, because the linear system for each group (row-block) of data involves L equations on L unknown subvectors of size $N/L/(P-M)$ (which is the height of each sub-block), and there are $P-M$ such groups. Thus, the computational cost using straightforward matrix-vector multiplication is $L^2 \cdot N/L/(P-M) \cdot (P-M) = NL$. The matrix-multiplication step at each worker has cost $\mathcal{O}(dN/(P-M))$. Thus, the decoding cost is smaller than the computational cost at each worker as long as $d = \Omega((P-M)L)$. Even if $d < \Omega((P-M)L)$, we can partition the machines into smaller groups and respectively code each group. One thing to note is that the decoding complexity, even for the straightforward matrix-vector multiplication method, is NL, independent of the number of workers P.

Lower bounds on the storage cost and the size of the selected data to use

Here, we provide a fundamental limit which shows that the achievable scheme provided in 5.3.1 is optimal in terms of the storage cost at each machine and the size of the actually used data at each iteration, for a fixed number of machines and a fixed number of tolerable machine preemptions. Before we present the theorem, we formalize the definition of the *size* of data using number of bits. This is because in theory, we cannot store arbitrarily high-precision numbers.

Assumption 1. Suppose the entries of the matrix $\mathbf{X}$ are i.i.d. random variables that take values in a finite set $\mathcal{S} \in \mathbb{R}$. Each of this random variable has entropy $H = \log|\mathcal{S}|$. Thus, the overall entropy of all the data in $\mathbf{X}$ is NdH.

Assumption 2. For a certain computation policy, suppose each machine initially stores an array of finite-precision numbers in its memory. Each finite-precision number can be an *arbitrary* function of the original data. The overall number of finite-precision numbers that is stored is finite.

Note that although we use real-number computation all the time, the numbers that we deal with are always discrete, i.e., we conduct computation of finite-precision numbers in the real field. This validates Assumption 1. Also note that the number of possible combinations of floating point numbers that can be stored by a finite-length bit array is finite. Therefore, Assumption 2 is also valid. We need these two assumptions because if they do not hold, the system can concatenate all the real numbers in $\mathbf{X}$ into one real number and only stores that particular real number. In that case, no bound on the storage is meaningful because one only needs to read this single number in memory to access all the information of $\mathbf{X}$. We also need the assumption that the size of the stored numbers are finite because otherwise, we can enumerate all possible $\mathbf{Xw}$ and store them.

Assumption 3. Suppose we do not alter the way that we store the data $\mathbf{X}$ inside the memory after the computation begins. Even when a preemption-type failure happens, we do not move the data at existing machines.

Assumption 4. By *selecting to use* one number stored in the memory, we mean the algorithm reads the whole number (e.g., reading all digits if the number is stored as a floating-point number) from the memory for further processing. The array of finite-precision numbers can only be accessed one number at a time, meaning that one cannot access a function value of two numbers and claim that only one number is selected to use.

Theorem 5.3.2. *Suppose the Assumptions 1-4 hold. Suppose we require the recovery of the exact computation result* $\mathbf{Xw}$*. Then, the following fundamental limits hold.*

(a) Denote the entropy of the encoded data at the k-th machine by $H_k, k = 1, 2, \ldots, P$. Then, to provide the tolerance to a maximum of $P-L$ failures, we have $\max_{k\in\{1,2,\ldots,P\}} H_k \geq \frac{N}{L}\cdot dH$;

(b) The worst-case entropy of the actually used data (maximized with respect to the choice of $\mathbf{w}$*) has to be no less than NdH, or $N/(P-M)\cdot dH$ at each machine.*

Proof. See Appendix C.2. □

Remark 11. By comparing Theorem 5.3.1 and Theroem 5.3.2, we see that the coded elastic computing technique in Algorithm 2 achieves the fundamental limit because each data point has dimension d and each entry has entropy H. We note a nuance here that the linear combinations in Algorithm 2 may make each encoded number have entropy larger than H. See [281] for more details.

Remark 12. Note that the claim (b) has to be stated in a worst-case way because for many choices of $\mathbf{w}$, computing $\mathbf{Xw}$ can be degenerated. For example, if we know in advance that $\mathbf{w}$ only takes value in a very small finite set of vectors, we can compute $\mathbf{Xw}$ for all possible $\mathbf{w}$ and store these vectors. When $\mathbf{w}$ is sparse, we also do not need to read the entire matrix $\mathbf{X}$. Therefore, we indeed need to state the fundamental lower bound in terms of the worst-case $\mathbf{w}$.

5.4 Extended results of the coded elastic computing

The cyclic way of elastic data partitioning applies to general coded computing techniques proposed thus far and is not limited to matrix-vector multiplications.

5.4.1 Matrix-matrix multiplications

First, we consider the application of matrix-matrix multiplications. We consider an *online* version: we store an encoded version of the matrix $\mathbf{A}$ at P machines and compute the matrix multiplication $\mathbf{AB}$ for different $\mathbf{B}$'s. We partition the matrix $\mathbf{A}$ column-wise and the matrix $\mathbf{B}$ row-wise, and stores linearly combined submatrices of $\mathbf{A}$ and $\mathbf{B}$ at each machine. In an online setting with elastic machine preemptions, we encode and store $\mathbf{A}$ at the initial stage, and do not move $\mathbf{A}$ anymore. When we receive $\mathbf{B}$, we still partition the matrix $\mathbf{B}$ row-wise but linearly combine them using the knowledge of the availability of the machines. The advantage is that we do not need to use the polynomial-based codes and can thus avoid possible numerical issues. At the same time, since the availability of the machines are known before we encode $\mathbf{B}$, we can also remove the factor of 2 in the recovery threshold of MatDot codes. We can also use the same computational time cost as the uncoded case, which is similar to what we can achieve in the matrix-vector case.

More precisely, suppose we parition the matrix $\mathbf{A}$ into L column blocks $\mathbf{A}_1, \dots, \mathbf{A}_L$, and encode these blocks into P blocks $\mathbf{A}_s^{\text{coded}}, s = 1, 2, \dots, P$, where P is the initial number of machines:

$$\mathbf{A}_s^{\text{coded}} = \sum_{k=1}^{L} g_{s,k} \mathbf{A}_k. \tag{5.4}$$

Denote by $\mathbf{G}$ the matrix $[g_{s,k}]$, which is of size $P \times L$. Assume M machines are preempted and $P - M \geq L$ machines remain. Similar to Algorithm 2, we partition each coded submatrix $\mathbf{A}_s^{\text{coded}}$ into $P - M$ submatrices. However, here, we partition each $\mathbf{A}_s^{\text{coded}}$ *column-wise* to $\mathbf{A}_{s,i}^{\text{coded}}, i = 1, 2, \dots, P - M$. This is mathematically equivalent to partitioning each uncoded submatrix $\mathbf{A}_s$ into $\mathbf{A}_{s,i}, i = 1, 2, \dots, P - M$, and compute

$$\mathbf{A}_{s,i}^{\text{coded}} = \sum_{k=1}^{L} g_{s,k} \mathbf{A}_{k,i}, i = 1, 2, \dots, P - M. \tag{5.5}$$

The subscript s in $\mathbf{A}_{s,i}$ belongs to a subset with size $P - M$ of the set $\{1, 2, \dots, P\}$, which corresponds to the $P - M$ available machines after the preemption failures. We again select to use these submatrices $\mathbf{A}_{s,i}$ in a cyclic fashion, just as shown in Figure 5.1. For example, consider the case when $P = 6$, $L = 3$, $M = 2$ and the 2nd and the 4th machines are preempted, which is exactly the same as shown in Figure 5.1(d). Then, we use $\mathbf{A}_{1,1}, \mathbf{A}_{1,2}, \mathbf{A}_{1,3}$ at the 1st machine, $\mathbf{A}_{3,2}, \mathbf{A}_{3,3}, \mathbf{A}_{3,4}$ at the 3rd machine, and so on. Denote by $\mathcal{S}_i$ the set of the indices of all the machines that use $\mathbf{A}_{s,i}$. For example, for $i = 1$, in the above example, $\mathcal{S}_1 = \{1, 5, 6\}$. Again, similar to Section 5.3.3, denote by $\mathbf{G}_i$ the submatrix of $\mathbf{G}$ with row indices in $\mathcal{S}_i$. Each $\mathbf{G}_i$ is a $L \times L$ matrix. Denote by

$$\mathbf{H}_i = (\mathbf{G}_i^{\top})^{-1}, \tag{5.6}$$

and assume $\mathbf{H}_i$ has element $h_{l,k}, l = 1, 2, \dots, L, k = 1, 2, \dots, L$.

When we get the matrix $\mathbf{B}$, we partition it row-wise into L blocks $\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_k$. Then, we partition each submatrix $\mathbf{B}_k$ into $P - M$ row blocks $\mathbf{B}_{k,i}, i = 1, 2, \dots, P - M$ as well. For each i, suppose $\mathcal{S}_i = \{s_{i,1}, s_{i,2}, \ldots, s_{i,L}\}$, where $1 \le s_{i,1} < s_{i,2} < \ldots < s_{i,L} \le P$. Then, we encode

$$\mathbf{B}^{\text{coded}}_{s_{i,l},i} = \sum_{k=1}^{L} h_{l,k} \mathbf{B}_{k,i}, \tag{5.7}$$

and send $\mathbf{B}^{\text{coded}}_{s_{i,l},i}$ to the $s_{i,l}$-th machine to compute $\mathbf{A}^{\text{coded}}_{s_{i,l},i}\mathbf{B}^{\text{coded}}_{s_{i,l},i}$.

When we do a reduction on all the partial results $\mathbf{A}^{\text{coded}}_{s_{i,l},i}\mathbf{B}^{\text{coded}}_{s_{i,l},i}$, we can indeed get $\mathbf{AB}$.

Lemma 5.4.1. *If we use the cyclic partitioning technique to determine each* $\mathcal{S}_i = \{s_{i,1}, s_{i,2}, \ldots, s_{i,L}\}$ *and encode* $\mathbf{B}$ *according to* (5.7)*, we have*

$$\sum_{i=1}^{P-M} \sum_{l=1}^{L} \mathbf{A}^{coded}_{s_{i,l},i}\mathbf{B}^{coded}_{s_{i,l},i} = \mathbf{AB}. \tag{5.8}$$

Proof. See the online report [281]. □

Remark 13. (Cost analysis) We can see that the computational cost for this algorithm comes from two parts: (1) the encoding of $\mathbf{B}$ using (5.7), and (2) the computational cost of $\mathbf{A}^{\text{coded}}_{s_{i,l},i}\mathbf{B}^{\text{coded}}_{s_{i,l},i}$. Assume the matrix $\mathbf{A}$ has size $d_A \times N$ and $\mathbf{B}$ has size $N \times d_B$, where $d_A, d_B = \Theta(N)$. Then, encoding $\mathbf{B}$ has complexity $(P-M)L \cdot L \cdot (d_B \times N/L/(P-M)) = Ld_BN = \Theta(LN^2)$. Computation of a single $\mathbf{A}^{\text{coded}}_{s_{i,l},i}\mathbf{B}^{\text{coded}}_{s_{i,l},i}$ has complexity $d_A \times N/L/(P-M) \times d_B$, and each machine computes L of them, which means the overall complexity is $d_A \times N/L/(P-M) \times d_B \times L = d_A d_B N/(P-M)$. Note that this is in the order of $\Theta(N^3)$ and the complexity is the same as distributing the matrix-matrix multiplication task to $P-M$ machines. The encoding time is much less than the computation time per worker if $Ld_BN \ll d_A d_B N/(P-M)$, or $L(P-M) \ll d_A$. If the encoding time is much smaller than the matrix-multiplication time, we again achieve linear scaling of the theoretical computational complexity in the number of available machines $P-M$ without moving data at the existing machines. At the same time, the computation time cost is the same at each worker machine as the uncoded case. Note that the scheme here considers a problem in which the system becomes aware of the indices of the failed machines after the preemptions have happened, and can adaptively change the encoding of $\mathbf{B}$.

5.4.2 Coded elastic computing for linear models

Then, we focus on the application of coded computing for linear regression. For the ease of presentation, we consider vanilla gradient descent (we also use line search in the experiment validation for all competing techniques), in which the full matrix $\mathbf{X}$ is used at each iteration. The technique developed here naturally generalizes to stochastic gradient descent and other generalized linear models such as logistic regression. Consider the linear objective function:

$$f(\mathbf{w}; \mathbf{X}, \mathbf{y}) = \sum_{i=1}^{n} (\mathbf{w}^\top \mathbf{x}_i - \mathbf{y}_i)^2 + h(\mathbf{w}). \tag{5.9}$$

The vanilla distributed gradient descent has the form $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{g}_t$ and $\mathbf{g}_t = \mathbf{X}^\top(\mathbf{X}\mathbf{w}_t - \mathbf{y}) + \partial_{\mathbf{w}} h(\mathbf{w}_t)$. When the data matrix $\mathbf{X}$ is large, the most time-consuming part is the computation of $\mathbf{X}^\top(\mathbf{X}\mathbf{w}_t - \mathbf{y})$. We thus extend Algorithm 2 in the following way to compute $\mathbf{X}^\top(\mathbf{X}\mathbf{w}_t - \mathbf{y})$. It is nothing but a combination of Algorithm 2 and the matrix-matrix multiplication algorithm in Section 5.4.1.

- Compute $\mathbf{X}^j\mathbf{w}_t$ (where $\mathbf{X}^j$ is the data in the j-th group, or the j-th row block in Figure 5.1 across different machines) for all group-index j in an elastic way using Algorithm 2;
- The master computes $\mathbf{z}_t^j = \mathbf{X}^j\mathbf{w}_t - \mathbf{y}^j$ for all group-index j, where $\mathbf{y}^j$ are the labels corresponding to the data points in the j-th group;
- The master re-encodes $\mathbf{z}_t^j$'s using the (pre-computed) inverse generator matrix $\mathbf{H}_j = (\mathbf{G}_j^{-1})^\top$ to obtain $(\mathbf{G}_j^{-1})^\top\mathbf{z}_t^j$, and scatters the results to the workers that use the j-th group of data;
- Since data at workers are encoded using $\mathbf{G}_j$, the reduced results from all the workers are

$$\sum_j (\mathbf{X}^j)^\top \mathbf{G}_j^\top (\mathbf{G}_j^{-1})^\top \mathbf{z}_t^j = \sum_j (\mathbf{X}^j)^\top(\mathbf{X}^j\mathbf{w}_t - \mathbf{y}^j) = \mathbf{X}^\top(\mathbf{X}\mathbf{w}_t - \mathbf{y}). \tag{5.10}$$

Note that in the above extension of Algorithm 2, the workers also utilize the data as in Figure 5.1. The experiment results of coded elastic computing in linear models are provided in Section 5.5.

5.4.3 Fully distributed coded elastic computing

Coded elastic computing is not restricted to a master-worker setting. In this section, we consider a fully distributed coded elastic matrix-vector multiplication technique that is a trivial generalization of Algorithm 2. The advantage of a fully distributed framework is that communication among workers can be overlapped and the communication to the master does not become the single bottleneck in the limit of a large number of machines. Consider an application of iterative matrix-vector multiplication

$$\mathbf{w}_{t+1} = f(\mathbf{X}\mathbf{w}_t), \tag{5.11}$$

where $\mathbf{X}$ is a square data matrix, and $f(\cdot)$ is an entry-wise low-complexity operation on the vector $\mathbf{X}\mathbf{w}_t$. We again consider the example shown in Figure 5.1,[2] but apart from the 6 worker machines, there is no master-node. In each iteration, each machine computes its own matrix-vector multiplication based on the selected data to use. For example, the 2nd machine computes $\mathbf{b}_2\mathbf{w}_t$, $\mathbf{c}_2\mathbf{w}_t$, $\mathbf{d}_2\mathbf{w}_t$, and stacks the results into one vector. Then, the results at each machine can be broadcast to all the other machines using an all-gather communication (using a bucketing algorithm [23]). The communication is also done in a cyclic way so that all the communications can be maximally overlapped

[2]Note that the example in Figure 5.1 partitions the data row-wise. The column-wise partition can use the elastic coding scheme in Section 5.4.1, which is essentially an elastic dot-product scheme.

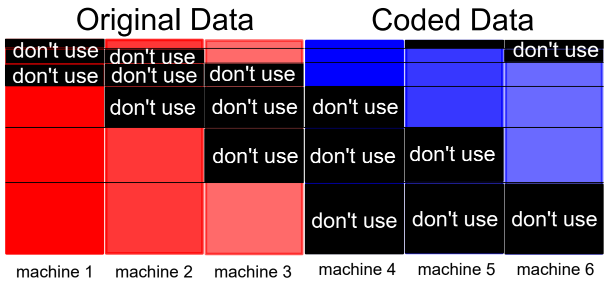


Figure 5.2: Non-uniform data partitioning

(i.e., all machines can communicate at the same time). In this way, the communication bandwidth can be reduced when compared to the master-worker framework. Moreover, the overall communication time in this scheme is the same as the uncoded scheme using the same number of machines, because the selected size of data at each machine is the same as the uncoded case. The decoding of the intermediate results at each iteration is done independently at each machine. The decoding can also be conducted distributedly for each group of data (each row-block), i.e., each machine only takes care of the decoding of each row-block. This can lead to increase on the communication time, but can significantly reduce the decoding time. Also, the smaller communication is possible at the cost of more storage or computation in the individual machines .

5.4.4 Uneven task partitioning in coded elastic computing

In distributed computing, heterogeneity can happen if some machines are predictably slower than other machines. In this case, the uniform partitioning on the stored data in Figure 5.1 becomes suboptimal. To partially address the heterogeneity problem, one can use the non-uniform cyclic partitioning on the data as shown in Figure 5.2. However, due to the insufficiency of the degree of freedom in this non-uniform partitioning scheme, it can only roughly approximate the computation capability, which is shown in the following.

Preliminary Results: To characterize this insufficiency precisely, we define *an achievable combination of computation rates* as an n-tuple $(x_1, x_2, \dots, x_n)$ such that the i-th machine in the n machines only uses x_i blocks of data, where $0 \leq x_i \leq 1$. Then, we have the following lemma.

Lemma 5.4.2. *The region of all achievable combinations of heterogeneous computation rates is characterized by the $(n-1)$-dimensional subset $\Omega = \{\mathbf{x} = (x_1, x_2, \dots, x_n) : \mathbf{x} = \mathbf{H}\mathbf{y}, 0 \leq y_i \leq 1, \mathbf{1}^\top \mathbf{y} = 1, \}$, where H is a cylic matrix in which the first row is $[1, 1, \dots, 1, 0, 0, \dots, 0]$ (the number of ones depends on the recovery threshold).*

However, this set does not coincide with the set of all possible computation rates, and hence the non-uniform data partitioning technique only partially addresses the problem of heterogeneous computation rates.

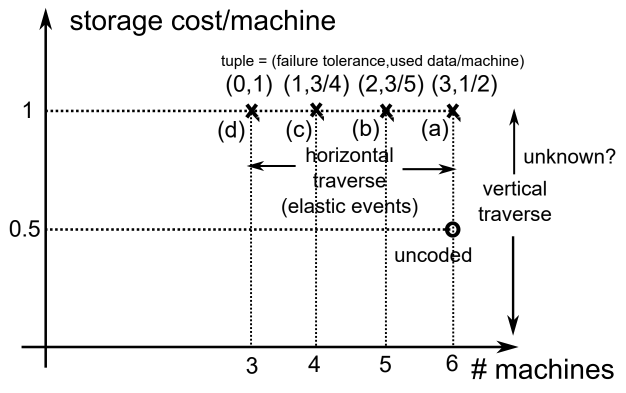


Figure 5.3: The chart of optimal policies under different configurations.

5.4.5 Towards fully transition compatibility: horizontal and vertical transition compatibility

Horizontal transition compatibility

We can represent the policies in Figure 5.1 as four configuration points (represented by crosses $\times$) in Figure 5.3. The x-axis is the number of machines, and the y-axis is the number of blocks of data stored at each machine. Each (x,y) point represents one configuration point. Each (t,u) tuple written at a point (x,y) means the optimal policy at this configuration point (x,y) can maximally tolerate t machine failures, while select to use only u blocks of data at each machine. The four crossed points, from left to right, correspond to the (d)(c)(b)(a) respectively in Figure 5.1. Therefore, our preliminary scheme [281] allows seamless movement in the horizontal direction between different Pareto-optimal points. Note that horizontal movement between configuration points results from elastic events (when existing machines leave and new machines join).

Vertical transition compatibility

Moving in the vertical direction in Figure 5.3 means that we add or remove data at each machine, but still achieve optimal fault tolerance and memory access. We analyze one example here. Again, assume that we have 6 machines and each machine stores one of the coded data blocks $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, $\mathbf{X}_1 + \mathbf{X}_2 + \mathbf{X}_3$, $\mathbf{X}_1 + 2\mathbf{X}_2 + 3\mathbf{X}_3$, and $\mathbf{X}_1 + 4\mathbf{X}_2 + 9\mathbf{X}_3$. Then, we can tolerate 3 failures while only accessing 0.5 blocks of data at each machine (Figure 5.1(b)). In the upper part of Figure 5.4, we show how to move vertically from the configuration point (6,1) to the configuration point (6, 5/6), i.e., removing 1/6 of data while fixing the number of machines to be 6. When 1/6 of data is removed, it can be shown that the maximum number of tolerable failures reduces from 3 to 2. Now, we show that the policy shown in Figure 5.4 can indeed tolerate 2 failures (see the lower part of Figure 5.4). Without loss of generality, suppose the 3rd and the 5th machines fail. Then, at each remaining machine, we use 3 small blocks of data, and for the two specific blocks that are marked by the green rectangles, we break each small block into four and

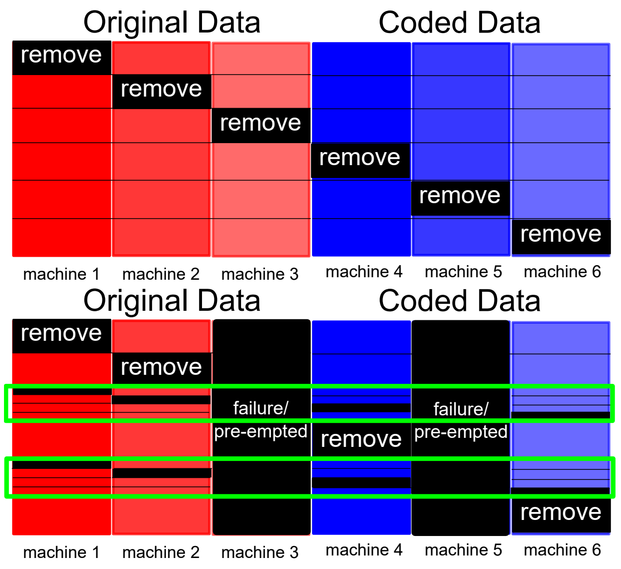


Figure 5.4: Remove data in a cyclic fashion to transit vertically on Figure 5.3.

only use three, again in a cyclic fashion. In this way, on each row, we have three small blocks, so we can recover the results. Further, each machine has the optimal memory access of 3/4 blocks of data, which can be checked by counting the non-black area on each column. Thus, this is a concrete example of moving vertically in Figure 5.3.

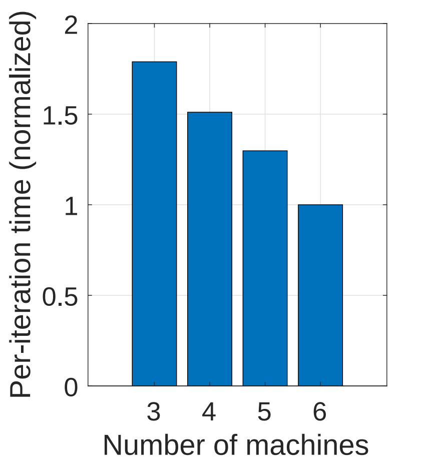
One future direction is to propose a general framework for all possible vertical transitions in Figure 5.3 and combine horizontal and vertical transitions to obtain seamless transitions between any pair of points. One problem to notice is that we cannot move up without moving data, because moving up requires increasing the data size. In this case, we would like to study **the minimum data movement to move upwards in the Pareto-optimal chart**.

5.5 Experimental evaluations

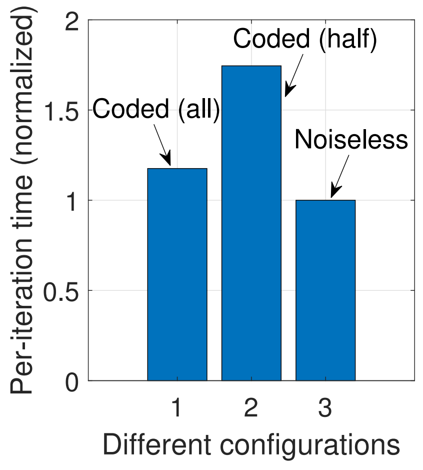
The proposed coded elastic computing technique has been implemented on top of Apache REEF [56] Elastic Group Communication (EGC) framework [3]. REEF EGC provides an API allowing to implement elastic computations by chaining fault-tolerant MPI-like primitives. In this chapter, we assess the performance of our elastic code computing approach through 2 mini-benchmarks.

Matrix-vector mini-benchmark. In this mini-benchmark, we test that indeed the time cost decreases linearly with the increase in the number of machines available. We mimic an elastic computing environment on Amazon EC2 by using different numbers of t2.large instances to compute the same matrix-vector product $\mathbf{X}\mathbf{w}$. The matrix is randomly generated and with size 30000×10000, and it is partitioned initially into 3 submatrices of size 10000×10000. Then, they are encoded into 6 submatrices of the same

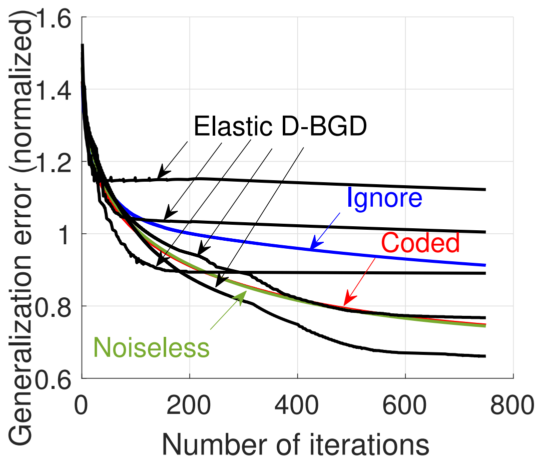
[3]`https://github.com/interesaaat/reef/tree/elastic-sync`



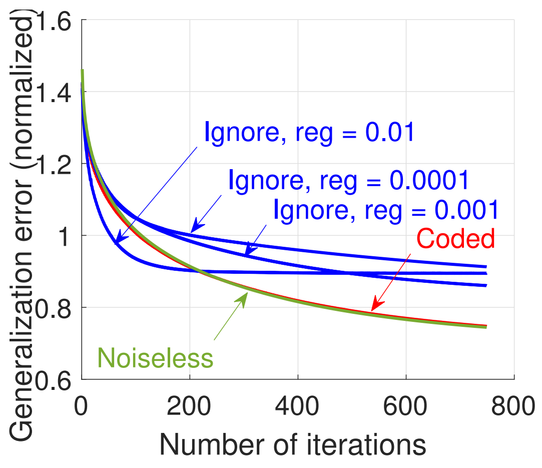
(a) Matrix-vector mini-benchmark



(b) Linear model overhead



(c) Linear model error (different methods)



(d) Linear model error (different regularization parameters)

Figure 5.5: Mini-benchmarks experiments (results normalized due to confidentiality).

size, and each submatrix is stored at one machine (for a total of 6 machines). To mimic the elastic events, we change the number of available machines by injecting artificial failures. The maximum number of failures is 3. The per-iteration overall time (including both communication and computation) is shown in Figure 5.5(a). The result is averaged using 20 independent trials. As we can see, the coded elastic computing technique can utilize the extra machines when the number of machines increases.

Linear model mini-benchmark. In this experiment, we test a coded implementation of linear regression using line-search-based batch gradient descent. We run the test over 20 machines on a Microsoft internal multi-tenancy cluster. Each data point in the dataset has 3352 features, and we sample 10000 data for training and 10000 data for testing. We generate random failures and allow REEF EGC to reschedule new machines when failures occur. We start with Figure 5.5(b) where we plot the time for each iteration. In theory, when all the workers are present, the computational cost per iteration should be the same as the uncoded case. However, the coded method (all) has slight overhead due to decoding cost. The coded method (half) shows the cost when only half of the workers are running, which is, as expected, twice the cost of the uncoded method. In Figure 5.5(c) and 5.5(d) we report the generalization error and we compare our coded elastic computing technique with three baselines, namely noiseless (no failure), ignore the failure and continue, and an existing algorithm called Elastic Distr-BGD. The coded method can achieve the same convergence behavior as the noiseless case, while the ignore method achieves worst generalization error, even with different regularization parameters. In Figure 5.5(d), we show 5 different experiments on Distr-BGD using the same failure probability but different realizations. The convergence of Distr-BGD depends on when a failure occurs and can lead to different algorithm performance. Note that Distr-BGD needs the assumption that failed machines are eventually recovered. If this assumption is not true, then, the algorithm can have different performance over different runs because the Distr-BGD keeps using previous gradient vectors at the failed machines, which can make the optimization miss the minimum point. In the plot of Distr-BGD, the *valley* part is due to overfitting, and the sudden change to near flat loss growth is because when the gradient descent has missed the optimal point of empirical training loss, the fixed gradient at the failure nodes makes the line search choose the smallest step size. In some cases, the Distr-BGD works extremely well because the fixed gradients act like momentum and can improve the speed of convergence. The performance of the *ignore* method can be improved using different regularization parameters. Thus, in Figure 5.5(c), we compare different regularization parameters, and observe again that the ignore method can have algorithmic level performance degradation.

From the experiment results, we can see that the coded elastic computing technique can obtain the same convergence behavior as ordinary gradient-descent-based algorithms but can elastically allocate the workload based on the number of available machines without moving data around.

5.6 Conclusions and future directions

The coded elastic computing technique presented in this chapter is designed for the new cloud offerings where machines can leave and join during the computation. Our framework handles the elastic events *positively*, meaning that when machines leave, it shifts the computation to the remaining workers without moving the data, and when new machines join the computation, it actively reduces the workload of existing machines in a seamless way. We provably show that the coded elastic computing technique can achieve the same memory-access cost as the noiseless case, and it achieves the theoretically optimal memory-access time and storage cost in matrix-vector multiplication problems. Therefore, the coded elastic computing technique is optimal for memory-bound applications when the memory-access time dominates the overall computation time. Using experiments in both Amazon EC2 and on a Microsoft multi-tenancy cluster, we show that the coded elastic computing technique can achieve the same convergence behavior as if no failure occurs, and can dynamically adjust working loads respect to the number of remaining workers. The proposed technique can be applied to coded matrix-vector, matrix-matrix multiplications and linear regression, and potentially other applications where the large-scale matrix operations are the bottleneck.

Chapter 6

Exploiting the multi-stage computing II: convergence viewed as additive error

6.1 Introduction

In this chapter, we present a coding-inspired technique to deal with the straggler effect in distributed computing of linear inverse problems using iterative solvers [217]. As we have mentioned at the beginning of Chapter 5, we will introduce techniques that explicitly utilize the properties of multi-stage computing. We focus on the standard iterative solvers for these linear systems. For example, we consider the *personalized PageRank* problem, and we study the power-iteration method which is the most classical PageRank algorithm [190]. The results in this chapter is a review of the paper [277].

Most algorithms in coded computation treat straggling workers as *erasures*. In other words, the computation results at the stragglers are discarded. Interestingly, in the iterative solvers for linear inverse problems, even if the computation result at a straggler has not converged, the algorithm presented in this chapter does not ignore the result but instead combines it (with appropriate weights) with results from other workers. This is in part because the result of the iterative method converges gradually to the correct solution.

We use a small example shown in Fig. 6.1 to illustrate this idea. Suppose we want to solve two linear inverse problems with solutions $\mathbf{x}_1^*$ and $\mathbf{x}_2^*$. If we use existing techniques, we can add an additional linear inverse problem, the solution of which is $\mathbf{x}_1^* + \mathbf{x}_2^*$ (see Section 6.3.1), and distribute these three problems to three workers. Using this method, the solutions $\mathbf{x}_1^*$ and $\mathbf{x}_2^*$ can be obtained from the results of any combination of two fast workers that are first to come close to their solutions. However, what if we have a computational deadline, T_{dl}, by which only one worker converges to its solution? In that case, existing coded-computation strategies will declare a computation failure because it needs at least two workers to respond. However, our strategy does not require convergence: even intermediate results from workers, as long as they are received, can be utilized to estimate solutions. In other words, our strategy degrades gracefully as the number of stragglers increases, or as the deadline is pulled earlier. The presented

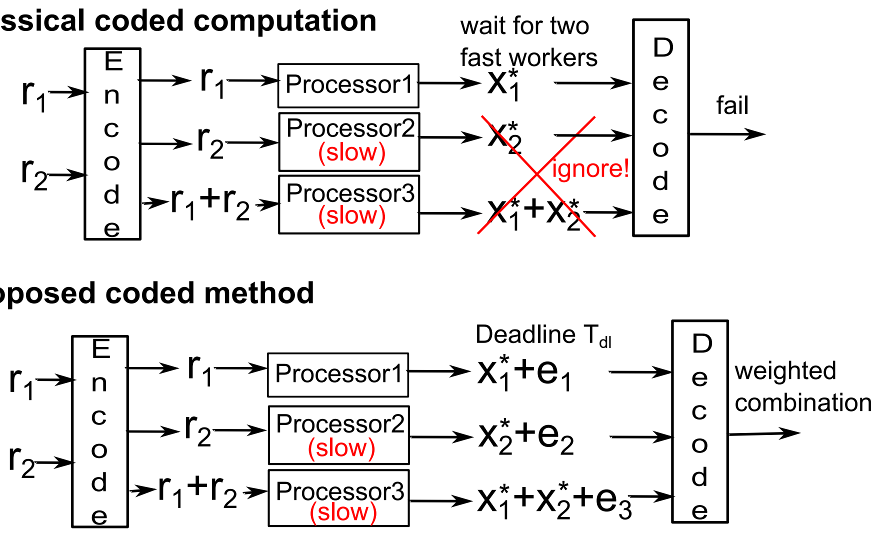


Figure 6.1: A toy example of the comparison between the existing coded computing scheme and the proposed algorithm.

algorithm essentially treats the difference from the optimal solution as "soft" additive noise.

Theoretically, we show that for a specified deadline time $T_{\rm dl}$, under certain conditions on worker speed, the coded linear inverse solver using structured codes has in scaling sense smaller mean squared error than the replication-based linear solver (Theorem 6.3.7). For validation of our theory, we performed experiments to compare the performance of coded and replication-based personalized PageRank (respectively using coded and replication-based power-iteration method) on the Twitter and Google Plus social networks under a deadline on computation time using a given number of workers on a real computation cluster (Section 6.4.1). We observe that the MSE of coded PageRank is smaller than that of replication by a factor of 10^4 at $T_{dl} = 2$ seconds.

The focus of this chapter is on utilizing computations to deliver minimal MSE in solving linear inverse problems. Our algorithm does not reduce communication cost. However, because each worker performs sophisticated iterative computations, such as the power-iteration computations in our problem, the time required for computation dominates that of communication (see Section 6.3.5).

Although our motivation comes from PageRank, the proposed technique applies more generally to the distributed computation of linear equations of the form $\mathbf{AX} = \mathbf{B}$ using iterative algorithms, and linear dynamical systems such as Navier-Stokes equations in fluid dynamics [35], in which case the dynamical system is modeled as iterative linear systems.

6.2 System model and problem formulation

In this section, we present the model of parallel linear systems that we will apply the idea of error correcting codes. Then, we provide two applications that can be directly formulated in the form of parallel linear systems. Note that, different from the previous chapters, we use P instead of N to denote the number of machines, because N is used to denote the size of the graph (similar to Chapter 4 and Chapter 8).

6.2.1 Solving parallel linear systems using iterative methods

Consider the problem of solving k inverse problems[1] with the same linear transform matrix and different inputs $\mathbf{r}_i$:

$$\mathbf{M}\mathbf{x}_i = \mathbf{r}_i, i = 1, 2, \dots k. \tag{6.1}$$

When $\mathbf{M}$ is a square matrix, the closed-form solution is

$$\mathbf{x}_i = \mathbf{M}^{-1}\mathbf{r}_i. \tag{6.2}$$

When $\mathbf{M}$ is a non-square matrix, the solution to (6.1) is interpreted as

$$\mathbf{x}_i = \arg\min \|\mathbf{M}\mathbf{x} - \mathbf{r}_i\|^2 + \lambda \|\mathbf{x}\|^2, i = 1, 2, \dots k, \tag{6.3}$$

with an appropriate regularization parameter λ. The closed-form solution of (6.3) is

$$\mathbf{x}_i = (\mathbf{M}^\top\mathbf{M} + \lambda\mathbf{I})^{-1}\mathbf{M}^\top\mathbf{r}_i. \tag{6.4}$$

Computing matrix inverse in (6.2) or (6.4) directly is often hard and commonly used methods are often iterative. In this chapter, we study two ordinary iterative methods, namely the Jacobian method for solving (6.1) and the gradient descent method for solving (6.3).

The Jacobian method for square system

For a square matrix $\mathbf{M}$, one can decompose $\mathbf{M} = \mathbf{D} + \mathbf{L}$, where $\mathbf{D}$ is diagonal. Then, the Jacobian iteration is written as

$$\mathbf{x}_i^{(l+1)} = \mathbf{D}^{-1}(\mathbf{r}_i - \mathbf{L}\mathbf{x}_i^{(l)}). \tag{6.5}$$

Under certain conditions of $\mathbf{D}$ and $\mathbf{L}$ (see [217, p.115]), the computation result converges to the true solution.

[1]Note that in this chapter, we consider solving k inverse problems instead of a single one. This way of partitioning the distributed computing problem can be used when we have a large amount of online requests and the computation capability of one machine is limited.

Gradient descent for non-square system

For the ℓ-2 minimization problem (6.3), the gradient descent solution has the form

$$\mathbf{x}_i^{(l+1)} = ((1-\lambda)\mathbf{I} - \epsilon\mathbf{M}^\top\mathbf{M})\mathbf{x}_i^{(l)} + \epsilon\mathbf{M}^\top\mathbf{r}_i, \tag{6.6}$$

where ϵ is an appropriate step-size.

General formulation

From these two problem formulations, we can see that the iterative methods have the same form

$$\mathbf{x}_i^{(l+1)} = \mathbf{B}\mathbf{x}_i^{(l)} + \mathbf{K}\mathbf{r}_i, i = 1, 2, \dots k, \tag{6.7}$$

for two appropriate matrices $\mathbf{B}$ and $\mathbf{K}$. Denote by $\mathbf{x}_i^*$ the solution to (6.1) or (6.3). Then,

$$\mathbf{x}_i^* = \mathbf{B}\mathbf{x}_i^* + \mathbf{K}\mathbf{r}_i, i = 1, 2, \dots k. \tag{6.8}$$

Then, from (6.8) and (6.7), the computation error $\mathbf{e}_i^{(l)} = \mathbf{x}_i^{(l)} - \mathbf{x}_i^*$ satisfies

$$\mathbf{e}_i^{(l+1)} = \mathbf{B}\mathbf{e}_i^{(l)}. \tag{6.9}$$

6.2.2 Distributed computing and the straggler effect

Consider solving k linear inverse problems in k parallel workers using the iterative method (6.7), such that each processor solves one problem. Due to the straggler effect, the computation of the linear inverse problem on different workers can have different computation speeds. Suppose after the deadline time T_{dl}, the i-th worker has completed l_i iterations in (6.7). Then, the residual error at the i-th worker is

$$\mathbf{e}_i^{(l_i)} = \mathbf{B}^{l_i}\mathbf{e}_i^{(0)}. \tag{6.10}$$

For our theoretical results, we sometimes need the following assumption.
Assumption 5. We assume that the optimal solutions $\mathbf{x}_i^*, i = 1, 2, \dots k$ are i.i.d. Denote by $\boldsymbol{\mu}_E$ and $\mathbf{C}_E$ respectively the mean and the covariance of each $\mathbf{x}_i^*$. Assume we start with the initial estimate $\mathbf{x}_i^{(0)} = \boldsymbol{\mu}_E$, which can be estimated from data. Then, $\mathbf{e}_i^{(0)} = \mathbf{x}_i^{(0)} - \mathbf{x}_i^*$ has mean $\mathbf{0}_N$ and covariance $\mathbf{C}_E$.

Note that Assumption 5 is equivalent to the assumption that the inputs $\mathbf{r}_i, i = 1, 2, \dots k$ are i.i.d., because the input and the true solution for a linear inverse problem are related by a linear transform. We provide an extension of this i.i.d. assumption in Section 6.3.3.

6.2.3 A motivating example

Now we study an example of the linear inverse problems.

PageRank as a square system

For a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$, the PageRank algorithm aims to measure the importance of the nodes in $\mathcal{V}$ by computing the stationary distribution of a discrete-time "random walk with restart" on the graph that mimics the behavior of a web surfer on the Internet. At each step, with probability $1-d$, the random walk chooses a random neighbor on the graph with uniform probability to proceed to (e.g. $d = 0.15$ in [190]). With probability d, it jumps to an arbitrary node in the graph. The probability d is often called the "teleport probability". From [190], the problem of computing the stationary distribution is equivalent to solving the following linear problem

$$\mathbf{x} = \frac{d}{N}\mathbf{1}_N + (1-d)\mathbf{A}\mathbf{x}, \tag{6.11}$$

where N is the number of nodes and $\mathbf{A}$ is the column-normalized adjacency matrix, i.e., for a directed edge $v_i \to v_j$, $\mathbf{A}_{ij} = \frac{1}{\text{deg}(v_j)}$, where $\text{deg}(v_j)$ is the in-degree of v_j.

The personalized PageRank problem [113] considers a more general linear equation

$$\mathbf{x} = d\mathbf{r} + (1-d)\mathbf{A}\mathbf{x}, \tag{6.12}$$

for any possible vector $\mathbf{r} \in \mathbb{R}^N$ that satisfies $\mathbf{1}^\top \mathbf{r} = 1$. Compared to the original PageRank problem [190], personalized PageRank [113] utilizes both the structure of the graph *and* the personal preferences of different users. The solution x is also the limiting distribution of a random walk, but with different restarting distribution. That is, with probability d, instead of jumping to each node with uniform probability, the random walk jumps to different nodes according to distribution $\mathbf{r}$. Intuitively, difference in the vector $\mathbf{r}$ represents different preferences of web surfers.

A classical method to solve PageRank is power-iteration, which iterates the following computation until convergence (usually with initial condition $\mathbf{x}_i^{(0)} = \frac{1}{N}\mathbf{1}_N$):

$$\mathbf{x}^{(l+1)} = d\mathbf{r} + (1-d)\mathbf{A}\mathbf{x}^{(l)}. \tag{6.13}$$

One can see that the power-iteration method is exactly the same as the Jacobian iteration (6.5).

Another example that we have not mentioned here is the problem of reconstructing *graph signals* using a non-square linear system [49, 50, 179, 218, 230]. More details can be found in our paper [277].

6.2.4 Preliminaries on error correcting codes

The encoding means multiplies the inputs to the parallel workers with a generator matrix $\mathbf{G}$ and the decoder multiplies the outputs of the workers with a decoding matrix $\mathbf{L}$. We use an (n, k) code where the generator matrix has size $k \times n$. In this chapter, we often use generator matrices $\mathbf{G}$ with orthonormal rows, which means

$$\mathbf{G}_{k\times n}\mathbf{G}_{n\times k}^\top = \mathbf{I}_k. \tag{6.14}$$

An example of such a matrix is the submatrix formed by any k rows of an $n \times n$ orthonormal matrix (e.g., a Fourier matrix). Under this assumption, $\mathbf{G}_{k\times n}$ can be augmented to form an $n \times n$ orthonormal matrix using another matrix $\mathbf{H}_{(n-k)\times n}$ (the parity check matrix), i.e. the square matrix $\mathbf{F}_{n\times n} = \begin{bmatrix} \mathbf{G}_{k\times n} \\ \mathbf{H}_{(n-k)\times n} \end{bmatrix}$ satisfies $\mathbf{F}^\top \mathbf{F} = I_n$. This structure assumption is not necessary when we compare the error exponents of different linear inverse algorithms when the computation time T_{dl} goes to infinity (e.g., coded and uncoded). However, it is useful when we present theorems on finite T_{dl}.

6.3 Main result: coded distributed computing of linear inverse problems

6.3.1 The coded linear inverse algorithm

The proposed coded linear inverse algorithm is shown in Algorithm 3 and illustrated in Figure 6.2. The algorithm has three stages: preprocessing (encoding) at the central controller, parallel computing at n parallel workers, and post-processing (decoding) at the central controller. As we show later in the analysis of computing error, the entries in the diagonal matrix $\mathbf{\Lambda}$ are the expected mean-squared error at each worker prior to decoding. The decoding matrix $\mathbf{L}_{k\times n}$ in the decoding step (6.18) is chosen to be $(\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\mathbf{G}\mathbf{\Lambda}^{-1}$ to reduce the mean-squared error of the estimates of linear inverse solutions by assigning different weights to different workers based on the estimated accuracy of their computation (which is what $\mathbf{\Lambda}$ provides).

Remark 14. This choice of $\mathbf{\Lambda}$ is very similar to the weighted least-squares solution. However, the analysis of the algorithm is different from the weighted least-squares solution in that the "samples" already become correlated after the linear encoding. Therefore, the results presented in this chapter can also be viewed as generalized weighted least-squares with correlated data samples.

6.3.2 Bounds on performance of the coded linear inverse algorithm

Define $\mathbf{l} = [l_1, l_2, \dots l_n]$ as the vector of the number of iterations at all workers. We use the notation $\mathbb{E}[\cdot|\mathbf{l}]$ to denote the conditional expectation taken with respect to the randomness of the optimal solution $\mathbf{x}_i^*$ (see Assumption 5) but conditioned on fixed iteration number l_i at each worker, i.e., for a random variable X,

$$\mathbb{E}[X|\mathbf{l}] = \mathbb{E}[X|l_1, l_2, \dots l_n]. \tag{6.21}$$

Theorem 6.3.1. *Define $\mathbf{E} = \hat{\mathbf{X}} - \mathbf{X}^*$, i.e., the error of the decoding result* (6.18). *Assuming that the solutions for each linear inverse problem are chosen i.i.d. (across all problems) according to a distribution with covariance $\mathbf{C}_E$. Then, the error covariance of $\mathbf{E}$ satisfies*

$$\mathbb{E}[\|\mathbf{E}\|^2 |\mathbf{l}] \le \sigma_{max}(\mathbf{G}^\top\mathbf{G}) trace\left[(\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\right], \tag{6.22}$$

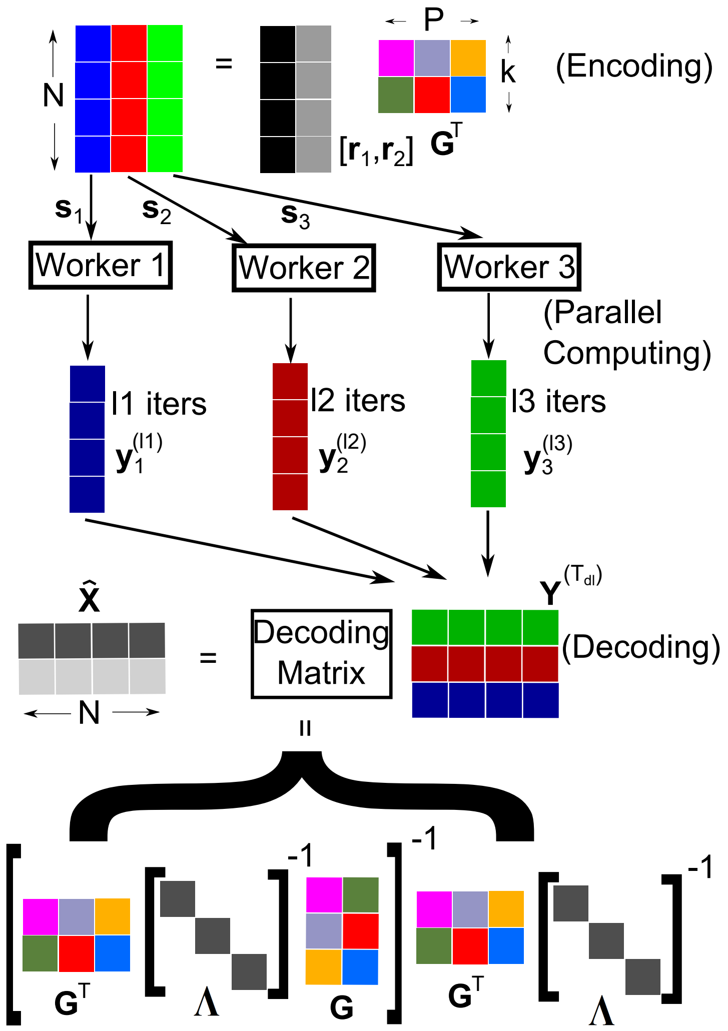


Figure 6.2: Illustration of Algorithm 3

Algorithm 3 Coded Distributed Linear Inverse

```
Input: Input vectors $[\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_k]$, generator matrix $\mathbf{G}_{k\times n}$, the linear system matrices
$\mathbf{B}$ and $\mathbf{K}$ defined in (6.7).
Initialize (Encoding): Encode the input vectors $[\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_k]$ and the initial estimates
by multiplying with the generator matrix $\mathbf{G}$:
    $[\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n] = [\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_k] \cdot \mathbf{G}.$    (6.15)
    $[\mathbf{y}_1^{(0)}, \mathbf{y}_2^{(0)}, \ldots, \mathbf{y}_n^{(0)}] = [\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)}, \ldots, \mathbf{x}_k^{(0)}] \cdot \mathbf{G}.$    (6.16)
Parallel Computing:
for $i = 1$ to $n$ do
    Send $\mathbf{s}_i$ and $\mathbf{y}_i^{(0)}$ to the $i$-th worker. Compute the solution of (6.1) or (6.3) using the
    specified iterative method (6.7) with initial estimate $\mathbf{y}_i^{(0)}$ at each worker in parallel
    until a deadline $T_{\text{dl}}$.
end for
After $T_{\text{dl}}$, collect all linear inverse results $\mathbf{y}_i^{(l_i)}$ from these $n$ workers. The superscript $l_i$
in $\mathbf{y}_i^{(l_i)}$ represents that the $i$-th worker finished $l_i$ iterations within time $T_{\text{dl}}$. Denote by
$\mathbf{Y}^{(T_{\text{dl}})}$ the collection of all results
    $\mathbf{Y}_{N\times n}^{(T_{\text{dl}})} = [\mathbf{y}_1^{(l_1)}, \mathbf{y}_2^{(l_2)}, \ldots, \mathbf{y}_n^{(l_n)}].$    (6.17)
Post Processing (Decoding):
Compute an estimate of the linear inverse solutions using the following matrix multi-
plication:
    $\hat{\mathbf{X}}^\top = \mathbf{L} \cdot (\mathbf{Y}^{(T_{\text{dl}})})^\top := (\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\mathbf{G}\mathbf{\Lambda}^{-1}(\mathbf{Y}^{(T_{\text{dl}})})^\top,$    (6.18)
where the estimate $\hat{\mathbf{X}}_{N\times k} = [\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_k]$, the matrix $\mathbf{\Lambda}$ is
    $\mathbf{\Lambda} = \text{diag}\left[\text{trace}(\mathbf{C}(l_1)), \ldots, \text{trace}(\mathbf{C}(l_n))\right],$    (6.19)
where the matrices $\mathbf{C}(l_i), i = 1, \ldots, n$ are defined as
    $\mathbf{C}(l_i) = \mathbf{B}^{l_i}\mathbf{C}_E(\mathbf{B}^\top)^{l_i}.$    (6.20)
In computation of $\mathbf{\Lambda}$, if trace$(\mathbf{C}(l_i))$ are not available, one can use estimates of this
trace as discussed in Section 6.3.5.
```

where the norm $\|\cdot\|$ *is the Frobenius norm,* $\sigma_{max}(\mathbf{G}^\top\mathbf{G})$ *is the maximum eigenvalue of* $\mathbf{G}^\top\mathbf{G}$ *and the matrix* $\mathbf{\Lambda}$ *is defined in* (6.19). *Further, when* $\mathbf{G}$ *has orthonormal rows,*

$$\mathbb{E}[\|\mathbf{E}\|^2 \,|\mathbf{l}] \leq \textit{trace}\left[(\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\right], \tag{6.23}$$

Proof. See appendix Section D.1 for a detailed proof. □

Corollary 6.3.2. *Suppose the i.i.d. Assumption 5 holds and the matrix* $\mathbf{G}_{k\times n}$ *is a submatrix of an* $n \times n$ *orthonormal matrix, i.e. there exists a matrix* $\mathbf{F}_{n\times n} = \begin{bmatrix} \mathbf{G}_{k\times n} \\ \mathbf{H}_{(n-k)\times n} \end{bmatrix}$ *satisfies* $\mathbf{F}^\top \mathbf{F} = \mathbf{I}_n$. *Assume that the symmetric matrix* $\mathbf{F}\mathbf{\Lambda}\mathbf{F}^\top$ *has the block form*

$$\mathbf{F}\mathbf{\Lambda}\mathbf{F}^\top = \begin{bmatrix} \mathbf{J}_1 & \mathbf{J}_2 \\ \mathbf{J}_2^\top & \mathbf{J}_4 \end{bmatrix}_{n\times n}, \tag{6.24}$$

that is, $(\mathbf{J}_1)_{k\times k}$ *is* $\mathbf{G}\mathbf{\Lambda}\mathbf{G}^\top$, $(\mathbf{J}_2)_{k\times(n-k)}$ *is* $\mathbf{G}\mathbf{\Lambda}\mathbf{H}^\top$, *and* $(\mathbf{J}_4)_{(n-k)\times(n-k)}$ *is* $\mathbf{H}\mathbf{\Lambda}\mathbf{H}^\top$. *Then, we have*

$$\mathbb{E}[\|\mathbf{E}\|^2 \,|\mathbf{l}] \le \mathit{trace}(\mathbf{J}_1) - \mathit{trace}(\mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top). \tag{6.25}$$

Proof. The proof essentially relies on the Schur complement. See appendix Section D.2 for details. □

6.3.3 Bounds on the mean-squared error beyond the i.i.d. case

Until now, we based our analysis on the i.i.d. assumption 5. For the PageRank problem discussed in Section 6.2.3, this assumption means that the PageRank queries are independent across different users. Although the case when the PageRank queries are arbitrarily correlated is hard to analyze, we may still provide concrete analysis for some specific cases. For example, a reasonable case when the PageRank queries are correlated with each other is when these queries are all affected by some "common fashion topic" that the users wish to search for. In mathematics, we can model this phenomenon by assuming that the solutions to the i-th linear inverse problem satisfies

$$\mathbf{x}_i^* = \bar{\mathbf{x}} + \mathbf{z}_i, \tag{6.26}$$

for some random vector $\bar{\mathbf{x}}$ and an i.i.d. vector $\mathbf{z}_i$ across different queries (different i). The common part $\bar{\mathbf{x}}$ is random because the common fashion topic itself can be random. This model can be generalized to the following "stationary" model.

Assumption 6. Assume the solutions $\mathbf{x}_i^*$'s of the linear inverse problems have the same mean $\boldsymbol{\mu}_E$ and stationary covariances, i.e.,

$$\mathbb{E}[\mathbf{x}_i^*(\mathbf{x}_i^*)^\top] = \mathbf{C}_E + \mathbf{C}_{\text{Cor}}, \forall 1 \le i \le k, \tag{6.27}$$

$$\mathbb{E}[\mathbf{x}_i^*(\mathbf{x}_j^*)^\top] = \mathbf{C}_{\text{Cor}}, \forall 1 \le i, j \le k. \tag{6.28}$$

Under this assumption, we have to change the coded linear inverse algorithm slightly. The details are shown in Algorithm 4.

For the stationary version, we can have the counterpart of Theorem 6.3.1 as follows. Trivial generalizations include arbitrary linear scaling $\mathbf{x}_i^* = \alpha_i\bar{\mathbf{x}} + \beta_i\mathbf{z}_i$ for scaling constants α_i and β_i.

Algorithm 4 Coded Distributed Linear Inverse (Stationary Inputs)

Call Algorithm 3 but replace the $\mathbf{\Lambda}$ matrix with

$$\tilde{\mathbf{\Lambda}} = \sigma_{\max}(\mathbf{G}^\top\mathbf{G})\mathbf{\Lambda} + \text{diag}\{\mathbf{G}^\top\mathbf{1}_k\} \cdot \mathbf{\Psi} \cdot \text{diag}\{\mathbf{G}^\top\mathbf{1}_k\}^\top, \tag{6.29}$$

where $\sigma_{\max}(\mathbf{G}^\top\mathbf{G})$ is the maximum eigenvalue of $\mathbf{G}^\top\mathbf{G}$, and $\mathbf{\Psi}_{n\times n} = [\Psi_{i,j}]$ satisfies

$$\Psi_{i,j} = \text{trace}[\mathbf{B}^{l_i}\mathbf{C}_{\text{cor}}(\mathbf{B}^\top)^{l_j}]. \tag{6.30}$$

Theorem 6.3.3. *Define $\mathbf{E} = \hat{\mathbf{X}} - \mathbf{X}^*$, i.e., the error of the decoding result* (6.18) *by replacing $\mathbf{\Lambda}$ defined in* (6.19) *with $\tilde{\mathbf{\Lambda}}$ in* (6.29). *Assuming that the solutions for all linear inverse problems satisfy Assumption 6. Then, the error covariance of $\mathbf{E}$ satisfies*

$$\mathbb{E}[\|\mathbf{E}\|^2 \,|\mathbf{l}] \leq trace\left[(\mathbf{G}\tilde{\mathbf{\Lambda}}^{-1}\mathbf{G}^\top)^{-1}\right]. \tag{6.31}$$

where the norm $\|\cdot\|$ is the Frobenius norm.

Proof. See [277] for the complete proof. □

In Section 6.3.4, we compare coded, uncoded and replication-based linear inverse schemes under the i.i.d. assumption. However, we include one experiment in Section 6.4.1 to show that Algorithm 4 also works in the stationary case.

6.3.4 Comparison with cncoded schemes and replication-based schemes

Here, we often assume (we will state explicitly in the theorem) that the number of iterations l_i at different workers are i.i.d.. $\mathbb{E}_f[\cdot]$ denotes expectation on randomness of both the linear inverse solutions $\mathbf{x}_i^*$ and the number of iterations l_i.

Assumption 7. Within time T_{dl}, the number of iterations of linear inverse computations at each worker follows an i.i.d. distribution $l_i \sim f(l)$.

Comparison between the coded and uncoded linear inverse before a deadline

First, we compare the coded linear inverse scheme with an uncoded scheme, in which case we use the first k workers to solve k linear inverse problems in (6.8) without coding. The following theorem quantifies the overall mean-squared error of the uncoded scheme given $l_1, l_2, \ldots, l_k$. The proof is in appendix Section D.3.

Theorem 6.3.4. *In the uncoded scheme, the overall error is*

$$\mathbb{E}\left[\|\mathbf{E}_{uncoded}\|^2 \,|\boldsymbol{l}\right] = \mathbb{E}\left[\left\|[\mathbf{e}_1^{(l_1)}, \mathbf{e}_2^{(l_2)} \ldots, \mathbf{e}_k^{(l_k)}]\right\|^2 \middle| \mathbf{l}\right] = \sum_{i=1}^{k} trace\left(\mathbf{C}(l_i)\right). \tag{6.32}$$

Further, when the i.i.d. Assumption 7 holds,

$$\mathbb{E}_f\left[\|\mathbf{E}_{uncoded}\|^2\right] = k\mathbb{E}_f[trace(\mathbf{C}(l_1))]. \tag{6.33}$$

Then, we compare the overall mean-squared error of coded and uncoded linear inverse algorithms. **Note that this comparison is not fair** because the coded algorithm uses more workers than uncoded. However, we still include Theorem 6.3.5 because we need it for the fair comparison between coded and replication-based linear inverse.

Theorem 6.3.5. *(Coded linear inverse beats uncoded) Suppose the i.i.d. Assumption 5 and 7 hold and suppose* $\mathbf{G}$ *is a* $k \times n$ *submatrix of an* $n \times n$ *Fourier transform matrix* $\mathbf{F}$*. Then, expected error of the coded linear inverse is strictly less than that of uncoded:*

$$\mathbb{E}_f\left[\|\mathbf{E}_{uncoded}\|^2\right] - \mathbb{E}_f\left[\|\mathbf{E}_{coded}\|^2\right] \geq \mathbb{E}_f[trace(\mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top)], \tag{6.34}$$

where $\mathbf{J}_2$ *and* $\mathbf{J}_4$ *are defined in* (6.24).

Proof. From Corollary 6.3.2, for fixed $l_i, 1 \leq i \leq n$,

$$\mathbb{E}[\|\mathbf{E}_{\text{coded}}\|^2 \,|\mathbf{l}] \leq \text{trace}(\mathbf{J}_1) - \text{trace}(\mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top). \tag{6.35}$$

We will show that

$$\mathbb{E}_f[\text{trace}(\mathbf{J}_1)] = \mathbb{E}_f\left[\|\mathbf{E}_{\text{uncoded}}\|^2\right], \tag{6.36}$$

which completes the proof. To show (6.36), first note that from (6.33),

$$\mathbb{E}_f\left[\|\mathbf{E}_{\text{uncoded}}\|^2\right] = k\mathbb{E}_f[\text{trace}(\mathbf{C}(l_1))]. \tag{6.37}$$

Since $\mathbf{G} := [g_{j,i}]$ is a submatrix of a Fourier matrix, we have $|g_{ji}|^2 = 1/n$. Thus, $\mathbf{J}_1 = \mathbf{G}\mathbf{\Lambda}\mathbf{G}^\top$ satisfies

$$\text{trace}(\mathbf{J}_1) = \sum_{j=1}^{k}\sum_{i=1}^{n}|g_{ji}|^2\text{trace}(\mathbf{C}(l_i)) = \frac{k}{n}\sum_{i=1}^{n}\text{trace}(\mathbf{C}(l_i)).$$

Therefore,

$$\mathbb{E}_f[\text{trace}(\mathbf{J}_1)] = k\mathbb{E}_f[\text{trace}(\mathbf{C}(l_1))]. \tag{6.38}$$

which, along with (6.37), completes the proof of (6.36), and hence also the proof of Theorem 6.3.5. □

Comparison between the replication-based and coded linear inverse before a deadline

Consider an alternative way of doing linear inverse using $n > k$ workers. In this chapter, we only consider the case when $n-k < k$, i.e., the number of extra workers is only slightly bigger than the number of problems (both in theory and in experiments). Since we have $n-k$ extra workers, a natural way is to pick any $(n-k)$ linear inverse problems and replicate them using these extra $(n-k)$ workers. After we obtain two computation results for the same equation, we use two natural "decoding" strategies for this replication-based linear inverse: (i) choose the worker with higher number of iterations; (ii) compute the weighted average using weights $\frac{w_1}{w_1+w_2}$ and $\frac{w_2}{w_1+w_2}$, where $w_1 = 1/\sqrt{\text{trace}(\mathbf{C}(l_1))}$ and $w_2 = 1/\sqrt{\text{trace}(\mathbf{C}(l_2))}$, and l_1 and l_2 are the number of iterations completed at the two workers.

Theorem 6.3.6. *The replication-based schemes satisfies the following lower bound on the mean-squared error:*

$$\mathbb{E}_f\left[\left\|\mathbf{E}_{rep}\right\|^2\right] > \mathbb{E}_f\left[\|\mathbf{E}_{uncoded}\|^2\right] - (n-k)\mathbb{E}_f[trace(\mathbf{C}(l_1))]. \tag{6.39}$$

Proof overview. Here the goal is to obtain a lower bound on the MSE of replication-based linear inverse and compare it with an upper bound on the MSE of coded linear inverse.

Note that if an extra worker is used to replicate the computation at the i-th worker, i.e., the linear inverse problem with input $\mathbf{r}_i$ is solved on two workers, the expected error of the result of the i-th problem could at best reduced from $\mathbb{E}_f[\text{trace}(\mathbf{C}(l_1))]$ to zero[2]. Therefore, $(n-k)$ extra workers make the error decrease by at most (and strictly smaller than) $(n-k)\mathbb{E}_f[\text{trace}(\mathbf{C}(l_1))]$. □

Using this lower bound, we can provably show that coded linear inverse beats replication-based linear inverse when certain conditions are satisfied. One crucial condition is that the distribution of the random variable $\text{trace}(\mathbf{C}(l))$ satisfies a "variance heavy-tail" property defined as follows.

Definition: The random variable $\text{trace}(\mathbf{C}(l))$ is said to have a "ρ-variance heavy-tail" property if

$$\text{var}_f[\text{trace}(\mathbf{C}(l))] > \rho\mathbb{E}_f^2[\text{trace}(\mathbf{C}(l))], \tag{6.40}$$

for some constant $\rho > 1$. For the coded linear inverse, we will use a Fourier code the generator matrix $\mathbf{G}$ of which is a submatrix of a Fourier matrix. This particular choice of code is only for ease of analysis in comparing coded linear inverse and replication-based linear inverse. In practice, the code that minimizes mean-squared error should be chosen.

Theorem 6.3.7. *(Coded linear inverse beats replication) Suppose the i.i.d. Assumption 5 and 7 hold and $\mathbf{G}$ is a $k \times n$ submatrix of an $n \times n$ Fourier matrix $\mathbf{F}$. Further, suppose $(n-k) = o(\sqrt{n})$. Then, the expected error of the coded linear inverse satisfies*

$$\lim_{n\to\infty}\frac{1}{n-k}\left[\mathbb{E}_f\left[\|\mathbf{E}_{uncoded}\|^2\right] - \mathbb{E}_f\left[\|\mathbf{E}_{coded}\|^2\right]\right] \geq \frac{var_f[trace(\mathbf{C}(l_1))]}{\mathbb{E}_f[trace(\mathbf{C}(l_1))]}. \tag{6.41}$$

Moreover, if the random variable $trace(\mathbf{C}(l))$ satisfies the ρ-variance heavy-tail property for $\rho > 1$, coded linear inverse outperforms replication-based linear inverse in the following sense,

$$\begin{aligned}&\lim_{n\to\infty}\frac{1}{(n-k)}\left[\mathbb{E}_f\left[\|\mathbf{E}_{uncoded}\|^2\right] - \mathbb{E}_f\left[\left\|\mathbf{E}_{rep}\right\|^2\right]\right]\\ &< \frac{1}{\rho}\lim_{n\to\infty}\frac{1}{(n-k)}\left[\mathbb{E}_f\left[\|\mathbf{E}_{uncoded}\|^2\right] - \mathbb{E}_f\left[\|\mathbf{E}_{coded}\|^2\right]\right].\end{aligned} \tag{6.42}$$

Proof overview. See appendix Section D.3.1 for a complete and rigorous proof. □

[2]Although this is clearly a loose bound, it makes for convenient comparison with coded linear inverse

Asymptotic comparison between coded, uncoded and replication-based linear inverse as the deadline $T_{\text{dl}} \to \infty$

Consider the coded and uncoded linear inverse when the overall computation time $T_{dl} \to \infty$. From Theorem 6.3.1 and Theorem 6.3.4, the computation error of uncoded and coded linear inverse are respectively

$$\mathbb{E}\left[\|\mathbf{E}_{\text{uncoded}}\|^2 |\mathbf{l}\right] = \sum_{i=1}^{k} \text{trace}\left(\mathbf{C}(l_i)\right), \tag{6.43}$$

$$\mathbb{E}[\|\mathbf{E}_{\text{coded}}\|^2 |\mathbf{l}] \leq \sigma_{\max}(\mathbf{G}^\top\mathbf{G})\text{trace}\left[(\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\right], \tag{6.44}$$

where the matrix $\mathbf{\Lambda}$ is

$$\mathbf{\Lambda} = \text{diag}\left[\text{trace}(\mathbf{C}(l_1)), \dots, \text{trace}(\mathbf{C}(l_n))\right], \tag{6.45}$$

and the matrices $\mathbf{C}(l_i), i = 1, \dots, n$ are defined as

$$\mathbf{C}(l_i) = \mathbf{B}^{l_i}\mathbf{C}_E(\mathbf{B}^\top)^{l_i}. \tag{6.46}$$

Assumption 8. We assume the computation time of one power iteration is fixed at each worker for each linear inverse computation, i.e., there exist n random variables $v_1, v_2, \dots v_n$ such that $l_i = \lceil \frac{T_{\text{dl}}}{v_i} \rceil, i = 1, 2, \dots n$.

The k-th order statistic of a statistic sample is equal to its k-th smallest value. Suppose the order statistics of the sequence $v_1, v_2, \dots v_n$ are $v_{i_1} < v_{i_2} < \dots v_{i_n}$, where $\{i_1, i_2, \dots i_n\}$ is a permutation of $\{1, 2, \dots n\}$. Denote by $[k]$ the set $\{1, 2, \dots k\}$ and $[n]$ the set $\{1, 2, \dots n\}$.
Theorem 6.3.8. *(Error exponent comparison when $T_{dl} \to \infty$) Suppose the i.i.d. Assumption 5 and Assumption 8 hold. Suppose $n - k < k$. Then, the error exponents of the coded and uncoded computation schemes satisfy*

$$\lim_{T_{dl}\to\infty, l_i = \lceil \frac{T_{dl}}{v_i} \rceil} -\frac{1}{T_{dl}} \log \mathbb{E}[\|\mathbf{E}_{coded}\|^2 |\mathbf{l}] \geq \frac{2}{v_{i_k}} \log \frac{1}{1-d}, \tag{6.47}$$

$$\begin{aligned} &\lim_{T_{dl}\to\infty, l_i = \lceil \frac{T_{dl}}{v_i} \rceil} -\frac{1}{T_{dl}} \log \mathbb{E}[\|\mathbf{E}_{uncoded}\|^2 |\mathbf{l}] \\ =&\lim_{T_{dl}\to\infty, l_i = \lceil \frac{T_{dl}}{v_i} \rceil} -\frac{1}{T_{dl}} \log \mathbb{E}[\left\|\mathbf{E}_{rep}\right\|^2 |\mathbf{l}] = \frac{2}{\max_{i\in[k]} v_i} \log \frac{1}{1-d}, \end{aligned} \tag{6.48}$$

The error exponents of uncoded, replication and coded linear inverse satisfy coded > replication=uncoded.

Here the expectation $\mathbb{E}[\cdot|\mathbf{l}]$ is only taken with respect to the randomness of the linear inverse sequence $\mathbf{x}_i, i = 1, 2, \dots k$, and conditioned on the number of iterations $\mathbf{l}$. The limit $\lim_{T_{dl}\to\infty}$ is taken under the Assumption 8, i.e., $l_i = \lceil \frac{T_{dl}}{v_i} \rceil$.

Proof. See appendix F in [277] for a detailed proof. □

6.3.5 Analyzing the computational complexity

Encoding and decoding complexity

We first show that the encoding and decoding complexity of Algorithm 3 are in scaling-sense smaller than that of the computation at each worker. This is important to ensure that straggling comes from the parallel workers, not the encoder or decoder. The proof of the following Theorem can be found in [277].

Theorem 6.3.9. *The computational complexity for the encoding and decoding is $\Theta(nkN)$, where N is the number of rows in the matrix $\mathbf{B}$ and k, n depend on the number of available workers assuming that each worker performs a single linear inverse computation. For a general dense matrix $\mathbf{B}$, the computational complexity of computing linear inverse at each worker is $\Theta(N^2 l)$, where l is the number of iterations in the specified iterative algorithm. The complexity of encoding and decoding is smaller than that of the computation at each user for large $\mathbf{B}$ matrices (large N).*

The complexity of encoding and decoding can be further reduced if the Coppersmith-Winograd algorithm for matrix-matrix multiplication is used [57]. In our experiment on the Google Plus graph for computing PageRank, the computation time at each worker is 30 seconds and the encoding and decoding time at the central controller is about 1 second.

Computing the matrix $\boldsymbol{\Lambda}$

One difficulty in our coded linear inverse algorithm is computing the entries trace$(\mathbf{C}(l)) =$ trace $\left(\mathbf{B}^l \mathbf{C}_E (\mathbf{B}^\top)^l\right)$ in the weight matrix $\boldsymbol{\Lambda}$ in (6.19), which involves a number of matrix-matrix multiplications. One way to side-step this problem is to estimate trace$(\mathbf{C}(l))$ using Monte Carlo simulations. Concretely, choose m i.i.d. N-variate random vectors $\mathbf{a}_1, \mathbf{a}_2, \dots \mathbf{a}_m$ that are distributed the same as the initial error $\mathbf{e}^{(0)}$ after Assumption 5. Then, compute the statistic

$$\hat{\gamma}_{m,l} = \frac{1}{m} \sum_{j=1}^{m} \left\| \mathbf{B}^l \mathbf{a}_j \right\|^2, l = 1, 2, \dots T_u, \tag{6.49}$$

where T_u is an upper bound of the number of iterations in a practical iterative computing algorithm. The lemma in appendix Section D.3.2 shows that $\hat{\gamma}_{m,l}$ is an unbiased and asymptotically consistent estimator of trace$(\mathbf{C}(l))$ for all l. In our experiments on PageRank, for each graph we choose $m = 10$ and estimate trace$(\mathbf{C}(l))$ before implementing the coded linear inverse algorithm (in this case it is the coded power-iteration algorithm), which has the same complexity as solving $m = 10$ extra linear inverse problems.

For the correlated case, we have to compute a slightly modified weighting matrix denoted by $\tilde{\boldsymbol{\Lambda}}$ in (6.29). The only change is that we have to compute $\Psi_{i,j}$ in (6.30) for all possible l_i, l_j such that $1 \leq l_i, l_j \leq T_u$. We also choose m i.i.d. N-variate random vectors $\mathbf{b}_1, \mathbf{b}_2, \dots \mathbf{b}_m$ that are distributed with mean $\mathbf{0}_N$ and covariance $\mathbf{C}_{\text{cor}}$, which is the same as the correlation part according to Assumption 6. Then, compute the statistic

$$\hat{\gamma}_{m,(l_i,l_j)} = \frac{1}{m} \sum_{u=1}^{m} \mathbf{b}_u \mathbf{B}^{l_j} \mathbf{B}^{l_i} \mathbf{b}_u, 1 \leq l_i, l_j \leq T_u. \tag{6.50}$$

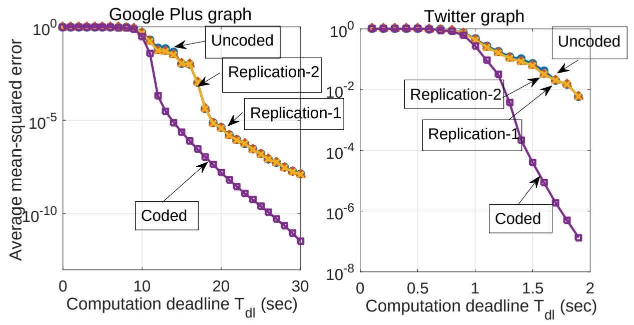


Figure 6.3: Experimentally computed overall mean squared error of uncoded, replication-based and coded personalized PageRank on the Twitter graph and Google Plus graph on a cluseter with 120 workers. The ratio of MSE for repetition-based schemes and coded PageRank increase as T_{dl} increases.

Then, the lemma in appendix Section D.3.2 shows that $\hat{\gamma}_{m,(l_i,l_j)}$ is also an unbiased and asymptotically consistent estimator of $\Psi_{i,j}$.

Analysis on the cost of communication versus computation

In this chapter, we focus on optimizing the computation cost. However, what if the computation cost is small compared to the overall cost, including the communication cost? If this is true, optimizing the computation cost is not very useful. In what follows, we show that the computation cost is larger than the communication cost in the scaling-sense.

Theorem 6.3.10. *The ratio between computation and communication at the i-th worker is $COST_{computation}/COST_{communication} = \Theta(l_i\bar{d})$ operations per integer, where l_i is the number of iterations at the i-th worker, and $\bar{d}$ is the average number of non-zeros in each row of the $\mathbf{B}$ matrix. The proof can be found in [277].*

6.4 Experiments and simulations

6.4.1 Experiments on real systems

We test the performance of the coded linear inverse algorithm for the PageRank problem on the Twitter graph and the Google Plus graph from the SNAP datasets [153]. The Twitter graph has 81,306 nodes and 1,768,149 edges, and the Google Plus graph has 107,614 nodes and 13,673,453 edges. We use the HT-condor framework in a cluster to conduct the experiments. The task is to solve $k = 100$ personalized PageRank problems in parallel using $n = 120$ workers. The uncoded algorithm picks the first k workers and uses one worker for each PageRank problem. The two replication-based schemes replicate the computation of the first $n-k$ PageRank problems in the extra $n-k$ workers (see Section 6.3.4). The coded PageRank uses n workers to solve these $k = 100$ equations using Algorithm 3. We use a $(120, 100)$ code where the generator matrix is the submatrix

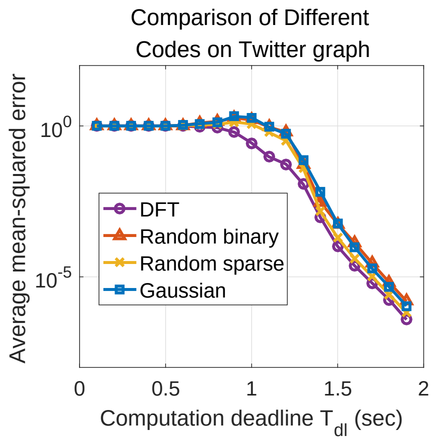


Figure 6.4: Experimental comparison of four different codes on the Twitter graph. In this experiment the DFT-code out-performs the other candidates in mean squared error.

composed of the first 100 rows in a 120×120 DFT matrix. The computation results are shown in Fig. 6.3. Note that the two graphs of different sizes so the computation in the two experiments takes different time. From Fig. 6.3, we can see that the mean-squared error of uncoded and replication-based schemes is larger than that of coded computation by a factor of 10^4.

We also compare Algorithm 3 with existing coded computing schemes (see the paper [277] for details). The classical coded computing scheme is not designed for iterative algorithms, but it has a natural extension to the case of computing before a deadline. This extension uses the results from the k fastest workers to retrieve the required PageRank solutions. More concretely, suppose $\mathcal{S} \subset [n]$ is the index set of the k fastest workers. Then, this extension retrieves the solutions to the original k PageRank problems by solving the following equation:

$$\mathbf{Y}_{\mathcal{S}} = [\mathbf{x}_1^*, \mathbf{x}_2^*, \ldots, \mathbf{x}_k^*] \cdot \mathbf{G}_{\mathcal{S}}, \tag{6.51}$$

where $\mathbf{Y}_{\mathcal{S}}$ is the computation results obtained from the fastest k workers and $\mathbf{G}_{\mathcal{S}}$ is the $k \times k$ submatrix composed of the columns in the generator matrix $\mathbf{G}$ with indexes in $\mathcal{S}$. However, since there is some remaining error at each worker (i.e., the computation results $\mathbf{Y}_{\mathcal{S}}$ have not converged yet), when conducting the matrix-inverse-based decoding, the error is magnified due to the large condition number of $\mathbf{G}_{\mathcal{S}}$. This is why existing coded computing schemes cannot be naively applied in the coded PageRank problem.

Finally, we test Algorithm 4 for correlated PageRank queries that are distributed with the stationary covariance matrix in the form of (6.27) and (6.28). Note that the only change to be made in this case is on the $\mathbf{\Lambda}$ matrix (see equation (6.29)). The other settings are exactly the same as the experiments that are shown in Figure 6.3. The results on the Twitter social graph are shown in Figure 6.4. In this case, we also have to compute

One question remains: what is the best code design for the coded linear inverse algorithm? Although we do not have a concrete answer to this question, we have tested different codes (with different generator matrices $\mathbf{G}$) in the Twitter graph experiment, all using Algorithm 3. The results are shown in Fig. 6.4. The generator matrix used for

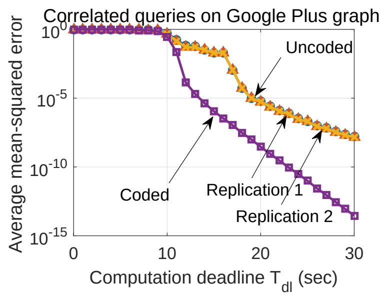


Figure 6.5: Experimentally computed overall mean squared error of uncoded, replication-based and coded personalized PageRank on the Twitter graph on a cluster with 120 workers. The queries are generated using the model from the stationary model in Assumption 6.

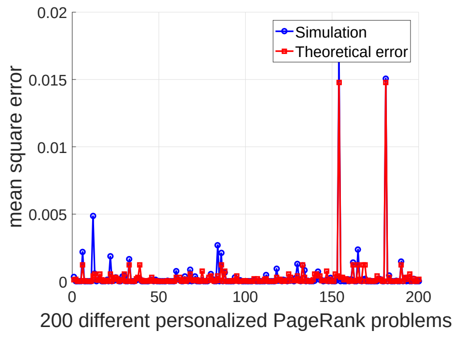


Figure 6.6: This simulation result shows the mean squared error of the computation results for $k = 200$ different problems in the uncoded scheme.

the "binary" curve has i.i.d. binary entries in $\{-1, 1\}$. The generator matrix used for the "sparse" curve has random binary sparse entries. The generator matrix for the "Gaussian" curve has i.i.d. standard Gaussian entries. In this experiment, the DFT-code performs the best. However, finding the best code in general is a meaningful future work.

6.4.2 Simulations

We also test the coded PageRank algorithm in a simulated setup with randomly generated graphs and worker response times. These simulations help us understand looseness in our theoretical bounding techniques. They can also test the performance of the coded Algorithm for different distributions. We simulate Algorithm 3 on a randomly generated Erdös-Rényi graph with $N = 500$ nodes and connection probability 0.1. The number of workers n is set to be 240 and the number of PageRank vectors k is set to be 200. We use the first $k = 200$ rows of a 240×240 DFT-matrix as the $\mathbf{G}$ matrix in the coded PageRank algorithm in Section 6.3.1. In Fig. 6.6 and Fig. 6.7, we show the simulation result on the mean squared error of all $k = 200$ PageRank vectors in both uncoded and coded PageRank, which are respectively shown in Fig. 6.6 and Fig. 6.7. The x-axis represents

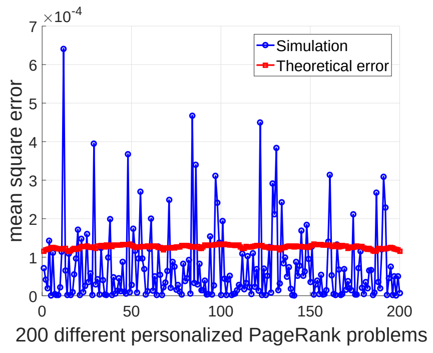


Figure 6.7: This simulation result shows the mean squared error of the computation results for $k = 200$ different problems in the coded scheme.

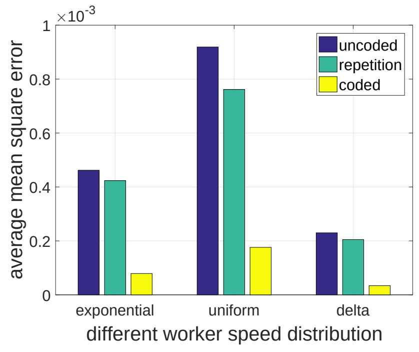


Figure 6.8: This figure shows the mean squared error of uncoded, replication-based and coded PageRank algorithms.

the computation results for different PageRank problems and the y-axis represents the corresponding mean-squared error. It can be seen that in the uncoded PageRank, some of the PageRank vectors have much higher error than the remaining ones (the blue spikes in Fig. 6.6), because these are the PageRank vectors returned by the slow workers in the simulation. However, in coded PageRank, the peak-to-average ratio of mean squared error is much lower than in the uncoded PageRank. This means that using coding, we are able to mitigate the straggler effect and achieve more uniform performance across different PageRank computations. From a practical perspective, this means that we can provide fairness to different PageRank queries.

We compare the average mean-squared error of uncoded, replication-based and coded PageRank algorithms in Fig. 6.8. The first simulation compares these three algorithms when the processing time of one iteration of PageRank computation is exponentially distributed, and the second and third when the number of iterations is uniformly distributed in the range from 1 to 20 and Bernoulli distributed at two points 5 and 20 (which we call "delta" distribution). It can be seen that in all three different types of distributions, coded PageRank beats the other two algorithms.

6.5 Conclusions and future directions

In this chapter, we study the problem of distributedly solving several linear systems. By studying coding for iterative algorithms designed for distributed inverse problems, we aim to identify novel properties of iterative and multi-stage computation systems and analytical tools for the design of coded computing algorithms.

Since iterative algorithms designed for inverse problems commonly have decreasing error with time, the partial computation results at stragglers can provide useful information for the distributed computing of the final outputs. By incorporating these partial results, we show improvement on the convergence time and error reduction compared to treating the results as erasures. In Chapter 7, we study more general settings of iterative computation, and extend the analytical framework developed in this chapter to the setting when the data matrix has to be partitioned and stored distributedly.

Chapter 7

Exploiting the multi-stage computing III: combining consecutive iterations in iterative computing

7.1 Introduction

Following Chapter 6, we continue to study the problem of coded multi-stage computation. Apart from iterative algorithms for linear systems and gradient methods, we also consider spectral analysis on large and sparse matrices in this chapter, such as computing eigenvectors and the singular value decomposition. A typical example of iterative computing is the power-iteration method that repeatedly multiplies the intermediate result with the data matrix until convergence.

In Chapter 6, we considered coded computing for power iterations when many inverse problem instances of the form $\mathbf{M}\mathbf{x}_i = \mathbf{b}_i, i = 1, 2, \ldots, m$ are computed in parallel. However, a more common setting is the computation of a *single* inverse problem $\mathbf{M}\mathbf{x} = \mathbf{b}$ when the linear system matrix (or data matrix) $\mathbf{M}$ is large and sparse and thus cannot fit in the memory of a single machine. Existing results in coded computing typically use dense generator matrices, such as those of MDS codes, to ensure good error-correcting capability. However, dense encoding of the sparse matrix $\mathbf{M}$ can significantly increase the number of non-zero entries, and hence can increase communication, storage and computational costs. In fact, dense encoding using MDS codes makes the sparse problem completely non-sparse. In this chapter, we use codes with sparse generator matrices instead (low-density generator matrices, or LDGM). However, despite the bad error-correcting ability of LDGM codes, we show that LDGM codes can be made surprisingly efficient in maintaining the convergence rate of iterative computing. The key is to use a novel decoding algorithm that we call "substitute decoding". The results in this chapter has appeared in the paper [280] and will be updated in the online report [272].

Substitute decoding is designed specifically for iterative computing problems. It relies on the fact that the intermediate result in the iterative computing $\mathbf{x}_{t+1} = f(\mathbf{x}_t)$ gradually converges to the fixed point or the optimum point, so $\mathbf{x}_{t+1}$ and $\mathbf{x}_t$ gradually become close to each other when t increases. Using this property, the substitute decoding method works by extracting the largest amount of available information from partial coded results in the computation of $\mathbf{x}_{t+1} = f(\mathbf{x}_t)$, and substituting the complementary unknown information by the available side information $\mathbf{x}_t$ from the previous step. In other words, instead of computing the exact result of $\mathbf{x}_{t+1} = f(\mathbf{x}_t)$, we compute a combined version $\mathbf{x}_{t+1} = \text{Proj}_1[f(\mathbf{x}_t)] + \text{Proj}_2[\mathbf{x}_t]$, where Proj_1 represents the projection onto the space of available information from partial decoding, and Proj_2 represents the orthogonal projection of Proj_1. This is useful in coded computing with LDGM codes because even if the exact result is not available (due to the insufficient error-correcting capability of LDGM codes), we can obtain a partial result and use the side information $\mathbf{x}_t$ to compensate the information loss caused by failures and the insufficiency of LDGM codes. As we show in our main theorem (Theorem 7.3.1), substitute decoding can multiply the error with a constant δ that drops to 0 when the partial generator matrix is close to full-rank. More specifically, δ is linear in the rank of the partial generator matrix formed by the linear combinations from non-erased workers, and is exactly 0 when the partial generator matrix is full-rank. The convergence rate of noiseless computation can be achieved when δ is small. This property is essential in using LDGM codes for coding sparse data because the partial generator matrix is often not full-rank but close to full-rank, which makes Proj_1 approximately equal to the identity projection, and makes $\mathbf{x}_{t+1} = \text{Proj}_1[f(\mathbf{x}_t)] + \text{Proj}_2[\mathbf{x}_t]$ close to $f(\mathbf{x}_t)$. As we show in our simulations (see Section 7.5), even for sparse generator matrices with only 2 non-zero entries in each row, coded iterative computing works significantly better than replication-based or uncoded iterative computing in convergence rate when the results from a constant fraction of workers are erased. When there are 3 non-zeros in each row, the convergence rate of noiseless iterative computing (the information limit) can be approximately achieved by coded computing.

In the first part of the chapter (see Section 7.2 to Section 7.3), we use PageRank as a simple example of iterative computing and introduce the substitute decoding method for different types of data splitting, e.g., row-wise splitting and column-wise splitting. In the second part of the chapter, we show that substitute decoding can be applied to more iterative computing problems beyond PageRank and linear systems. For example, we show that substitute decoding can make the orthogonal-iteration method robust to erasure-type failures, which can be applied to eigen-decomposition and the truncated singular value decomposition. We also show that substitute decoding can be applied to the computation of gradient descent with or without sparse data and can improve on existing techniques when an extremely sparse encoding matrix is used.

7.2 System model and problem formulation

Similar to Chapter 6, we use the PageRank problem as an example to introduce substitute decoding and relegate the discussion on more iterative computing tasks to Section 7.4. We consider solving $\mathbf{x} = d\mathbf{r} + (1-d)\mathbf{A}\mathbf{x}$ using the power-iteration method, which iterates $\mathbf{x}_{t+1} = d\mathbf{r} + (1-d)\mathbf{A}\mathbf{x}_t$ for $t = 0, 1, 2, \ldots$ until convergence (see Chapter 6). If we define

$\mathbf{B} = (1-d)\mathbf{A}$ and $\mathbf{y} = d\mathbf{r}$. Then, we have:

$$\mathbf{x}_{t+1} = \mathbf{y} + \mathbf{B}\mathbf{x}_t. \tag{7.1}$$

$\mathbf{x}_t$ converges to the true solution $\mathbf{x}^*$ if and only if spectral radius $\rho(\mathbf{B}) < 1$. For PageRank, $\rho(\mathbf{B}) = 1 - d < 1$ and $\mathbf{x}_t$ always converges to $\mathbf{x}^*$.

7.2.1 Noiseless distributed computing of power iterations

When the size of the linear system matrix $\mathbf{B}$ is too large to fit in the memory of a single machine, the computation of (7.1) is performed distributedly. The most straightforward way is to partition $\mathbf{B}$ into several blocks and store them in the memory of several workers. Denote the number of workers by P, and this is also the number of blocks. We describe three types of data splitting, namely row-wise splitting, column-wise splitting and 2D splitting (i.e., both row and column). The 2D matrix splitting is widely used for matrix multiplications [248] to overlap the communication cost[1].

Row-wise splitting

We split the matrix $\mathbf{B}$ into several row blocks. At the beginning of the t-th iteration, a master node sends the current result $\mathbf{x}_t$ to all workers. Then, the worker that has the row block $\mathbf{B}_i$ computes $\mathbf{B}_i\mathbf{x}_t$ and sends it back to the master node. At the end of the iteration, the master node concatenates all the results $\mathbf{B}_i\mathbf{x}_t, i = 1, 2, \ldots, P$ from the P workers to obtain $\mathbf{B}\mathbf{x}_t$, and computes $\mathbf{x}_{t+1} = \mathbf{B}\mathbf{x}_t + \mathbf{y}$.

Column-wise splitting

We split the linear system matrix $\mathbf{B}$ into several column blocks and also breaks the current result $\mathbf{x}_t$ into P subvectors of the same length N/P. Then, the i-th worker computes $\mathbf{B}_i\mathbf{x}_t^i$, and the master node adds up all the results.

2D splitting

In 2D splitting, we split the linear system matrix $\mathbf{B}$ both row-wise and column-wise into $\sqrt{P} \times \sqrt{P}$ blocks $\mathbf{B}_{ij}, i = 1, \ldots, \sqrt{P}, j = 1, \ldots, \sqrt{P}$, and each worker in a 2D mech holds a submatrix. We also break the current result $\mathbf{x}_t$ into $\sqrt{P}$ subvectors of the same length $N/\sqrt{P}$ and sends each subvector $\mathbf{x}_t^j$ to the j-th column of workers. The (i, j)-th worker computes $\mathbf{B}_{ij}\mathbf{x}_t^j$. Finally, the master node reduces on each row of workers.

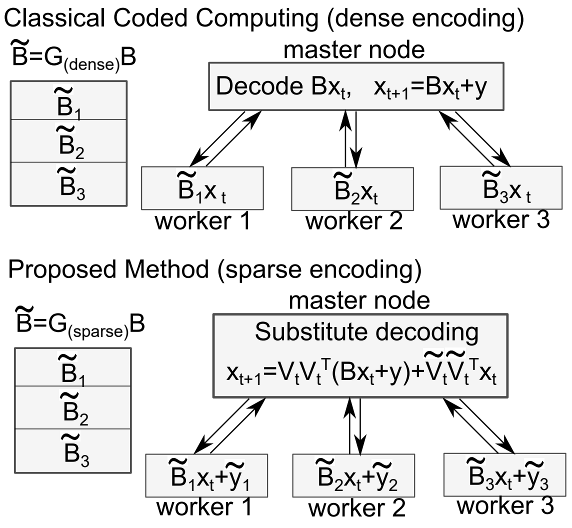

Classical Coded Computing (dense encoding)
$\widetilde{B}=G_{(dense)}B$
$\widetilde{B}_1$
$\widetilde{B}_2$
$\widetilde{B}_3$
master node
Decode Bx_t, $x_{t+1}=Bx_t+y$
$\widetilde{B}_1x_t$
worker 1
$\widetilde{B}_2x_t$
worker 2
$\widetilde{B}_3x_t$
worker 3
Proposed Method (sparse encoding)
$\widetilde{B}=G_{(sparse)}B$
$\widetilde{B}_1$
$\widetilde{B}_2$
$\widetilde{B}_3$
master node
Substitute decoding
$x_{t+1}=V_tV_t^T(Bx_t+y)+\widetilde{V}_t\widetilde{V}_t^Tx_t$
$\widetilde{B}_1x_t+\widetilde{y}_1$
worker 1
$\widetilde{B}_2x_t+\widetilde{y}_2$
worker 2
$\widetilde{B}_3x_t+\widetilde{y}_3$
worker 3

Figure 7.1: This shows the comparison between the existing works on coded computing and the proposed coded computing technique for the computation of power iterations in the row-splitting case. In the proposed method, we only show the scalar version as mentioned in Remark 15.

7.2.2 Preliminaries and notation on coded computing

We first present the direct application of coded computing to the power iteration (7.1) with row-wise splitting, and point out a drawback of it. As shown in the upper part of Fig. 7.1, if the common idea of coded computing is applied, $\mathbf{B}_{N\times N}$ is partitioned into k row blocks and linearly combined into $P > k$ row blocks $\widetilde{\mathbf{B}}_i, i = 1, 2, \ldots, P$ using a (P, k) linear code with a generator matrix $\mathbf{G}$ of size $P \times k$. Each encoded block is stored at one of the P workers. Denote the number of rows in each row block by $b = \frac{N}{k}$. Then, encoding can be written as

$$\widetilde{\mathbf{B}} = (\mathbf{G}_{P\times k} \otimes \mathbf{I}_b)\mathbf{B}, \tag{7.2}$$

where we use the Kronecker product because we encode row blocks.

At each iteration, the i-th worker computes $\widetilde{\mathbf{B}}_i\mathbf{x}_t$ and the master node concatenates all the results to obtain $\widetilde{\mathbf{B}}\mathbf{x}_t$, which is a coded version of $\mathbf{B}\mathbf{x}_t$. We define two operations (shown in Fig. 7.2) to simply the notation and analysis.

Definition: (block-wise operation) Denote by $\mathbf{v} = \text{vec}\,(\mathbf{X})$ the operation to vectorize the matrix $\mathbf{X}$ into the concatenation of its transposed rows, and denote by $\mathbf{X} = \text{mat}\,(\mathbf{v})$ the operation to partition the column vector $\mathbf{v}$ into small vectors and stack the transposed small vectors into the rows of $\mathbf{X}$. We always partition the vector $\mathbf{v}$ into smaller ones of length $b = \frac{N}{k}$ which represents the length of a single row-block at each worker. Then, it is straightforward to show that any operation of the form $\mathbf{x} = (\mathbf{A} \otimes \mathbf{I}_b) \cdot \mathbf{v}$ for

[1]The results in this chapter assumes the existence of a master node, which may not be true in general 2D splitting [248]. However, the results can be generalized to the fully-distributed setting, because the only assumption that we need for substitute decoding is that intermediate results gradually converge.

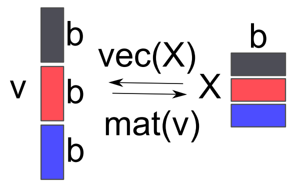


Figure 7.2: An illustration on the vec $(\cdot)$ and the mat $(\cdot)$ operations.

an arbitrary matrix $\mathbf{A}$ and an arbitrary vector $\mathbf{v}$ can be rewritten in a compact form $\mathbf{x} = \text{vec}\left(\mathbf{A}\text{mat}\left(\mathbf{v}\right)\right)$. Therefore, from (7.2), the obtained results at the master node is

$$\widetilde{\mathbf{B}}\mathbf{x} = (\mathbf{G}_{P\times k} \otimes \mathbf{I}_b)\mathbf{B}\mathbf{x} = \text{vec}\left(\mathbf{G}\text{mat}\left(\mathbf{B}\mathbf{x}\right)\right). \tag{7.3}$$

The matrix-version of $\widetilde{\mathbf{B}}\mathbf{x}$ is hence $\mathbf{G}\text{mat}\left(\mathbf{B}\mathbf{x}\right)$, which means each column of the matrix-version of $\widetilde{\mathbf{B}}\mathbf{x}$ is a codeword.

In the presence of stragglers or erasures, the coded results $\mathbf{G}\text{mat}\left(\mathbf{B}\mathbf{x}\right)$ would lose some of its rows, and the decoding can be done in a parallel fashion on each column. There are b columns in $\mathbf{G}\text{mat}\left(\mathbf{B}\mathbf{x}\right)$. Thus, the decoding complexity is $b\mathcal{N}_{\text{dec}}$, where $\mathcal{N}_{\text{dec}}$ is the complexity of decoding a single codeword.

A main drawback of the above method is that the generator matrix $\mathbf{G}$ is usually dense, such as MDS codes or random Gaussian codes. However, in practice, the system matrix $\mathbf{B}$ is often sparse, as in the PageRank problem. Therefore, using a dense $\mathbf{G}$ may significantly increase the number of non-zeros of the sparse linear system matrix. For example, if the generator matrix $\mathbf{G}$ has 20 non-zeros in each row, it means that the submatrices $\widetilde{\mathbf{B}}_i$ stored at each worker can have (in the worst case) 20 times larger size than the uncoded case if the matrix $\mathbf{B}$ is sparse. In this chapter, we show that $\mathbf{G}$ can actually be very sparse (such as two ones in each row), while the iterative computing (such as power iterations) can still remain robust to a linear number of stragglers in P.

7.2.3 Preliminaries on the proposed technique

For the clarification purposes, in this chapter, we focus on presenting the algorithm for the row-wise splitting. Readers are referred to the online document [272] for more details on column-wise splitting and 2D splitting. In our technique for row-wise splitting, similar to standard coded computing, the linear system matrix $\mathbf{B}$ is partitioned into k row blocks and encoded into P row blocks using a (P, k) code with rate $R = \frac{k}{P}$. Each encoded row block is stored at one worker. We now state an important difference in our code: *at each iteration, we use a different generator matrix* $\mathbf{G}^{(t)}$*, but its sparsity pattern remains the same across iterations*. We choose each non-zero entry g_{ij}^t to be a standard Gaussian r.v., and all of these r.v.s are independent of each other. The fixed sparsity pattern $\mathbf{G}$ determines which (sparse) row blocks of the uncoded matrix $\mathbf{B}$ are stored at each worker. In particular, $\mathbf{B}_j, j = 1, \ldots, k$ is stored at worker-i, $i = 1 \ldots P$, when $\mathbf{G}_{i,j} = 1$. However, instead of precomputing the encoded submatrices $\widetilde{\mathbf{B}}_i$ as in (7.2), the i-th worker just stores its required row blocks in $\mathbf{B}$, because the code is time-varying. Similarly, we also partition the vector $\mathbf{y}$ into k subvectors of length b and store them in the P workers in exactly the same fashion as $\mathbf{B}$. At the t-th iteration, worker-i computes $(\mathbf{G}^{(t)})_{i\text{-th row}}\text{mat}\,(\mathbf{B}\mathbf{x}_t + \mathbf{y})$. Since the sparsity pattern is fixed, although the code is time-varying, stored data blocks at each worker remain the same.

At each iteration, a random fraction ϵ of the workers fail to send their results back due to either erasures (packet losses) or stragglers (the communications with slow workers are discarded to save time). Then, at the master node, available results are $\mathbf{G}_s^{(t)}\text{mat}\,(\mathbf{B}\mathbf{x}_t + \mathbf{y})$, where $\mathbf{G}_s^{(t)}$ is the submatrix of $\mathbf{G}_s$ formed by the linear combinations at the non-erased workers. We call $\mathbf{G}_s^{(t)}$ a "partial generator matrix". In existing works on coded computing, if a dense Vandermonde-type code is used, the desired result $\mathbf{B}\mathbf{x}_t + \mathbf{y}$ can be decoded from $\mathbf{G}_s^{(t)}\text{mat}\,(\mathbf{B}\mathbf{x}_t + \mathbf{y})$ if $1 - \epsilon > R$, because any square submatrix $\mathbf{G}_s^{(t)}$ of a Vandermonde matrix is invertible. However, if $\mathbf{G}$ is extremely sparse, even if ϵ is very small, it is possible that $\mathbf{B}\mathbf{x}_t + \mathbf{y}$ cannot be decoded because $\mathbf{G}_s^{(t)}$ may not be invertible.

7.3 Main results: substitute decoding for coded iterative computing

The key observation is that although $\mathbf{G}_s^{(t)}$ (i.e., the remaining part of the encoding matrix corresponding to the workers that do not fail) may not have full column rank, we can get partial information of $\mathbf{B}\mathbf{x}_t + \mathbf{y}$ from $\mathbf{G}_s^{(t)}\text{mat}\,(\mathbf{B}\mathbf{x}_t + \mathbf{y})$. Suppose that the SVD of $\mathbf{G}_s^{(t)}$ is

$$(\mathbf{G}_s^{(t)})_{(1-\epsilon)P\times k} = \mathbf{U}_t\mathbf{D}_t\mathbf{V}_t^\top, \tag{7.4}$$

where the matrix $\mathbf{V}_t$ has orthonormal columns and has size $k\times\text{rank}(\mathbf{G}_s^{(t)})$. By multiplying $\mathbf{L}_t = \mathbf{D}_t^{-1}\mathbf{U}_t^\top$ to the partial coded results $\mathbf{G}_s^{(t)}\text{mat}\,(\mathbf{B}\mathbf{x}_t + \mathbf{y})$, the master node obtains

$$(\mathbf{D}_t^{-1}\mathbf{U}_t^\top)\mathbf{G}_s^{(t)}\text{mat}\,(\mathbf{B}\mathbf{x}_t + \mathbf{y}) \overset{(a)}{=} \mathbf{V}_t^\top\text{mat}\,(\mathbf{B}\mathbf{x}_t + \mathbf{y})\,, \tag{7.5}$$

where (a) follows from (7.4). Then, the master node finds an orthonormal basis of the orthogonal complementary space of the column space of $\mathbf{V}_t$, i.e., an orthonormal basis of $\mathcal{R}^\perp(\mathbf{V}_t)$ (where $\mathcal{R}^\perp(\cdot)$ means the orthogonal complementary space), and forms the basis into a matrix $\widetilde{\mathbf{V}}_t$, such that the matrix $[\mathbf{V}_t, \widetilde{\mathbf{V}}_t]$ is an orthonormal one[2] of size $k \times k$, i.e., $\mathbf{V}_t$, $\widetilde{\mathbf{V}}_t$ are orthogonal to each other, and

$$\mathbf{V}_t\mathbf{V}_t^\top + \widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top = \mathbf{I}_k. \tag{7.6}$$

The master node uses $\mathbf{V}_t^\top\text{mat}\,(\mathbf{B}\mathbf{x}_t + \mathbf{y})$ obtained from (7.5) and the stored $\mathbf{x}_t$ as side information to obtain a good estimate of $\mathbf{B}\mathbf{x}_t + \mathbf{y}$ to compute $\mathbf{x}_{t+1}$. In particular, $\mathbf{x}_{t+1}$ is

$$\mathbf{x}_{t+1} = \text{vec}\left([\mathbf{V}_t, \widetilde{\mathbf{V}}_t] \cdot \begin{bmatrix}\mathbf{V}_t^\top\text{mat}\,(\mathbf{B}\mathbf{x}_t + \mathbf{y}) \\ \widetilde{\mathbf{V}}_t^\top\text{mat}\,(\mathbf{x}_t)\end{bmatrix}\right). \tag{7.7}$$

[2]If $\mathbf{G}_s^{(t)}$ has full rank, $\mathbf{V}_t$ is already a square orthonormal matrix and in this case $\widetilde{\mathbf{V}}_t$ is the NULL matrix, because $\mathcal{R}^\perp(\mathbf{V}_t)$ is the trivial space $\{\mathbf{0}\}$.

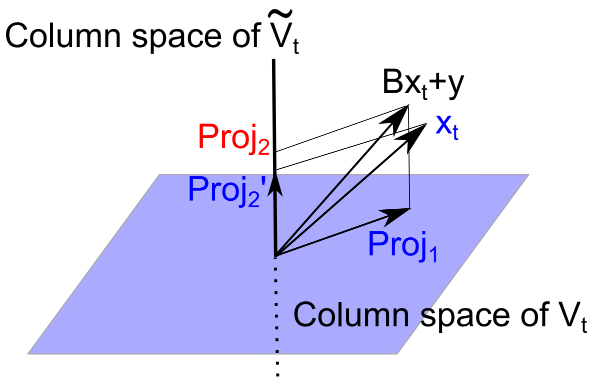


Figure 7.3: This is an illustration of substitute decoding where the known parts are colored blue and the unknown parts are colored red. From $\mathbf{G}_s^{(t)}(\mathbf{B}\mathbf{x}_t + \mathbf{y})$, we can get the projection of $\mathbf{B}\mathbf{x}_t + \mathbf{y}$ onto the column space of $\mathbf{V}_t$ (see Proj$_1$). For the unknown part Proj$_2$, we use the projection of $\mathbf{x}_t$ instead, which is Proj$_2'$.

This is equivalent to

$$\mathbf{x}_{t+1} = \text{vec}\left(\mathbf{V}_t\mathbf{V}_t^\top \text{mat}\left(\mathbf{B}\mathbf{x}_t + \mathbf{y}\right) + \widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top \text{mat}\left(\mathbf{x}_t\right)\right). \tag{7.8}$$

Remark 15. (**Intuition underlying substitute decoding**) We provide intuition by looking at a scalar-version as shown in the lower part of Fig. 7.1. In the scalar version, the vector length b of each subvector of $\mathbf{x}_t$ satisfies $b = 1$ and $\mathbf{I}_b = 1$ and (7.8) becomes

$$\mathbf{x}_{t+1} = \mathbf{V}_t\mathbf{V}_t^\top(\mathbf{B}\mathbf{x}_t + \mathbf{y}) + \widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top \mathbf{x}_t. \tag{7.9}$$

Since $\mathbf{V}_t$ and $\widetilde{\mathbf{V}}_t$ have orthogonal columns and they are orthogonal to each other, $\mathbf{V}_t\mathbf{V}_t^\top$ and $\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top$ are two projection matrices onto the column spaces of $\mathbf{V}_t$ and $\widetilde{\mathbf{V}}_t$ respectively. Since $\mathbf{V}_t$ is obtained from the SVD of $\mathbf{G}_s^{(t)}$ (see equation (7.4)), the projection $\mathbf{V}_t\mathbf{V}_t^\top$ is the projection to the row space of $\mathbf{G}_s^{(t)}$. The intuition of substitute decoding is that even if we cannot get the exact result of $\mathbf{B}\mathbf{x}_t + \mathbf{y}$ by inverting the sparse $\mathbf{G}_s^{(t)}$, we can at least obtain the projection of $\mathbf{B}\mathbf{x}_t + \mathbf{y}$ onto the row space of $\mathbf{G}_s^{(t)}$. Then, for the remaining unknown part of $\mathbf{B}\mathbf{x}_t + \mathbf{y}$, i.e., the projection of $\mathbf{B}\mathbf{x}_t + \mathbf{y}$ onto the right null space of $\mathbf{G}_s^{(t)}$, we use the projection $\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top \mathbf{x}_t$ of the side information $\mathbf{x}_t$ to substitute. This intuition is illustrated in Fig. 7.3. We give an outline of the substitute decoding algorithm for the row-splitting case in Algorithm 5.

7.3.1 Cost analysis of substitute decoding

First, let us analyze the communication cost. For each worker, the communication cost at each iteration comes from the transmission of $\mathbf{x}_t$ of length N and the transmission of the result of length $b = N/k$. So the total number of communicated floating point numbers is $N(1 + 1/k)$ which is linear in N.

Algorithm 5 Coded Power Iterations for Row-wise Splitting

```
Input: Input $\mathbf{y}$, matrix $\mathbf{B}$ and sparsity pattern $\mathbf{G}$.
Preprocessing: Partition $\mathbf{B}$ into row blocks and $\mathbf{y}$ into subvectors and store them distributedly as specified by the sparsity pattern matrix $\mathbf{G}$. Generate a series of random generator matrices $\mathbf{G}^{(t)}, t = 1, 2, \ldots, T$.
Master Node: Send out $\mathbf{x}_t$ at each iteration and receive partial coded results. Compute $\mathbf{x}_{t+1}$ using substitute decoding (7.8), where $\mathbf{V}$, $\widetilde{\mathbf{V}}$ are obtained from SVD (7.4) and $\text{mat}\left(\mathbf{B}\mathbf{x}_t + \mathbf{y}\right)$ is computed using (7.5).
Workers: Worker-$i$ computes $(\mathbf{G}^{(t)})_{i\text{-th row}}\text{mat}\left(\mathbf{B}\mathbf{x} + \mathbf{y}\right)$.
Output: The master node outputs $\mathbf{x}_T$.
```

For the computation cost at the master node, the SVD has complexity $O(kP^2)$ which is negligible. The computation of (7.5) and the substitute decoding given in (7.7) together have complexity $O(k^2b) = O(kN)$ which is linear in N. The vectorization and matricization steps have a negligible cost.

The i-th worker computes a sub-vector of the entire $\mathbf{B}\mathbf{x}$. Denote by E the number of non-zeros in $\mathbf{B}$. Suppose the sparse generator matrix $\mathbf{G}$ has d ones in each row. Then, the complexity at each worker is $O((dE)/k)$. This complexity can be superlinear in N for dense graphs, but usually, the average degree of a graph is a large constant. This means the computation cost at each worker is also linear in N, but with a large constant.

7.3.2 Convergence Analysis of the Coded Power Iterations Using Substitute Decoding

Analysis of Algorithm 5

Denote by $\mathbf{x}^*$ the true solution of $\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{y}$. Then,

$$\begin{aligned}\mathbf{x}^* &\overset{(a)}{=} \text{vec}\left(\mathbf{V}_t\mathbf{V}_t^\top \text{mat}\left(\mathbf{x}^*\right)\right) + \text{vec}\left(\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top \text{mat}\left(\mathbf{x}^*\right)\right)\\ &= \text{vec}\left(\mathbf{V}_t\mathbf{V}_t^\top \text{mat}\left(\mathbf{B}\mathbf{x}^* + \mathbf{y}\right) + \widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top \text{mat}\left(\mathbf{x}^*\right)\right),\end{aligned} \tag{7.10}$$

where (a) holds because of (7.6). Defining the remaining error as $\mathbf{e}_t = \mathbf{x}_t - \mathbf{x}^*$ and subtracting (7.10) from (7.8), we have

$$\mathbf{e}_{t+1} = \text{vec}\left(\mathbf{V}_t\mathbf{V}_t^\top \text{mat}\left(\mathbf{B}\mathbf{e}_t\right)\right) + \text{vec}\left(\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top \text{mat}\left(\mathbf{e}_t\right)\right). \tag{7.11}$$

Remark 16. (Why substitute decoding suppresses error) Before presenting formal proofs, we show the underlying intuition on why the substitute decoding (7.7) approximates the noiseless power iteration $\mathbf{x}_{t+1} = \mathbf{B}\mathbf{x}_t + \mathbf{y}$ well. Again, we look at the scalar version, i.e., $\mathbf{I}_b = 1$. In this case, (7.11) becomes

$$\mathbf{e}_{t+1} = \mathbf{V}_t\mathbf{V}_t^\top\mathbf{B}\mathbf{e}_t + \widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top\mathbf{e}_t. \tag{7.12}$$

Ideally, we want $\mathbf{e}_{t+1} = \mathbf{B}\mathbf{e}_t$ since $\mathbf{B}$ is a contraction matrix (recall that $\rho(\mathbf{B}) < 1$). Due to noise, we can only realize this contraction in the column space of $\mathbf{V}_t$, which is the first term $\mathbf{V}_t\mathbf{V}_t^\top\mathbf{B}\mathbf{e}_t$. Although the partial generator matrix $\mathbf{G}_s^{(t)}$ may not have full rank due to being sparse (i.e., $\dim(\mathcal{R}(\mathbf{V}_t)) < k$), $\mathbf{G}_s^{(t)}$ can be close to full rank. This means that the column space $\mathcal{R}(\widetilde{\mathbf{V}}_t)$ can have low dimension. Therefore, for the second term in (7.12), the projection matrix $\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top$ suppresses the larger error $\mathbf{e}_t$ (compared to $\mathbf{B}\mathbf{e}_t$) by projecting it onto the low-dimensional space $\mathcal{R}(\widetilde{\mathbf{V}}_t)$. In Theorem 7.3.1, we will show $\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top$ reduces $\mathbb{E}[\|\mathbf{e}_t\|^2]$ by a small multiple factor that decreases to 0 linearly as $\text{rank}(\mathbf{G}_s^{(t)})$ increases.

Definition: (Combined cyclic sparsity pattern) The sparsity pattern matrix satisfies $\mathbf{G} = \begin{bmatrix}\mathbf{S}_1 \\ \mathbf{S}_2\end{bmatrix}$, where $\mathbf{S}_1$ and $\mathbf{S}_2$ are both $k \times k$ square cyclic matrices with d non-zeros in each row.

Assumption 9. (Random failures) At each iteration, a random subset of the workers fail to compute the result due to either stragglers or erasure-type errors. Failure events are independent across all iterations.

Theorem 7.3.1. *(Convergence Rate of Algorithm 5) If the sparsity pattern $\mathbf{G}$ in Definition 7.3.2 is used and Assumption 9 holds, the remaining error $\mathbf{e}_t = \mathbf{x}_t - \mathbf{x}^*$ of Algorithm 5 satisfies*

$$\mathbb{E}[\|\mathbf{e}_{t+1}\|^2] = (1-\delta_t)\mathbb{E}[\|\mathbf{B}\mathbf{e}_t\|^2] + \delta_t\mathbb{E}[\|\mathbf{e}_t\|^2], \tag{7.13}$$

where

$$\delta_t = 1 - \frac{\mathbb{E}[\mathit{rank}(\mathbf{G}_s^{(t)})]}{k}. \tag{7.14}$$

The proof is in Section E.1. From Theorem 7.3.1, we can simply upper-bound $\mathbb{E}[\|\mathbf{B}\mathbf{e}_t\|^2]$ by $\|\mathbf{B}\|_2^2\,\mathbb{E}[\|\mathbf{e}_t\|^2]$ and hence

$$\mathbb{E}[\|\mathbf{e}_{t+1}\|^2] \le [(1-\delta_t)\|\mathbf{B}\|_2^2 + \delta_t] \cdot \mathbb{E}[\|\mathbf{e}_t\|^2]. \tag{7.15}$$

This means that when δ_t is close to 0, i.e., when $\mathbf{G}_s^{(t)}$ is close to full rank, $\mathbb{E}[\|\mathbf{e}_t\|^2]$ converges to 0 with rate close to $\|\mathbf{B}\|_2^{2t}$. In Table 7.1 in Section 7.5.1, we show how δ_t changes with the degree d in Definition 7.3.2. Notice that the noiseless power iterations converge with rate $(\rho(\mathbf{B}))^{2t}$. For the PageRank problem, $\mathbf{B} = (1-d)\mathbf{A}$ and $\mathbf{A}$ is the column-normalized adjacency matrix. We show in Lemma E.3.1 that for Erdös-Rényi model $G(N,p)$, $\Pr\left(\|\mathbf{A}\| > \sqrt{\frac{1+\epsilon}{1-\epsilon}}\rho(\mathbf{A})\right) < 3Ne^{-\epsilon^2 Np/8}$. This means that with high probability $\|\mathbf{A}\|_2 \approx \rho(\mathbf{A})$ and hence $\|\mathbf{B}\|_2 \approx \rho(\mathbf{B})$, and the convergence rate of coded power iteration and that of noiseless power iteration are close. Here, in $G(N,p)$, it suffices for p to be $\Omega(\log N/(N\epsilon^2))$ for $3Ne^{-\epsilon^2 Np/8}$ to be small.

In Figure 7.4, we show the comparison of the convergence exponent achieved by substitute decoding and the other two baselines in a simulation result using a (100,50) code with 3 ones on each row of the generator matrix. The first alternative is the "pessimistic" worst-case bound, i.e., if the number of errors is larger than the number of ones on a column, the computation has a failure. The second one is more optimistic:

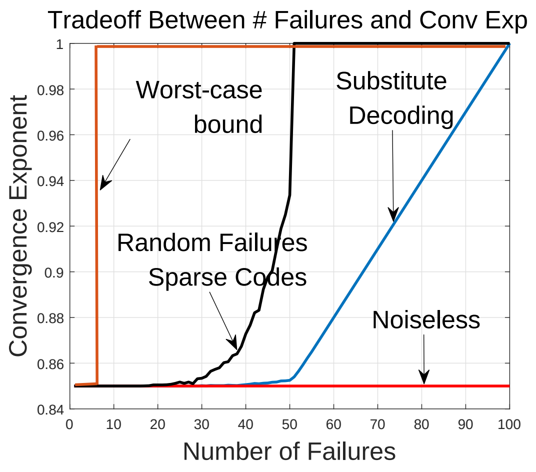


Figure 7.4: Convergence exponent comparison. Substitute decoding can make sparsely coded iterative computing achieve the optimal convergence rate even for a large number of failures/stragglers.

only when the error combination really makes the partial decoding matrix $\mathbf{G}_s^{(t)}$ singular, we claim a failure and use last-iteration's result instead.

7.4 Extended results: applications of substitute decoding

Substitute decoding can be applied to many iterative computing problems. In this section, we show three applications, namely computing multiple leading eigenvectors, computing multiple leading singular vectors and gradient descent. The first two applications are widely used in sparse matrix problems such as spectral clustering [252], principal component analysis and anomaly detection [200]. In this thesis, we only present the extension of computing multiple eigenvectors. The other two extensions can be found online [272]. However, we still present the simulation results regarding all three extensions.

7.4.1 Computing multiple eigenvectors using coded orthogonal iterations

Background on the orthogonal-iteration method

When the input vector $\mathbf{y}$ is zero, the power-iteration method in (7.1) is equivalent to computing the principal eigenvector of B. However, we may be interested in more than one eigenvectors. For example, in spectral clustering or spectral embedding, instead of computing a single eigenvector, one often computes the first r eigenvectors of the (normalized) graph Laplacian matrix $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ [187] and use these eigenvectors as the coordinates of the r-dimensional Euclidean embedding of the nodes in the graph.

The power-iteration method can be generalized to the *orthogonal-iteration* method to compute the first r eigenvectors [25]. To compute the first r eigenvectors of an $N \times N$

matrix $\mathbf{B}$, we initialize an $N \times r$ random matrix $\mathbf{X}_0$ and iterate the following until convergence:

- Compute
$$\mathbf{Z}_t = \mathbf{B}\mathbf{X}_t. \tag{7.16}$$
- Factorize
$$\mathbf{Q}_t\mathbf{R}_t = \mathbf{Z}_t, \tag{7.17}$$
using QR-decomposition.
- Set $\mathbf{X}_{t+1} = \mathbf{Q}_t$.

Usually the $N \times r$ matrix $\mathbf{X}_t$ is tall and thin, because the number of required eigenvectors r is much less than N.

The orthogonal-iteration method in (7.16) and (7.17) is the prototype of many large-scale eigendecomposition methods [215, 239][94, Section 7.3.2][27, 111, 122, 133]. For example, to accelerate the convergence of the orthogonal-iteration method for a symmetric $\mathbf{B}$, one can apply the following procedure as suggested in [27, 215, 216, 239] after obtaining $\mathbf{Q}_t$ and $\mathbf{R}_t$ in (7.17):

- Compute
$$\mathbf{D}_t = \mathbf{R}_t\mathbf{R}_t^\top. \tag{7.18}$$
- Compute the eigendecomposition
$$\mathbf{S}_t\mathbf{\Lambda}_t\mathbf{S}_t^\top = \mathbf{D}_t. \tag{7.19}$$
- Compute the modified eigenvectors
$$\mathbf{X}_{t+1} = \mathbf{Q}_t\mathbf{S}_t. \tag{7.20}$$

Another method is to apply QR-decomposition (7.17) only once after computing (7.16) several times in each iteration, which has the effect of making the spectrum of $\mathbf{B}$ more skewed and making the convergence faster [111].

Coded orthogonal iterations for computing multiple eigenvectors

We show how to implement a coded version of the orthogonal-iteration method to compute the first r eigenvectors of an $N \times N$ matrix $\mathbf{B}$. The number r is much less than the size of $\mathbf{B}$, so the QR-decomposition (7.17) can be directly computed at the master node. The only computation at the workers is the matrix-matrix multiplication $\mathbf{Z}_t = \mathbf{B}\mathbf{X}_t$ in (7.16).

The procedures of coded orthogonal iterations are outlined in Algorithm 6. We partition the matrix $\mathbf{B}$ into column blocks $[\mathbf{B}_1, \dots, \mathbf{B}_k]$ and distribute them to the workers as specified by the sparsity pattern matrix $\mathbf{G}$. At each iteration, the master node breaks the $\mathbf{X}_t$ into k submatrices $\mathbf{X}_t^j, j = 1, \dots, k$ and each worker computes a linear combination of $\mathbf{W}_{j,t} := \mathbf{B}_j\mathbf{X}_t^j, j = 1, \dots, k$. The i-th worker computes the linear combination $\sum_{j=1}^k g_{ij}^t\mathbf{W}_{j,t}$. The collected results at the master node, if no noise is present, can be compactly written as $\bar{\mathbf{W}}_t(\mathbf{G}^{(t)})^\top$, where $\bar{\mathbf{W}}_t = [\mathbf{W}_{1,t}, \mathbf{W}_{2,t}, \dots, \mathbf{W}_{k,t}]$. Notice that a rigorous way to write $\bar{\mathbf{W}}_t(\mathbf{G}^{(t)})^\top$ is to change $\mathbf{G}^{(t)}$ into $\mathbf{G}^{(t)} \otimes \mathbf{I}$ to match the matrix sizes. However, to avoid cumbersome notation and provide a clean presentation of the main idea, we view $\mathbf{W}_{j,t}, j = 1, \ldots, k$ as symbols and still uses $\mathbf{G}^{(t)}$. The master node maintains an estimate of all the $\mathbf{W}_{j,t}$, which can be written as $\widehat{\mathbf{W}}_t = [\widehat{\mathbf{W}}_{1,t}, \ldots, \widehat{\mathbf{W}}_{k,t}]$. In the presence of erasure noise, the master node can combine the partial coded results $\bar{\mathbf{W}}_t(\mathbf{G}_s^{(t)})^\top$ and the previous results $\widehat{\mathbf{W}}_{t-1}$ to obtain the current estimate $\widehat{\mathbf{W}}_t$.

In order to accelerate the convergence, we further apply the procedures from (7.18) to (7.20). This modification has to be applied carefully to the coded computing because the modified eigenvectors in (7.20) may not have the same order as in the previous iteration. Therefore, the naive combination of the results from the past and the current iteration in substitute decoding can be affected by the order of the eigenvectors. To address this problem, we notice that (7.17)-(7.20) are equivalent to the following:

- Factorize
$$\mathbf{Q}_t\mathbf{R}_t = \mathbf{Z}_t, \tag{7.21}$$
- Compute the SVD
$$\mathbf{R}_t = \mathbf{S}_t\mathbf{\Lambda}_t^{\frac{1}{2}}\widetilde{\mathbf{S}}_t^\top, \tag{7.22}$$
- Compute the modified eigenvectors
$$\mathbf{X}_{t+1} = \mathbf{Q}_t\mathbf{S}_t = \mathbf{Z}_t\mathbf{R}_t^{-1}\mathbf{S}_t = \mathbf{Z}_t\widetilde{\mathbf{S}}_t\mathbf{\Lambda}_t^{-\frac{1}{2}}. \tag{7.23}$$

Thus, we can see that the modified QR-steps in (7.21)-(7.23) essentially right-multiplies a matrix $\widetilde{\mathbf{S}}_t\mathbf{\Lambda}_t^{-\frac{1}{2}}$ to $\mathbf{Z}_t$, in which the matrix $\widetilde{\mathbf{S}}_t$ has the function of reordering the eigenvectors, because $\mathbf{\Lambda}_t^{-\frac{1}{2}}$ is only a diagonal matrix. Therefore, at each iteration, we apply the same reordering to $\widehat{\mathbf{W}}_t$ and obtain
$$\widehat{\mathbf{W}}_t^{\text{rotate}} = [\widehat{\mathbf{W}}_{1,t}\widetilde{\mathbf{S}}_t, \ldots, \widehat{\mathbf{W}}_{k,t}\widetilde{\mathbf{S}}_t], \tag{7.24}$$
and uses $\widehat{\mathbf{W}}_{t-1}^{\text{rotate}}$ from last iteration for substitute decoding at the t-th iteration
$$\widehat{\mathbf{W}}_t = \bar{\mathbf{W}}_t\mathbf{V}_t\mathbf{V}_t^\top + \widehat{\mathbf{W}}_{t-1}^{\text{rotate}}\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top, \tag{7.25}$$
where $\bar{\mathbf{W}}_t\mathbf{V}_t\mathbf{V}_t^\top$ are computed from $\bar{\mathbf{W}}_t(\mathbf{G}^{(t)})^\top$ using the SVD on $\mathbf{G}^{(t)}$. The summation of the symbols in $\widehat{\mathbf{W}}_t$ is the estimate of $\mathbf{Z}_t$:
$$\mathbf{Z}_t = \sum_{j=1}^{k} \widehat{\mathbf{W}}_{j,t}, \tag{7.26}$$
Then, instead of (7.17)-(7.20), the master node performs the equivalent steps (7.21)-(7.23) to obtain $\mathbf{X}_{t+1}$. In order to reduce the decoding time, the master can compute $\mathbf{Z}_t$ directly using
$$\mathbf{Z}_t = \bar{\mathbf{W}}_t(\mathbf{G}_s^{(t)})^\top[\mathbf{U}_t\mathbf{D}_t^{-1}\mathbf{V}_t^\top\mathbf{1}_k] + \widehat{\mathbf{W}}_{t-1}^{\text{rotate}}[\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top\mathbf{1}_k], \tag{7.27}$$
and updates (7.25) and (7.24) while communicating with the workers.

Algorithm 6 Computing r Eigenvectors Using Coded Orthogonal Iterations with Column Splitting

```
**Input:** Matrix $\mathbf{B}$ and sparsity pattern $\mathbf{G}$.
**Preprocessing:** Partition $\mathbf{B}$ into column blocks and store them distributedly as specified by the sparsity pattern $\mathbf{G}$. Generate a series of random generator matrices $\mathbf{G}^{(t)}, t = 1, 2, \dots, T$.
**Master Node:** At each iteration, partition $\mathbf{X}_t$ into $k$ submatrices $\mathbf{X}_t^j, j = 1, \dots, k$ and transmits $\mathbf{X}_t^j$ to all worker $i$ such that $\mathbf{G}_{ij} = 1$. Then, receive partial coded results of $\mathbf{B}\mathbf{X}_t$ and conduct substitute decoding (7.27) and get $\mathbf{Z}_t$.
Perform the QR-decomposition steps (7.21)-(7.23) and set $\mathbf{X}_{t+1} = \mathbf{Q}_t$.
While transmitting $\mathbf{X}_{t+1}$ to the other workers, the master node updates the estimates (7.25) and (7.24).
**Workers:** The $i$-th worker computes $\sum_{j=1}^{k} g_{ij}^t \mathbf{W}_{j,t}$.
**Output:** The master node outputs $\mathbf{X}_T$.
```

7.5 Simulation results

7.5.1 Coded power iterations for PageRank computation on the Twitter graph

The row-splitting case

To support Theorem 7.3.1, we compare uncoded, replication-based power iterations and Algorithm 5 on the Twitter graph [153]. We also show the result of noiseless power iterations. There are $P = 20$ workers. In each iteration, 50% of the workers are disabled randomly. In the uncoded simulation, the graph matrix is partitioned into $P = 20$ row blocks. The master node updates $\mathbf{x}_{t+1} = \mathbf{B}\mathbf{x}_t + \mathbf{y}$ on the row blocks where results are available, and maintains the unavailable rows as $\mathbf{x}_t$. In the replication-based simulation, $\mathbf{B}$ is partitioned into 10 row blocks and each one is replicated in 2 workers. Therefore, in each iteration, effectively 50% of the entries in $\mathbf{x}_t$ get updated in the uncoded simulation and about 75% of the entries in $\mathbf{x}_t$ get updated in the replication-based simulation. For the coded case, the sparsity pattern matrix $\mathbf{G}$ is randomly generated using Definition 7.3.2 with degree $d = 2$ and $d = 3$. The code is a $(20, 10)$ code with rate $1/2$. We show in Table 7.1 how the sample average estimate of δ_t changes with the degree of $\mathbf{G}$.

Cost Analysis: We also compare the convergence rates against communication cost (see Fig. 7.5; right). For Algorithm 5, $\mathbf{B}$ is partitioned into $k = 10$ row blocks and encoded into 20 row blocks. The communication complexity in each iteration is $N(1+1/k) = 1.1N$. Similarly, it can be shown that the communication complexity of uncoded and replication-based power iterations are respectively $N(1+1/P) = 1.05N$ and $N(1+1/k) = 1.1N$. Since the average degree of the sparsity pattern is $d = 2 \sim 3$, computation cost and memory consumption only increase by a constant. We also plot the tradeoff for replication scheme with the same storage cost as $d = 3$. However, in this case, the communication complexity for replication is larger, which is $N(1 + d/k) = 1.3N$.

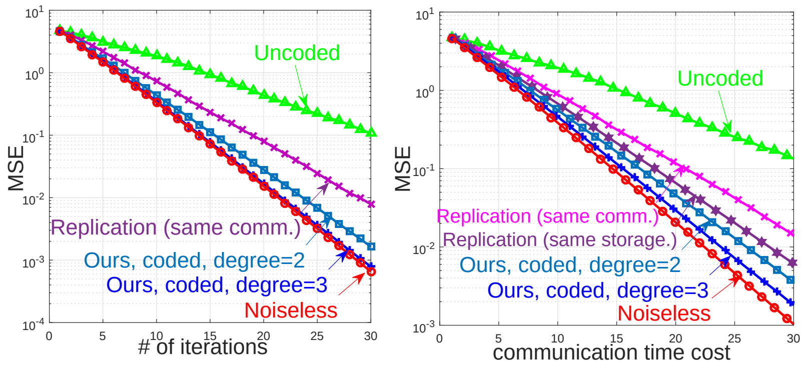


Figure 7.5: The comparison between uncoded, replication-based and substitute-decoding-based power iterations on the Twitter graph. Substitute decoding (blue line) achieves almost exactly the same convergence rate as the noiseless case (red line) for the same number of iterations. Coded computing also beats the other techniques for the same communication time complexity.

$\bar{d}(\mathbf{G})$	2	3	4	5
δ_t	0.1294	0.0442	0.0243	0.0040

Table 7.1: The factor δ_t decreases when the degree of $\mathbf{G}$ increases.

The column-splitting case

We compare the substitute decoding method with the baseline methods again on the Twitter graph for the column splitting case (see Fig. 7.6). We split the graph into $k = 48$ column-blocks and encoded them into $P = 96$ column-blocks. Notice that compared to the row-splitting case, the coded power iterations in the column-splitting case have higher communication cost because each worker has to receive either 2 or 3 subvectors from the master node depending on the degree of the generator matrix (number of non-zeros in each row). This makes its communication complexity increase from $N(1 + 1/k)$ to $N(1 + d/k)$, in which $d = 2$ or 3.

The 2D-splitting case

In 2D splitting, we also compare uncoded, replication-based power iterations and the coded power iterations on the Twitter graph (see Fig. 7.7). The number of workers is $P = 110$ and the matrix is partitioned into $k = 100$ square submatrices using a $\sqrt{k} \times \sqrt{k} = 10 \times 10$ 2D splitting. The workers are grouped into subsets of 11 and in each subset, we apply substitute decoding with a (11,10) code. In each iteration, 10 workers out of the overall 110 ones are disabled, among which 1 worker is permanently disabled and 9 workers are randomly disabled. The other experimental setting is exactly the same as in the row-splitting case.

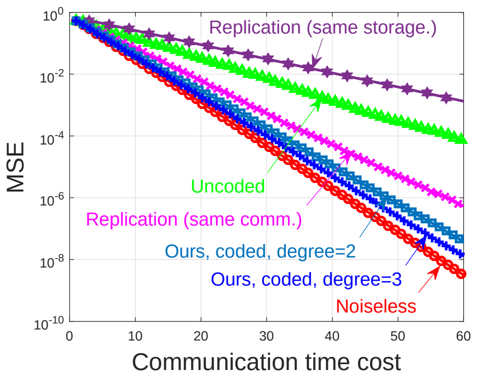


Figure 7.6: The comparison between uncoded, replication-based and substitute-decoding-based power iterations in column-wise splitting. Substitute decoding (blue line) achieves almost exactly the same convergence rate as the noiseless case (red line). Coded computing beats the other techniques for the same communication time complexity.

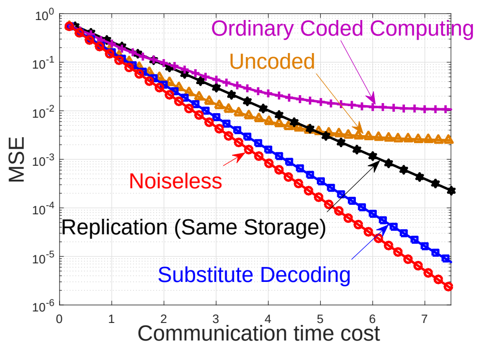


Figure 7.7: The comparison between uncoded, replication-based and substitute-decoding-based power iterations in 2D splitting. All schemes use 2D splitting on the linear system matrix. Substitute decoding with degree 2 (blue line) is not close to the noiseless case (red line) because of increased communication time cost (due to rate=10/11). Coded computing still beats the other techniques for the same communication time cost.

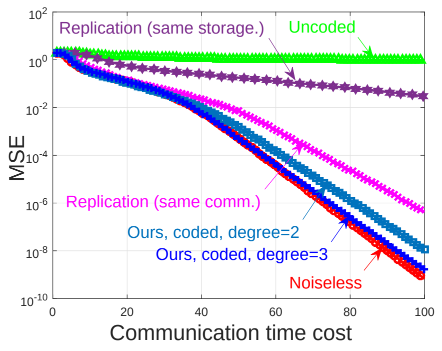


Figure 7.8: The comparison between uncoded, replication-based and substitute-decoding-based orthogonal iterations. Substitute decoding (blue line) achieves almost exactly the same convergence rate as the noiseless case (red line).

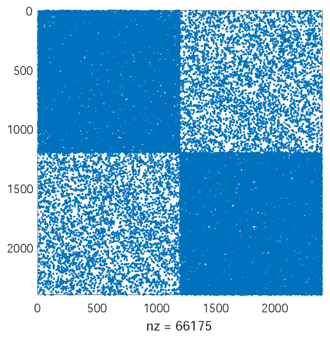


Figure 7.9: This figure shows the clustering result of the graph adjacency matrix using the spectral clustering algorithm with coded computing techniques.

7.5.2 Coded orthogonal iterations for spectral clustering

To test the performance of Algorithm 6, we compare uncoded, replication-based orthogonal iterations and Algorithm 6 in the application of spectral clustering [187, 252] on synthesized graphs (see Fig. 7.8) generated from the stochastic block model with two clusters. We also show the result of noiseless orthogonal iterations. We compute the first two eigenvectors of the normalized graph Laplacian matrix and measure convergence in terms of the MSE of the eigenvector estimation. The second eigenvector (the Fiedler eigenvector) is used to generate the clustering result in Fig. 7.9 with a threshold value 0, i.e., the nodes are partitioned into two clusters based on the signs of the corresponding entries in the second eigenvector.

We generate a graph from the stochastic block model with two clusters and with intra-cluster connection probability 0.02 and inter-cluster connection probability 0.003. There are $P = 96$ workers. In each iteration, 50% of the workers are disabled randomly. In the uncoded simulation, the graph matrix is partitioned into $P = 96$ column blocks. The master node updates $\mathbf{Z}_t = \mathbf{B}\mathbf{X}_t$ using the column blocks where results are available, and maintains the unavailable column blocks from the last iteration. In the replication-based

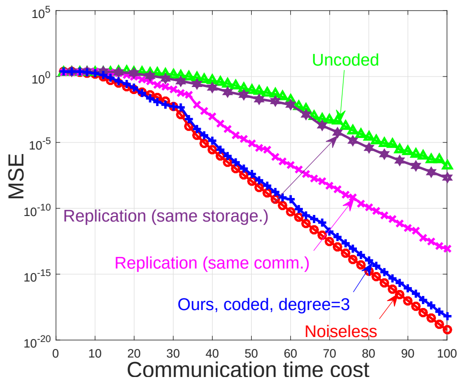


Figure 7.10: The comparison between uncoded, replication-based and substitute-decoding-based orthogonal iterations for principal component analysis. Substitute decoding (blue line) achieves almost exactly the same convergence rate as the noiseless case (red line). Coded computing beats the other techniques for the same communication time complexity.

simulation (same communication), $\mathbf{B}$ is partitioned into $k = 48$ column blocks and each one is replicated in 2 workers. The replication-based method (same storage) uses the same storage of data as the coded case but does not compute the linear combinations of the partial results. For the case of coded computing, the sparsity pattern matrix $\mathbf{G}$ is randomly generated using Definition 7.3.2 for degree $d = 2$ and $d = 3$. The code is a $(96, 48)$ code with rate $1/2$. We run 100 independent simulations and average the results. Since the number of eigenvectors to compute is very small, we do not use the acceleration method in (7.21)-(7.23). In this case, the uncoded computation does not converge at all.

Cost Analysis: In each iteration, the communication complexity is $Nr(1 + d/k) = 1.04Nr$ or $1.06Nr$ because we have r eigenvectors to compute. Similarly, it can be shown that the communication complexity of uncoded, replication-based (same communication) and replication-based (same storage) orthogonal iterations are respectively $Nr(1+1/P) = 1.01Nr$, $Nr(d + d/k) = 3.06Nr$ (we use d=3 for replication with the same storage) and $Nr(1 + 1/k) = 1.02Nr$. Therefore, the communication costs of these strategies are similar.

7.5.3 Coded orthogonal iterations for singular value decomposition

We compare substitute decoding with the baseline algorithms on synthesized matrices with planted dense submatrices (see Fig. 7.11; left) and compute the first 5 singular vectors. We report the MSE of the computed eigenvectors in Fig. 7.10. We also show the result of noiseless orthogonal iterations. We run 100 independent simulations and average the results. In each simulation, we generate a random sparse matrix of size 1000×1000 with non-zero probability 0.01 and plant 5 dense blocks of size 50×50 with non-zero probability 0.2. Each non-zero entry is uniformly distributed in $[0, 1]$. There are $P = 100$ workers. In each iteration, 50% of the workers are disabled randomly. In the uncoded simulation, the master node maintains the partial computation result

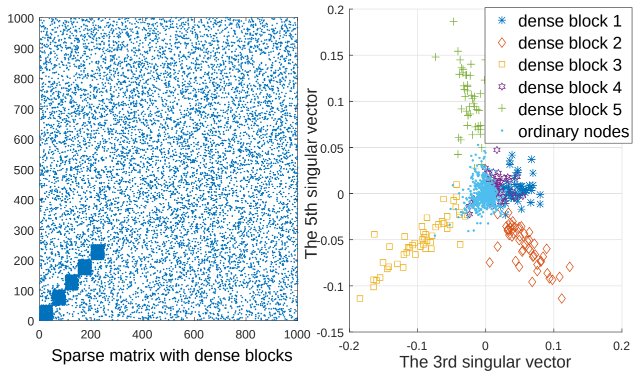


Figure 7.11: This figure shows the phenomenon of "Eigenspokes", which shows that the principal components of a sparse matrix with dense blocks can have spoke-like patterns[200]. These patterns can help identify anomalous dense blocks inside a huge network.

$\mathbf{Z}_{i,t} = \mathbf{B}_i^\top \mathbf{B}_i \mathbf{X}_t$ from the i-th worker at each time slot. If at the $(t+1)$-th time slot the i-th worker fails to send the result, the master node just uses the same partial result in the last iteration. In the replication-based simulation (same communication), $\mathbf{B}$ is partitioned into $k = 50$ row blocks and each one is replicated in 2 workers. The replication-based method (same communication) is similar to the uncoded one except that each $\mathbf{Z}_{i,t}$ is computed in two workers for the purpose of achieving fault/straggler tolerance. The replication-based method (same storage) uses the same storage of data as the coded case but does not compute the linear combinations of the partial results. For the coded case, the sparsity pattern matrix $\mathbf{G}$ is randomly generated by assigning 3 ones in each row. The code is a $(100, 50)$ code with rate $1/2$. We also report the phenomenon of "Eigenspokes" in the right part of Fig. 7.11. For each scattered point (x, y), x is the corresponding entry in the 3rd singular vector, and y is the corresponding entry in the 5th singular vector. The scattered points in the anomalous dense blocks show regular patterns on this plot.

Cost Analysis: The communication complexity is $2Nr$ for all schemes except the replication scheme with the same storage, in which case the communication complexity is $2(1+d)Nr$, where d is the number of ones assigned to each row of the encoding matrix. The computation cost of coded computing increases by a constant factor compared to the uncoded case.

7.5.4 Coded gradient descent using substitute decoding

We compare uncoded, approximate-gradient-coding-based [47] gradient computing (using fractional repetition codes) and substitute decoding on synthesized data (see Fig. 7.12). We also apply substitute decoding to a short-MDS scheme inspired by [286] and report the result. We will introduce the short-MDS scheme in Remark 17.

We compute the result of the following optimization problem

$$\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2 , \tag{7.28}$$

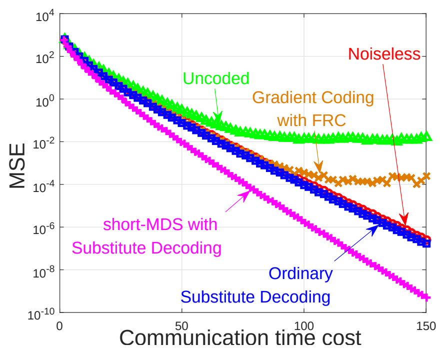


Figure 7.12: The comparison between uncoded, replication-based, approximate-gradient-coding-based [47] and substitute-decoding-based gradient-descent computing. Substitute decoding (blue line) achieves exactly the same convergence rate as noiseless computation (red line). Coded computing beats the other techniques for the same communication time complexity. The reason that the coded computing converges slightly faster than the noiseless case is explained in Remark 18.

where $\mathbf{A}$ is the data matrix of size 6000×2000 and $\mathbf{y}$ is of length 6000. We compute $\mathbf{x}$ using the vanilla gradient descent

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \epsilon \mathbf{A}^\top(\mathbf{A}\mathbf{x}_t - \mathbf{y}). \tag{7.29}$$

We run 100 independent simulations and average the results. In each simulation, we generate a random Gaussian matrix $\mathbf{A}$ and a Gaussian random vector $\mathbf{y}$. There are $P = 100$ workers. All schemes use the same step size $\epsilon = 10^{-4}$. In the noiseless case, we partition the dataset into 120 parts and store one part at each worker. All workers can successfully compute the correct results and send to the master. In the other cases, there are still 120 workers but in each iteration, 15 workers are randomly disabled. In the uncoded simulation, the master aggregates the gradients from only the workers that successfully send back the partial gradients. For the coded case, the sparsity pattern matrix $\mathbf{G}$ is randomly generated by assigning 3 ones in each row. The code is a $(120, 120)$ code with rate $1/2$. The gradient coding algorithm that we compare with is the one in [47] using fractional repetition code. The coding matrix is also of size $(120, 120)$ and there are three ones in each row, i.e., each worker computes three partial gradients and transmits the sum, and each pair of partial gradients is computed at three workers. We choose this algorithm to compare with because it also computes the approximate gradient.

Remark 17. (Short-MDS code using substitute decoding) Now we introduce the short-MDS scheme in Fig. 7.12. This scheme is inspired by the communication-efficient gradient coding scheme [286] in which the partial gradient vectors are broken into subvectors, and a coded technique on the subvectors is applied to further reduce communication cost compared to ordinary gradient coding. We use $P = 120$ to show how the technique works. Suppose the $P = 120$ workers are partitioned evenly into 40 groups of size 3. The data is also partitioned evenly into 40 parts. Each group of 3 workers compute the same partial gradient on one part of the data. Then, each worker breaks the partial gradient into 2 subvectors and sends a linear combination of the two subvectors. Thus, the 3 workers in each group essentially codes the 2 subvectors using a (3,2)-code (note that each worker in the same group computes the same partial gradient so each worker also has the same 2 subvectors).

In this scheme, the communication from each worker to the master is reduced by a factor of 2 compared to ordinary gradient coding, and the worst-case straggler tolerance is 1. Using the notation of [286], this can be denoted by $m = 2$ and $s = 1$. Note this scheme essentially partitions the dataset into the same number of subsets as the number of workers and stores $d = 3$ subsets at each worker, so $d = 3$ in [286]. Thus, we have $d = s + m$, which achieves the same bound in [286, equation (5)]. The advantage of this simple scheme is that we can apply substitute decoding to each (3,2)-code and achieves much higher failure-tolerance than $s = 1$. In fact, as we have shown earlier, the number of failures in each iteration is 15.

Cost Analysis: The communication complexity in this case is $2d_{\text{data}}$ where d_{data} is the dimension of the data, because we have two rounds of communication during each iteration. The communication costs of all the compared schemes are the same except the short-MDS scheme using substitute decoding, in which case the communication complexity is $1.5d_{\text{data}}$. The computation cost of the coded method increases by a constant factor compared to the uncoded method and replication-based method (same communication). The gradient coding method, the replication method (same storage) and our algorithm have exactly the same communication cost, computation cost and storage cost.

Remark 18. It may be surprising that the result of coded computing actually converges slightly faster than the noiseless case. Our explanation is that the coded computing with substitute decoding provides a way of combining past gradients with the current gradients and hence introduces a certain type of momentum into the computation of gradient descent. It has been observed for long [214] that introducing momentum may prevent the convergence trajectory from oscillating in a "narrow valley". Therefore, we conjecture that for the specific problem of computing gradient descent, the substitute decoding method provides a coded way of introducing momentum. Deeply investigating this behavior and the possible improvements in coding schemes resulted from this behavior is our future goal.

7.6 Conclusions and future directions

In this chapter, we propose a new decoding method called substitute decoding for coded computation of iterative algorithms. We utilize the intermediate result from the previous iteration as side information to improve the performance of sparse codes which are often considered weak in the error-correction capability. We show through both theorems and simulations that when the substitute decoding is used, even when we randomly disable $1 - R$ (R is the coding rate) percent of the workers during each iteration, the convergence rate can almost equal to the noiseless rate. This has been shown in Figure 7.4: when the percentage of erasures is more than $1 - R$, one can still use substitute decoding to obtain the near-optimal convergence rate, which is not obtainable even using dense codes. We have also applied the decoding method to a wide range of iterative computation problems, including eigenvalue decomposition, singular value decomposition, and gradient descent.

Chapter 8

Computation on graphs: network topology and distributed computing

8.1 Introduction

In this chapter, we study the interaction between computing problems and the structure of the computation network. The results in this chapter have appeared mainly in the paper [283] and have been extended to the setting of energy efficient computing in [282]. In this chapter, we focus on the problems where there is one *sink* node in a network that needs to collect all the measurement data from all other nodes for further function computation tasks. This problem is of practical importance, e.g., when the specific function computation task of measurements cannot be foreseen, collecting all measurements is the safest strategy. We consider the one-time computation model, which means a one-time gathering of all the data. We assume the data is generated in the form of short and instant messages, and the number of nodes can be quite large. This kind of communication problems with limited data is frequent in distributed control of networks or a distributed monitoring system, where each node is required to report just a few bits to describe the state of the corresponding subsystem in a timely manner. Short message gathering is also necessary in monitoring each agent in an emergency response system, for instance, the wearable wireless sensors that are connected with device-to-device links provide real-time monitoring signals for smart health care.

In these applications, communication throughput might not be the ultimate goal, since data is instant, instead of generated in streams. Following the seminal work of Gallager [86], we consider communication complexity [142], measured in the number of broadcasts in bits, as the optimization goal. We assume, in each time slot, a network agent can *broadcast* a message bit to its neighborhood, and each other agent in this neighborhood receives an independent noisy copy[1] of the broadcast message. Without loss of generality, we assume that each network agent has only one bit of information and the sink node needs to collect all these bits with some required accuracy and minimum

[1]The assumption on noisy networks is suitable to model wireless sensor networks with limited transmission power and decoding capabilities.

number of broadcasts.

Under the assumption of instant message collecting, applying classic error control coding to cope with noisy links is highly non-trivial, since it is impossible for each node to gather enough data to be encoded into blocks before being transmitted and distributed encoding is necessary. This is also one of the main reasons why we explicitly consider noisy channels, rather than considering effectively noiseless channels (on which noise-free communications can be achieved as long as the communication rate is below the channel capacity). An effective computation (encoding) scheme in our context involves carefully designed in-network computations and inter-agent message exchanges.

We call the in-network computations schemes *graph codes*, which extend error control coding to distributed in-network computations. In the following, we briefly discuss the three graph codes that are used in this chapter.

8.1.1 Brief summary of main results

In Section 8.3, general graph topologies are considered and the $\mathcal{GC}$-1 graph code is provided. It is shown that in both BSC and BEC networks with N nodes, the number of broadcasts required by the $\mathcal{GC}$-1 graph code is $\max\{\Theta(\bar{d}_{\mathcal{G}}N), \Theta(N\log N)\}$, where $\bar{d}_{\mathcal{G}}$ denotes the average distance from all agents to the sink. We also obtain a $\max\left\{\Theta(\bar{d}_{\mathcal{G}}N),\right.$ $\left.\Theta(N\log\log N)\right\}$ lower bound on the communication complexity through cut-set techniques in BSC networks, and a $\Theta(\bar{d}_{\mathcal{G}}N)$ lower bound in BEC networks using the same techniques. We also show a $\max\{\Theta(\bar{d}_{\mathcal{G}}N), \Theta(N\log N)\}$ lower bound in constant-degree networks with BEC channels, which implies that the $\mathcal{GC}$-1 graph code also achieves optimality in this scenario.

Motivated by the above mismatch between the achievable result and the converse result, we reconsider the data gathering problem in specific graphs on BSC or BEC links and showed matching results with the lower bound mentioned above. In our paper [283], we considered two types graph, namely geometric graphs and Erdös-Rényi graphs, and designed two computation schemes respectively called $\mathcal{GC}$-2 and $\mathcal{GC}$-3. In the thesis, we only present the results regarding the Erdös-Rényi random graphs. More specifically, we consider an Erdös-Rényi random graph [34] with two further assumptions:

- More links are added to the Erdös-Rényi graph such that the multi-hop distance from each agent to the sink is bounded (e.g., when the sink is a central node and all other nodes have an extra directed link to it);
- The noisy links are BEC instead of BSC.

We call it the extended Erdös-Rényi graph and design the $\mathcal{GC}$-3 code for this graph. We show that the $\Theta(N\log\log N)$ upper bound can be achieved in this graph, without the complete graph assumption. The analysis of the error probability of $\mathcal{GC}$-3 code leads to, as by-products, new fundamental results in the design of erasure codes for point-to-point communications. In particular, we use the analyses for the $\mathcal{GC}$-3 code to show that there exist sparse erasure codes that can achieve diminishing error probability decaying polynomially with the code length even when the encoding is noisy. Moreover, we show that the number of ones in the generator matrix of an erasure code should be at

Table 8.1: Major attributes of the three different types of graph codes.

	Applicable Networks	Analyzable in	Number of Broadcasts
$\mathcal{GC}$-1	Arbitrary connected	BSC (Section 8.3) BEC [283]	$\max\{\Theta(\bar{d}_{\mathcal{G}}N), \Theta(N\log N)\}$
$\mathcal{GC}$-2	Connected geometric	BSC [283] BEC [283]	$\max\{\Theta(\bar{d}_{\mathcal{G}}N), \Theta(N\log\log N)\}$
$\mathcal{GC}$-3	Extended geometric	BEC (Section 8.4)	$\Theta(N\log\log N)$

least $\Omega(N\log N)$ in order to achieve decaying block error probability, using the analysis of $\mathcal{GC}$-3 codes. Note that $\Omega(N\log N)$ is in the same scale as LT codes (Luby transform codes) [161]. In all, the $\mathcal{GC}$-3 code has strong a relevance to erasure codes, and techniques in the in-network computing problem can be applied to the analysis of erasure codes for the classical point-to-point communication setup.

By studying different graph codes, our goal is to theoretically understand in-network computing and data aggregation under the assumptions of link noise and distributed data, with the aim of minimizing the number of communications. Some of the major attributes of the three different types of graph codes are presented in Table 8.1. Since the results on $\mathcal{GC}$-2 codes are omitted, the readers are referred to our paper [283] for more details.

8.1.2 Related works

The works in this chapter were initially inspired by the seminal work of Gallager [86], where the minimum broadcast complexity problem in a noisy complete network is examined, and broadcasting scheme is designed to achieve a complexity of $\Theta(N\log\log N)$ for the parity calculation problem and the identity calculation problem. In [95], this bound is proved tight for the identity calculation problem. Our result in an extended Erdös-Rényi random graph can be viewed as a generalization of prior results under weaker topology assumptions, but the coding techniques are completely different.

In [132], data gathering in a grid network is studied. Theorem IV.1 and Theorem IV.2 in [132] state that, in an $\sqrt{N}\times\sqrt{N}$ grid broadcast network with a transmission radius r, the communication complexity for identity function computation is $\max\left\{\Theta(N^{3/2}/r),\right.$ $\left.\Theta(N\log\log N)\right\}$. In [154], the same problem in a random geometric graph is examined. The proposition 2 of [154] claims that the communication complexity is upper bounded by $\mathcal{O}(N\sqrt{\frac{N}{\log N}})$, under the assumption that the diameter of the network is $\mathcal{O}(\sqrt{\frac{N}{\log N}})$.

8.2 System model and problem formulation

8.2.1 Data gathering with broadcasting

Consider a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $N+1$ agents $\mathcal{V} = \{v_n\}_{n=0}^{N}$, where v_0 is a preassigned sink node. Each agent v_n with $1 \leq n \leq N$ has one bit of information $x_n \in \{0,1\}$ distributed as Bernoulli($\frac{1}{2}$). This is called the *self-information bit*. All self-information bits are independent of each other. Denote the vector of all self-information bits by $\mathbf{x} = (x_1, x_2, ..., x_N)^\top$. The objective is to collect $\mathbf{x}$, in the sink v_0 with high accuracy. Denote by $\mathbf{A}$ the graph adjacency matrix. In this chapter, an edge is directed unless otherwise stated. Denote the one-hop out-neighbors of a node v by $\mathcal{N}_v^+ := \{w \in \mathcal{V} | (v,w) \in \mathcal{E}, w \neq v\}$. Denote the one-hop in-neighbors of a node v by $\mathcal{N}_v^- := \{w \in \mathcal{V} | (w,v) \in \mathcal{E}, w \neq v\}$. If v_m and v_n have a bidirectional link, $A_{m,n} = A_{n,m} = 1$. When the graph is undirected, we write $\mathcal{N}(v)$ for simplicity. Time is slotted. In the t-th slot, only one chosen node $v(t)$ is allowed to broadcast one bit of information in $\mathbb{F}_2$ to its out-neighborhood $\mathcal{N}^+(v(t))$. The channel between any two connected nodes is assumed noisy. We consider either BSC channels or BEC channels.
Assumption 10. (BSC) All channels or graph edges are BSCs with identical crossover probability $\epsilon \in (0, 1/2)$. All channels are independent of each other.
Assumption 11. (BEC) All channels of graph edges are BECs with identical erasure probability ϵ. All channels are independent of each other.

A *broadcast scheme* $\mathscr{S} = \{f_t\}_{t=1}^{\mathscr{C}_{\mathscr{S}}^{(\mathrm{N})}}$ is a sequence of Boolean functions, such that at each time slot t the broadcasting node $v(t)$ computes the function f_t and broadcasts the computed output bit to its out-neighborhood. The parameter $\mathscr{C}_{\mathscr{S}}^{(\mathrm{N})}$ is used to denote the total number of broadcasts in a broadcasting scheme $\mathscr{S}$ which, in our setup, also corresponds to the time complexity. The minimum value of $\mathscr{C}_{\mathscr{S}}^{(\mathrm{N})}$ among all broadcast schemes is defined as the communication complexity which is denoted as $\mathscr{C}^{(\mathrm{N})}$. The arguments of f_t may consist of all the information that the broadcasting node $v(t)$ has up to time t, including its self-information bit $x_{v(t)}$, randomly generated bits and information obtained from its in-neighborhood called the *outer information*. We only consider oblivious transmission schemes, i.e., the broadcasting scheme is predetermined. Transmission by silence is not allowed, i.e., a node has to broadcast when it is required. Denote by $\mathcal{F}$ the set of all feasible oblivious schemes. The final error probability is defined as $P_e^{(N)} = \Pr(\hat{\mathbf{x}} \neq \mathbf{x})$, where $\hat{\mathbf{x}}$ denotes the final estimate at the sink v_0. The problem to be studied is therefore

$$\begin{aligned} &\min_{\mathscr{S} \in \mathcal{F}} \quad \mathscr{C}_{\mathscr{S}}^{(\mathrm{N})}, \\ &\text{s.t.} \lim_{N \to \infty} P_e^{(N)} \leq p_{\text{tar}}. \end{aligned} \tag{8.1}$$

We call this problem the noisy broadcasting problem.

In this chapter, we will consider both fixed graph topologies and random graph topologies, which will be clear in the next subsection. The above mentioned error probability $P_e^{(N)}$ needs to be interpreted in the expected sense when dealing with random graph topologies, i.e., $P_e^{(N)} = \mathbb{E}_{\mathcal{G}}[P_e^{\mathcal{G}}]$ when dealing with random graphs.

8.2.2 Network models

When working with deterministic (but arbitrary) graph topologies, we assume that the network is connected.

Assumption 12. (Network Connectivity) In the directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the sink node v_0 is reachable from each non-sink node $v \in \mathcal{V} \setminus \{v_0\}$ through a sequence $v \to v_{i_1} \to v_{i_2} \cdots \to v_0$ of directed edges.

We use $\mathcal{T} = (\mathcal{V}, \mathcal{E}_{\mathcal{T}})$ to represent the breadth-first search (BFS) spanning tree [58] of the graph rooted at the sink v_0. The edge set $\mathcal{E}_{\mathcal{T}}$ satisfies $|\mathcal{E}_{\mathcal{T}}| = |\mathcal{V}| - 1$. By $d(v, v_0)$, we denote the multi-hop distance from a node v to the sink v_0. An obvious property of the breadth-first search spanning tree $\mathcal{T}$ is that the multi-hop distance $d(v, v_0)$ is the same in $\mathcal{T}$ as in the original graph $\mathcal{G}$. By the l-th layer $\mathcal{V}_l \subset \mathcal{V}$, we denote the set of nodes that have identical multi-hop distance $d(v, v_0) = l$. Denote the maximum distance from a node v to the sink v_0 by L_d. We know that $\mathcal{V} = \bigcup_{l=1}^{L_d} \mathcal{V}_l$ forms a layered partition of the node set. In the BFS tree, the parent-node v_f of a node v is defined to be the unique node such that there exists a directed edge (v, v_f) in the BFS tree's edge set $\mathcal{E}_{\mathcal{T}}$. The descendants of a node v is defined as the set $\mathcal{D}_v \subset \mathcal{V}$ that includes all nodes w that are connected to v through a sequence of directed edges in $\mathcal{E}_{\mathcal{T}}$.

In Section 8.4, we consider the noisy broadcasting problem in the extended Erdös-Rényi network (see Assumption 13), which is slightly different from the original Erdös-Rényi model in [34]. In this model, the connection probability $p = \Theta(\frac{\log N}{N})$ indicates that the average node degree is $\Theta(\log N)$. We will show that the minimum average node degree is at least $\Omega(\frac{\log N}{\log \log N})$, if the error probability of data gathering is required to approach zero when the node number approaches infinity. This result states that $p = \Theta(\frac{\log N}{N})$ is minimum in the order sense except for a $\log \log N$ factor. A sink might be a base station and all agents have direct links to it. In this section, links are assumed to be BECs.

Assumption 13. (Extended Erdös-Rényi Graph) The extended Erdös-Rényi graph is an ER graph with the minimal number of additional links that ensures that each non-source node has directed link to the sink. In the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, all connections are independent of each other. Assume that p satisfies $p = \frac{c \log N}{N}$, where c is a constant. We further assume that each node in $\mathcal{V}$ has a direct link to the sink, in addition to the random connections between these nodes themselves.

Assumption 13 can be interpreted as follows: the edge set $\mathcal{E}$ can be decomposed into $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$, where $\mathcal{E}_1$ is the set of directed edges connecting non-sink nodes, which form the edge set of a directed Erdös-Rényi network with connection probability $p = \frac{c \log N}{N}$, and $\mathcal{E}_2$ can be viewed as the minimum set of edges that is further added to the graph with edge set $\mathcal{E}_1$ so that each non-sink node has a directed link to the sink[2]

[2]It can be shown that this assumption can be relaxed by assuming a bounded multi-hop distance $d_{\max}$

8.3 Main result 1: $\mathcal{GC}$-1 graph codes in a general graph

In this section, we consider general connected network topologies. We design a general distributed in-network computing algorithm called the $\mathcal{GC}$-1 graph code. Recall that in the case of complete networks, as studied in [86, 95], a lower bound on the communication complexity for data gathering is $\Theta(N \log\log N)$. In what follows, we provide a lower bound for general networks. Then, we use the $\mathcal{GC}$-1 graph code to get an upper bound which, we show, is close to the lower bound when the graph diameter is small, and meets the lower bound when the diameter is large. In [283], we also give an intuitive example on why this upper bound can be achieved, and why there is a small gap between the lower and upper bounds.

Theorem 8.3.1. *[283] Suppose the communication links in the graph $\mathcal{G}$ satisfy Assumption 12. Then, if all data are gathered at the sink v_0 with error probability $P_e^{(N)}$ by a feasible broadcasting scheme $\mathscr{S}$, the communication complexity is necessarily bounded below by*

$$\mathscr{C}^{(N)} \geq c_\epsilon \bar{d}_{\mathcal{G}} N, \tag{8.2}$$

where $c_\epsilon = \frac{1-H(P_e^{(N)})}{1-H(\epsilon)}$ is a constant, N denotes the number of nodes in the graph and $\bar{d}_{\mathcal{G}}$ is the average distance to the sink, defined as

$$\bar{d}_{\mathcal{G}} = \frac{1}{N}\sum_{n=1}^{N} d(v_n, v_0). \tag{8.3}$$

8.3.1 In-network computing algorithm

In this part we provide the $\mathcal{GC}$-1 in-network computing algorithm for gathering all data at v_0 in an arbitrary network. Before we provide the algorithm, we provide some preparatory procedures as follows. First, we construct the BFS spanning tree $\mathcal{T} = (\mathcal{V}, \mathcal{E}_{\mathcal{T}})$ rooted at the sink v_0 as defined in Section 8.2.2. Recall that we denote the descendants of the node v by $\mathcal{D}_v$. Define

$$\mathcal{B}_{\mathcal{T}} = \{v \in \mathcal{V} : |\mathcal{D}_v| < \gamma \log N\}, \tag{8.4}$$

where γ is a constant. Define $\mathcal{A}_{\mathcal{T}} = \mathcal{V} \setminus \mathcal{B}_{\mathcal{T}}$. It is obvious that each path from a leaf-node v_n to the root v_0 is constituted by a series of nodes in $\mathcal{B}_{\mathcal{T}}$, followed by another series of nodes in $\mathcal{A}_{\mathcal{T}}$ (as shown in Fig. 8.1).

Then, we propose the $\mathcal{GC}$-1 algorithm, as shown in Algorithm 7. The basic idea is: Each $v \in \mathcal{V}$ gathers all self-information bits from its descendants in $\mathcal{D}_v$. Then, it sends all the information in $\mathcal{D}_v \cup \{v\}$, including bits from its descendants and its own self-information bit, to its parent-node.

All nodes use linear block codes to encode the information that it needs to transmit. Nodes with small descendant size ($|\mathcal{D}_v| < \gamma \log N$) has to insert zeros (dummy bits) to

from each non-sink node to the sink. See [283] for more details.

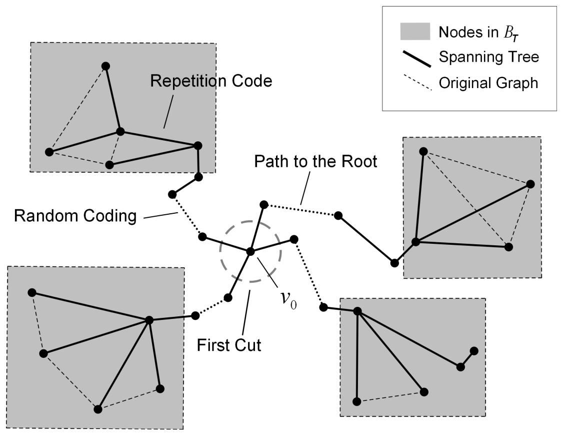


Figure 8.1: *The in-network computing algorithm carried out on the spanning tree.*

the message vector before encoding. The performance guarantee of this algorithm is shown in Theorem 8.3.2. The intuition underlying why the error probability is small is put in Remark 19.

Algorithm 7 $\mathcal{GC}$-1 algorithm

Initialization: Construct the BFS spanning tree $\mathcal{T} = (\mathcal{V}, \mathcal{E}_{\mathcal{T}})$ rooted at the sink v_0.
Step 1: Each leaf-node v encodes the binary vector $(x_v, 0, \ldots, 0)$ with length $\gamma \log N$ using random coding with rate R and transmits the codeword to its parent-node.
Step 2: Each non-leaf node v, from its children-nodes, receives the self-information bits of its entire set of descendants $\mathcal{D}_v$. After all of its children-nodes finish transmitting, the node v relays the self-information bits of all of its descendants and its own self-information bit x_v to its parent-node, using error control codes. Depending on if v is in $\mathcal{B}_{\mathcal{T}}$ or $\mathcal{A}_{\mathcal{T}}$, the coding schemes differ. The coding details are shown below.

- **Actions in $\mathcal{B}_{\mathcal{T}}$:** Each $v \in \mathcal{B}_{\mathcal{T}}$ decodes the self-information bits from $\mathcal{D}_v$ and form a binary vector with length $\mathcal{D}_v + 1$ with its own self-information bit. Then the node v inserts $\gamma \log N - 1 - |\mathcal{D}_v|$ zeros to the vector to make the length $\gamma \log N$ and uses random coding to encode this vector. Finally, it sends the whole $\lceil (|\mathcal{D}_v| + 1)/R \rceil$ bits to its parent-node, where R is the coding rate.
- **Actions in $\mathcal{A}_{\mathcal{T}}$:** Each $v \in \mathcal{A}_{\mathcal{T}}$ decodes the self-information bits from $\mathcal{D}_v$, and uses random coding to encode these bits and its own self-information. Finally, it sends the whole $\lceil (|\mathcal{D}_v| + 1)/R \rceil$ bits to its parent-node, where R is the coding rate.

Theorem 8.3.2. *Suppose the communication links in the graph $\mathcal{G}$ satisfy the assumption 10. Then, for each tuple of constants (R, γ) satisfying*

$$R < \gamma E_r(\epsilon, R), \tag{8.5}$$

where $E_r(\epsilon, R)$ is the random coding error exponent from (2.7)*, the number of broadcasts that the* *scheme $\mathscr{S}$ provided in Algorithm 7 incurs is upper bounded by*

$$\mathscr{C}_{\mathscr{S}}^{(\mathrm{N})} < N(\frac{\bar{d}_{\mathcal{G}}}{R}+1)+N(\gamma\log N/R+1)=\max\{\Theta(\bar{d}_{\mathcal{G}}N),\Theta(N\log N)\}, \tag{8.6}$$

where N denotes the number of nodes in the graph and $\bar{d}_{\mathcal{G}}$ is the average distance to the sink, which is defined in (8.3)*. Moreover, as $N\to\infty$, the error probability $P_e^{(N)}$ decreases polynomially as*

$$P_e^{(N)} < N^{-(\frac{\gamma E_r(\epsilon,R)}{R}-1)}\cdot\left(1+\exp[-E_r(\epsilon,R)/R]\right), \tag{8.7}$$

and, in particular, achieves $\lim_{N\to\infty}P_e^{(N)}=0$.

Proof. In what follows, we show how to obtain the upper bound on the number of broadcasts in (8.6), while the error probability analysis of (8.7) is put in the Appendix F.2. Each node $v\in\mathcal{B}_{\mathcal{T}}$ (including leaf-nodes) transmits a codeword of size $\lceil\gamma\log N/R\rceil$, so the number of broadcasts at each node $v\in\mathcal{B}_{\mathcal{T}}$ satisfies

$$\mathscr{C}_v < \gamma\log N/R+1. \tag{8.8}$$

The number of broadcasts at each node $v\in\mathcal{A}_{\mathcal{T}}$ is

$$\mathscr{C}_v=\lceil(\mathcal{D}_v+1)/R\rceil < (\mathcal{D}_v+1)/R+1. \tag{8.9}$$

Therefore, the final number of broadcasts is

$$\begin{aligned}\mathscr{C}_{\mathscr{S}}^{(\mathrm{N})}&=\sum_{v\in\mathcal{A}_{\mathcal{T}}}\mathscr{C}_v+\sum_{v\in\mathcal{B}_{\mathcal{T}}}\mathscr{C}_v\\&<\sum_{v\in\mathcal{V}}[(\mathcal{D}_v+1)/R+1]+\sum_{v\in\mathcal{V}}(\gamma\log N/R+1)\\&=N(\frac{\bar{d}_{\mathcal{G}}}{R}+1)+N(\gamma\log N/R+1).\end{aligned} \tag{8.10}$$

In Appendix F.2 the remaining part of the theorem, i.e., Eq. (8.7), is proved in detail. □

Remark 19. The nodes in $\mathcal{B}_{\mathcal{T}}$ all have a descendent size $|\mathcal{D}_v|<\gamma\log N$, and hence they do not have enough data to use powerful error control codes with large code length, unless dummy bits are inserted. The code length $\gamma\log N$ is to ensure that, the probability that all transmissions in $\mathcal{B}_{\mathcal{T}}$ are reliable, decays polynomially with N under the union bound. The nodes in $\mathcal{A}_{\mathcal{T}}$ all have large descendent size, so they can use powerful error control codes to carry out block transmissions with low error probability.

Clearly, when the average distance $\bar{d}_{\mathcal{G}}$ to the sink is large and grows polynomially with N, the first term in the RHS of (8.6) dominates. Thus, the upper bound is the same order as the lower bound in Theorem 8.3.1 when the average multi-hop distance $\bar{d}_{\mathcal{G}}$ is large.

Corollary 8.3.3. *Suppose the communication links in the graph $\mathcal{G}$ satisfy the assumption 10. Then, the communication complexity $\mathscr{C}^{(N)}$ of data gathering has an upper bound $\overline{\mathscr{C}^{(N)}}$ and an lower bound $\underline{\mathscr{C}^{(N)}}$, satisfying*

$$\overline{\mathscr{C}^{(N)}} = \max\{\Theta(\bar{d}_{\mathcal{G}}N), \Theta(N\log N)\}, \tag{8.11}$$

$$\underline{\mathscr{C}^{(N)}} = \max\{\Theta(\bar{d}_{\mathcal{G}}N), \Theta(N\log\log N)\}. \tag{8.12}$$

Proof. See Appendix F.1. □

8.4 Main result 2: $\mathcal{GC}$-3 codes in a low-diameter graph

In this section, we provide an in-network computing scheme when the graph diameter is low (in particular, when the average multi-hop distance $\bar{d}_{\mathcal{G}}$ is a constant) and the graph topologies are random, i.e., when the graph $\mathcal{G}$ satisfies the topology assumption 13 and the channel assumption 12. In this in-network computing scheme, the number of broadcasts meets the general lower bound (8.12)[3], with the assumption that $\bar{d}_{\mathcal{G}}$ has order $\mathcal{O}(1)$. As noted in Section 8.2.1, since we are dealing with random graph instances in this section, there are two error probabilities associated with an in-network computation scheme: the conditional error probability $P_e^{\mathcal{G}}$ conditioned on a given graph and the expected error probability P_e^N over all random graph instances.

We recall the assumption 13 of extended Erdös-Rényi-type graphs. Each link is assumed to be a BEC with erasure probability ϵ. In what follows, we show that our proposed $\mathcal{GC}-3$ coding based in-network computing scheme which requires $\Theta(N\log\log N)$ broadcasts. Therefore, our broadcasting scheme can indeed achieve the broadcasting communication complexity lower bound in scaling sense, and, moreover, in sparser graph settings.

8.4.1 In-network computing algorithm

The $\mathcal{GC}-3$ algorithm has two steps. During the first step, let each node broadcast its self-information bit to its out-neighborhood $\mathcal{N}^+(v)$ for t times, where

$$t = \frac{\log(\frac{c\log N}{p_{\text{ch}}})}{\log(1/\epsilon)}, \tag{8.13}$$

and $p_{\text{ch}} > 0$ is a predetermined constant smaller than $1/2$. Then, each node estimates each self-information bit from its in-neighbors. The next lemma provides the probability of a certain bit being erased when transmitted from a node v to one of its out-neighbors. This lemma is a counterpart result of Lemma 2.2.2 in BEC.

[3]Note that the lower bound (8.12) is for BSCs and the techniques we use here are for BECs. However, even if the algorithm in [86] is applied to a complete graph with BECs, the number of broadcasts still scales as $\Theta(N\log\log N)$. Thus, our result is still better in that we allow non-complete graph topologies.

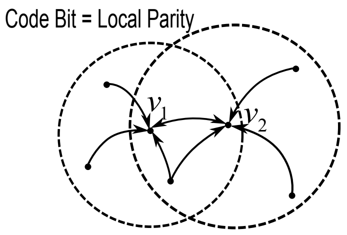


Figure 8.2: *Each code bit is the parity of all one-hop in-neighbors of a specific node. Some edges might be bi-directional.*

Lemma 8.4.1. *Suppose we have a BEC with erasure probability ϵ. Then, the erasure probability of a bit that is repeatedly transmitted for t times on this channel is*

$$P_e = \epsilon^t = \frac{p_{ch}}{c \log N}. \tag{8.14}$$

Proof. The proof follows immediately by substituting in (8.13). □

After estimating each bit, each v_n calculates the local parity. Suppose node v_n receives the self-information bits from its in-neighborhood $\mathcal{N}^-(v_n)$ and if all information bits are sent successfully, v_n can calculate

$$y_n = \sum_{v_m \in \mathcal{N}^-(v_n)} x_m = \mathbf{x}^\top \mathbf{a}_n, \tag{8.15}$$

where $\mathbf{a}_n$ is the n-th column of the adjacency matrix $\mathbf{A}$, and the summation is in the sense of modulo-2. If any bit x_m is not sent successfully, i.e., erased for t times, the local parity cannot be calculated. In this case, y_n is assumed to take the value 'e'. We denote the vector of all local parity bits by $\mathbf{y} = [y_1, y_2, ..., y_N]^\top$. If all nodes could successfully receive all information from their in-neighborhood, we would have

$$\mathbf{y}^\top = \mathbf{x}^\top \mathbf{A}, \tag{8.16}$$

where A is the adjacency matrix of the graph $\mathcal{G}$, and particularly, a random matrix in this section.

During the second step, each node v_n transmits its self-information bit x_n and the local parity y_n in its in-neighborhood back to the sink exactly once. Denote the received version of the bit x_n at the sink by $\tilde{x}_n$. Denote the vector of all self-information bits at the sink by $\tilde{\mathbf{x}} = [\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_N]^\top$. There might be '$e$'s in this vector. Apart from self-information bits, the sink also gets a (possibly erased) version of all local parities. We denote all information gathered at the sink by

$$\mathbf{r} = [\tilde{x}_1, ..., \tilde{x}_N, \tilde{y}_1, ..., \tilde{y}_N] = [\tilde{\mathbf{x}}^\top, \tilde{\mathbf{y}}^\top], \tag{8.17}$$

where $[\tilde{y}_1, ..., \tilde{y}_N]$ is the received version (with possible erasures) of all local parity bits $\mathbf{y}$. That is, there might be some bits in $\mathbf{y}$ changed into value 'e' during the second step. If the channels were perfect, the received information could be written as

$$\mathbf{r}^\top = \mathbf{x}^\top \cdot [\mathbf{I}, \mathbf{A}], \tag{8.18}$$

which is exactly a channel control code with rate $1/2$ and a generator matrix $\mathbf{G} = [\mathbf{I}, \mathbf{A}]$. However, the received version is possibly with erasures, so the sink carries out the Gaussian elimination algorithm to recover all information bits, using all non-erased information. If there are too many bits erased, leading to more than one possible decoded values $\hat{\mathbf{x}}^\top$, the sink claims an error.

In all, the number of broadcasts is

$$\mathscr{C}_{\mathscr{S}}^{(\mathrm{N})} = N \cdot t + 2N = N(2 + \frac{\log(\frac{c \log N}{p_{\mathrm{ch}}})}{\log(1/\epsilon)}) = \Theta(N \log \log N), \tag{8.19}$$

where t is defined in (8.13), and the constant 2 is introduced in the second step of the in-network computing algorithm, when the self-information bit and the local parity are transmitted directly to the sink.

8.4.2 An upper bound on the error probability

In this subsection, we analyze the expected error probability of the previous algorithm. As defined in Section 8.2.1, denote by $P_e^{\mathcal{G}}(\mathbf{x})$ the conditional error probability in gathering all data at the sink conditioned on a graph instance $\mathcal{G}$ and self-information bit vector $\mathbf{x}$. The expected error probability is defined to be $P_e^{(N)}(\mathbf{x}) = \mathbb{E}_{\mathcal{G}}[P_e^{\mathcal{G}}(\mathbf{x})]$. In this section, we prove that $P_e^{(N)}(\mathbf{x})$ converges to zero as $N \to \infty$ for all $\mathbf{x}$.

Theorem 8.4.2. *Suppose the graph $\mathcal{G}$ satisfies the topology assumption 13 and the channel assumption 11. Suppose $\delta > 0$ is a constant, $p_{ch} \in (0, \frac{1}{2})$ is a constant, ϵ is the channel erasure probability and $\varepsilon_0 = (\frac{2}{1-1/e} + 1)p_{ch} + \epsilon$. Assume $c \log N > 1$. Define*

$$b_\delta = \frac{1}{2}(1 - \varepsilon_0)(1 - \frac{1 - e^{-2c\delta}}{2}), \tag{8.20}$$

and assume

$$\epsilon < b_\delta. \tag{8.21}$$

Then, for the transmission scheme in Section 8.4.1, we have

$$P_e^{(N)} \leq \left\{ (1 - b_\delta)^N + \delta e \epsilon \frac{N^{2 - c(1-\varepsilon_0)(1-c\delta)}}{\log N} \right\}. \tag{8.22}$$

That is to say, if $2 < c(1 - \varepsilon_0)(1 - c\delta)$, the error probability eventually decreases polynomially with N. The rate of decrease can be maximized over all δ that satisfies (8.21).

Proof. See Appendix F.5. □

Remark 20. The $\mathcal{GC}$-3 code is "capacity achieving" in some sense, in that this code has rate $\frac{1}{2}$, and this code can be used even when the erasure probability $\epsilon \approx \frac{1}{2}$. Consider the case when $\epsilon = \frac{1}{2} - \Delta$, where Δ is a small constant. In Theorem 4, choose $\delta = \frac{\Delta}{2c}$ and $p_{\mathrm{ch}} = \frac{\Delta}{2(\frac{2}{1-1/e}+1)}$. In this case, the constants in Theorem 4 satisfy $\varepsilon_0 = \epsilon + \frac{\Delta}{2} = \frac{1}{2} - \frac{\Delta}{2}$, and

$2b_\delta \geq (1-\varepsilon_0)(1-c\delta) \geq 1-\varepsilon_0-c\delta = \frac{1}{2}$. Then, the error probability upper bound in Theorem 4 can be simplified to

$$\begin{aligned} P_e^{(N)} \leq & (1-(\frac{1}{2}-(\frac{1}{2}-\Delta)))^N + \frac{e\Delta}{2c}(\frac{1}{2}-\Delta)\frac{N^{2-c(\frac{1}{2}+\frac{\Delta}{2})(1-\frac{\Delta}{2})}}{\log N} \\ \leq & (1-\Delta)^N + \frac{e\Delta}{4c}\frac{N^{2-c(\frac{1}{2}+\frac{\Delta}{4})}}{\log N}, \end{aligned} \tag{8.23}$$

which decays polynomially with N for all small $\Delta > 0$ and $c > 4$.

8.4.3 Connections to random erasure codes

Interestingly, the result of Theorem 8.4.2 implies a more fundamental result for erasure codes.

Corollary 8.4.3. *For a discrete memoryless point-to-point BEC with erasure probability ϵ, there exists a systematic linear code with rate-$1/2$ and an $N \times 2N$ generator matrix $\mathbf{G} = [\mathbf{I}, \mathbf{A}]$ such that the block error probability decreases polynomially with N. Moreover, the generator matrix is sparse: the number of ones in $\mathbf{A}$ is $\mathcal{O}(N \log N)$.*

Proof. The proof relies on building the relation between the $\mathcal{GC}$-3 graph code and an ordinary error control code. We construct the error control code as follows:

- Construct a directed Erdös-Rényi network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes and connection probability $p = \frac{c \log N}{N}$, where c is a constant which will be defined later.
- Construct a linear code with the generated matrix $\mathbf{G} = [\mathbf{I}, \mathbf{A}]$, where $\mathbf{A}_{N\times N}$ is the adjacency matrix of the directed network in the previous step, i.e., the entry $A_{m,n} = 1$ if and only if v_m is connected to v_n.

The number of edges in $\mathcal{E}$ is a binomial random variable distributed according to Binomial(N^2, p). Using the Chernoff bound [52], we obtain

$$\Pr(|\mathcal{E}| > 2pN^2) < \exp(-\frac{p^2}{2}N^2) = (\frac{1}{N})^{\frac{c^2}{2}\log N}. \tag{8.24}$$

Then we use the code constructed above to encode N binary bits and transmit the encoded bits via $2N$ parallel BECs to the receiver. Denote by $A_e^{(N)}$ the event of a block error on the receiver side. Define $P_e^{(N)} = \Pr(A_e^{(N)})$ as the block error probability. Note that

$$P_e^{(N)} = \mathbb{E}\left[P_e^{\mathcal{G}}\right], \tag{8.25}$$

where $P_e^{\mathcal{G}} = \Pr\left(A_e^{(N)} \mid \mathcal{G}\right)$ is the block error probability conditioned on the graph instance $\mathcal{G}$. In other words, $P_e^{(N)}$ is the expected block error probability of an ensemble of codes constructed based on directed Erdös-Rényi networks.

Clearly, this point-to-point transmitting scheme is the same as carrying out the in-network computing algorithm in Section 8.4.1, except that the encoding step in the point-to-point case is centralized instead of being distributed. This is equivalent to the in-network computing scheme when channels between neighboring nodes are without erasures and erasures happen only when communicating over the channels to the decoder (compare with the second step of the in-network computing algorithm). Since erasure events constitute a strict subset of those encountered in the in-network computing scheme, the upper bound on the error probability in Theorem 8.4.2 still holds, which means that the expected block error probability $P_e^{(N)}$ goes down polynomially when the constant c designed for the connection probability $p = \frac{c \log N}{N}$ satisfies the same condition in Theorem 8.4.2. Note that

$$\begin{aligned} P_e^{(N)} =& \Pr(A_e^{(N)}) \\ =& \Pr(|\mathcal{E}| > 2pN^2) \Pr\left(A_e^{(N)} \mid |\mathcal{E}| > 2pN^2\right) \\ &+ \Pr(|\mathcal{E}| < 2pN^2) \Pr\left(A_e^{(N)} \mid |\mathcal{E}| < 2pN^2\right). \end{aligned} \tag{8.26}$$

Thus, combining (8.26) with (8.24) and (8.22), we conclude that the block error probability conditioned on $|\mathcal{E}| < 2pN^2$, or equivalently $\Pr(A_e^{(N)} || \mathcal{E}| < 2pN^2)$, decreases polynomially with N. This means that, by expurgating the code ensemble and eliminating the codes that have more than $2pN^2 = \mathcal{O}(N \log N)$ ones in their generator matrices, we obtain a sparse code ensemble, of which the expected error probability decreases polynomially with N. Therefore, there exists a series of sparse codes which obtains polynomially decaying error probability with N. □

8.4.4 The degree lower bound for the $\mathcal{GC}$-3 graph code

In this part, we prove that $p = \Theta(\frac{\log N}{N})$ is the minimum connection probability that gives the polynomial decay of error probability in Theorem 8.4.2. In fact, we will prove a worst-case result for the total number of edges in the computation graph $\mathcal{G}$: the number of edges in the network must be $\Omega(\frac{N \log N}{\log \log N})$. This result shows that, despite a negligible ratio $\frac{1}{\log \log N}$, the connection probability $p = \frac{c \log N}{N}$ is optimal in terms of sparseness. Since the worst-case result is for a fixed graph, we require the connectivity assumption 12.

Theorem 8.4.4. *Suppose the channel assumption 11 holds. Suppose the algorithm in Section 8.4.1 is carried out. Then, if $\lim\limits_{N \to \infty} P_e^{(N)} = 0$, it holds that*

$$|\mathcal{E}| = \Omega(\frac{N \log(N/P_e^{(N)})}{\log \log N}), \tag{8.27}$$

where $|\mathcal{E}|$ denotes the number of all directed edges in the edge set $\mathcal{E}$.

Proof. See Appendix F.6. □

Remark 21. Note that the lower bound (8.27) holds for individual graph instances with arbitrary graph topologies, instead of holding for certain ensemble average.

Similar with Theorem 8.4.2 and Corollary 8.4.3, Theorem 8.4.4 also implies a result in point-to-point coding theory, but the proof is not obtained by directly applying Theorem 8.4.4. We have to carry out a series of network transforms, as shown in Fig. F.1.

Corollary 8.4.5. *For a rate-$1/2$ linear block code with an $N \times 2N$ generator matrix $\mathbf{G} = [\mathbf{I}, \mathbf{A}]$, if there are d_n ones in the n-th column of $\mathbf{A}$, then, the code is asymptotically good for a point-to-point discrete memoryless BEC with erasure probability ϵ, i.e., the block error probability $\lim_{N\to\infty} P_e^{(N)} = 0$, only if*

$$\sum_{n=1}^{N} d_n \log d_n = \Omega(N \log(N/P_e^{(N)})). \tag{8.28}$$

Proof. Suppose we have a code $\mathbf{G} = [\mathbf{I}, \mathbf{A}]$ that satisfies the conditions in this corollary. As shown in Fig F.1(a), construct a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the following procedures

- Set $|\mathcal{V}| = N$;
- Connect a directed edge from the node v_m to the node v_n if $A_{m,n} = 1$, where m can be equal to n, in which case a directed self loop is constructed;
- Assume each edge is a noiseless channel.

After constructing the graph, construct an extra node v_0 to be the sink, and connect each node to the sink. The links to the sink are all assumed to be discrete memoryless BECs with identical erasure probability ϵ. Suppose in the network constructed above, each node $v_n \in \mathcal{V}$ carries a self-information bit x_n. Then, we can use the in-network computing algorithm in Section 8.4.1 to gather all data measurements at the sink v_0. Clearly, what the algorithm does is encoding the information vector $\mathbf{x}$ with the generator matrix $\mathbf{G} = [\mathbf{I}, \mathbf{A}]$ (see (8.18)) and sending the encoded message through $2N$ parallel BECs to the sink. Until now, the inter-node edges in $\mathcal{E}$ are all noiseless. The only noisy edges are from the distributed nodes to the sink, which means in the first step of the in-network computing algorithm, instead of broadcasting each self-information bit for t times (as defined in (8.13)), each node only needs to broadcast once. Therefore, the in-network gathering of all data in the constructed network is equivalent to the encode-and-decode procedure with the block code $\mathbf{G} = [\mathbf{I}, \mathbf{A}]$ on a point-to-point link, and hence they have the same error probability $P_e^{(N)}$.

Now, modify the constructed network by assuming that links from all non-sink nodes to the sink are noiseless when transmitting the parity bits. That is, in the second step of the in-network computing algorithm, these node-to-sink links are only noisy when self-information bits are transmitted. However, assume that the links between non-sink nodes are noisy, as shown in Fig F.1(b). Specifically, for each node v_n, assume that all the directed links from the in-neighborhood $\mathcal{N}_{v_n}^-$ are changed into BECs with identical erasure probability ϵ/d_n, where $d_n = |\mathcal{N}^-(v_n)|$. Now that the local parity that v_n sends to the sink is erased with probability $1 - (1 - \frac{\epsilon}{d_n})^{d_n} < \epsilon$, therefore, if the original network can gather all data with error probability $P_e^{(N)}$, the transformed network can compute it with error probability strictly less than $P_e^{(N)}$.

Now make a further change as shown in Fig F.1(c), which consists of substituting each communication link with erasure probability ϵ/d_n to a set of $\lceil 1 + \frac{\log d_n}{\log(1/\epsilon)} \rceil$ parallel links with erasure probability ϵ connected to a merging gate. This gate claims an 'erasure' only if all bits in the incoming edges are erased. This transform is exactly the same as repeatedly transmitting t times of the same bit as defined in (8.13). After this transform, the erasure probability changes to $\epsilon^{1+\frac{\log d_n}{\log(1/\epsilon)}} < \epsilon/d_n$. Similarly, if the original network can reliably gather all data with error probability $P_e^{(N)}$, the new network can also compute it with lower error probability.

Therefore, if the block code $\mathbf{G} = [\mathbf{I}, \mathbf{A}]$ can be used to successfully transmit all bits on a point-to-point BEC with error probability $P_e^{(N)}$, data gathering in the transformed network shown in Fig F.1(c) can be reliably completed with lower error probability. By Theorem 8.4.4, to achieve error probability $P_e^{(N)}$, the degree of the transformed network should satisfy

$$\sum_{n=1}^{N} d_n \lceil 1 + \frac{\log d_n}{\log(1/\epsilon)} \rceil > N \cdot \frac{\log N - \log\log(1/(1-P_e^{(N)}))}{\log(1/\epsilon)}. \tag{8.29}$$

This implies that (8.28) holds. □

This corollary shows that, if one wants to find a sparse linear block code for BECs, then (8.28) can serve as a lower bound on 'sparseness'. Moreover, if the matrix $\mathbf{A}$ has the same number of ones in each column, then, there are $\Omega(\frac{\log N}{\log\log N})$ ones in each column, in order for (8.28) to hold. Our result is in coding theory but relates to distributed encoding as well.

8.5 Simulation results

We simulate the $\mathcal{GC}$-3 code with different code lengths in an extended Erdös-Rényi network. The ratio of successful identity function computing at the sink node is compared with the number of broadcasts during the entire in-network function computing scheme (see Section 8.4.1 for details), including t in-network broadcasts in the first phase and 2 transmissions to the sink node in the second phase. In Figure 8.3, we can see from the simulation result that the number of broadcasts at each node required for successful identity function computing almost does not change for different network size. This is because the required number of broadcasts is $\mathcal{O}(\log\log N)$ at each node, and hence it increases very slowly with the code length or the number of nodes in the network.

8.6 Conclusions and future directions

In this chapter, we obtain both upper and lower scaling bounds on the communication complexity, measured in the number of bit broadcasts, in the problem of data gathering in arbitrary noisy networks. In particular, using different graph-based distributed encoding schemes, which we call graph codes, we find special graph topologies in which the upper bounds on the number of broadcasts obtained by graph codes meet with the general lower bound in scaling sense.

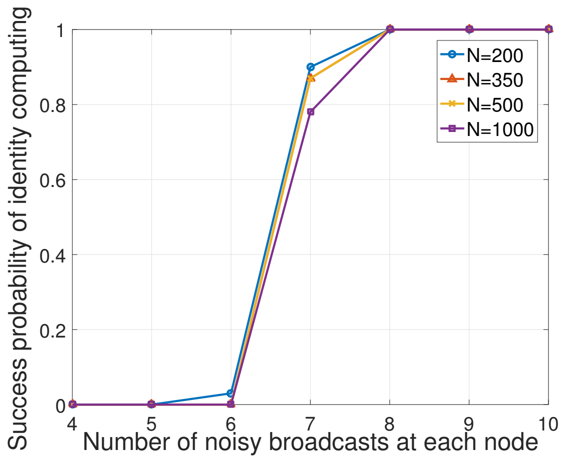


Figure 8.3: *Simulation results of $\mathcal{GC}$-3 codes of different code lengths.*

Furthermore, the analysis techniques of the third graph code, the $\mathcal{GC}$-3 code, is used to construct a sparse erasure code that is used in point-to-point communications. We also use cut-set techniques to show that the obtained code is almost optimal in terms of sparseness (with minimum number of ones in the generator matrix) except for a $\log\log N$ multiple gap. However, quite a few open questions worthy of further research remain. For instance, an issue with the $\mathcal{GC}$-3 code proposed in this chapter is that it can be analyzed only in BEC networks. The focus of this chapter has been primarily on the design of codes that minimize the broadcast complexity, i.e., the number of broadcasts required to achieve function computation. Other practical metrics such as the energy of broadcast (which, depending on the network structure, is somewhat indirectly related to the number of broadcasts) may be of interest in applications too. An extension of $\mathcal{GC}$-3 codes from an energy minimization perspective is provided in our paper [282], which is not included in this chapter.

Chapter 9

Case study 1: distributed logistic regression with noisy decoding

9.1 Introduction

Starting from this chapter, we provide two case studies of the coding-based computing problems. We will see that the techniques and analytical frameworks developed in the previous chapters can be readily applied to many different settings and various computation primitives. The results appeared in these two chapters are reviews of the papers [276] and [275]. In this chapter, we focus on distributed computation of logistic regression, with both unreliable processing units and unreliable memory units. In Section 9.3, we present a distributed coding technique that can tolerate storage faults that happen in multiple time slots in an adversarial manner. We show that if the number of faults during each time slot is bounded below a constant fraction of the code length, the overall number of faults will remain bounded. In Section 9.4, we extend the result to the computation of logistic regression with both memory faults and computation faults. We show that, if the number of faults during each time slot is bounded below a constant fraction of the code length, the difference between the result of logistic regression using unreliable components and the result of logistic regression using fault-free components is bounded by a small constant that is proportional to the number of faults that happen in each time slot and does not grow with the number of iterations.

The coding approach that we adopt is based on verification decoding for compressive sensing. In particular, using a parity check matrix composed of small blocks that represent subspaces, we convert the noisy decoding process into a verification-type decoding which is originally proposed for decoding with erasures. We consider sparse-graph codes over the real field. We use real-number decoding because the ALUs in current GPUs for high-speed learning systems are often optimized for floating-point computations. Since we use sparse-graph codes, the decoding computational complexity overhead is shown to be small. As we will discuss in Remark 23, our algorithm can adopt a "coalesced memory access" technique [207], which further reduces the decoding overhead.

9.2 System model and problem formulation

In this chapter, we use $\mathcal{G}$ to denote a graph and use $\mathcal{V}$ to denote the set of all nodes in $\mathcal{G}$. For a node $v \in \mathcal{V}$ or a node set $\mathcal{S} \subset \mathcal{V}$, we use $\mathcal{N}(v)$ or $\mathcal{N}(\mathcal{S})$ to denote the neighboring nodes of v or $\mathcal{S}$. We use $\|\mathbf{x}\|$ to denote the ℓ_2-norm of the vector $\mathbf{x}$ and use $\|\mathbf{X}\|$ to denote the induced ℓ_2-norm of the matrix $\mathbf{X}$.

Definition: Agent(k,d): An agent is a device[1] that can store k vectors in $\mathbb{R}^d$. Note that d might be one.

The following definition depends on the definition of an expander graph. See Definition 2.2.2 for details.

Definition: Bipartite multi-core system(N,M,k,d,$\mathcal{G}(N, M, d_v, d_c, \alpha.\delta)$,$\mathbf{A}_{ij}$, $\mathbf{b}_{ij}$,$i \in [N]$, $j \in [M]$): A bipartite multi-core system is a set of agents that satisfy the following conditions:

- There are N agents with parameters (k,d), called 'variable agents' or 'left agents', and M agents with parameters ($k+1$,d), called 'check agents' or 'right agents'.
- There is a bipartite expander graph $\mathcal{G}(N, M, d_v, d_c, \alpha, \delta)$ that represents the connections between the N left agents and the M right agents of the bipartite multi-core system: a variable agent v_i and a check agent p_j can communicate with each other only if v_i is connected with p_j. With each edge (v_i,p_j), there is an associated k-by-$(k+1)$ dimensional matrix $\mathbf{A}_{ij}$ and an associated non-zero column vector $\mathbf{b}_{ij}$ such that $\mathbf{A}_{ij}$ has full row rank and $\mathbf{A}_{ij}\mathbf{b}_{ij} = \mathbf{0}$.
- Denote by $\mathbf{X}_i = [\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,k}]$ the k vectors in the i-th variable agent. We say that a check agent is 'satisfied' if

$$\sum_{v_i \in \mathcal{N}(p_j)} \mathbf{X}_i \mathbf{A}_{ij} = (\mathbf{0})_{d\times(k+1)}, j \in [M]. \tag{9.1}$$

Definition: Storage fault ($p(\cdot)$): If a storage fault happens at an agent with parameters (k,d), the stored vectors $\mathbf{X}_i$ change to $\mathbf{X}_i + \mathbf{N}_i$, where $(\mathbf{N}_i)_{d\times k}$ is distributed according to the pdf $p(\cdot)$ with support in $\mathbb{R}^{d\times k}$.

Definition: Computation fault ($q(\cdot)$): If a computation fault happens to a computation result, the correct result is added to a noise value distributed according to the pdf $q(\cdot)$. The dimension of the noise is in accordance with the dimension of the computation result.

Assumption 14. The storage fault pdf $p(\cdot)$ and the computation fault pdf $q(\cdot)$ have no point masses in their supports.

Assumption 15. The storage fault and the computation fault are bounded in ℓ_2-norm by a constant L_e.

Remark 22. Here we briefly remark on how realistic these two assumptions are. In the floating point representation of real-numbers, each real number is represented as a combination of three types of bits: a sign bit, an exponent and a mantissa specifying the actual digits of the number. We assume that in storage and during the computation, the

[1]In a GPU, an agent can be a multi-processor.

mantissa bits and the sign bit can be flipped by storage or computation faults. Then, if we ignore the finite-length effect of real numbers, the faults can indeed be viewed as having no point masses and have bounded value. However, even if a single exponent bit changes, the stored data can change a significant amount, which effectively violates the assumption on bounded error. Therefore, in practice the exponent bits should be taken with extreme care.

9.3 Preliminary result: reliable distributed storage of reals with undetectable faults

Suppose we want to distributedly store $\widetilde{\mathbf{X}} = [\mathbf{X}_1, \dots \mathbf{X}_L] = [\mathbf{x}_{1,1}, \dots \mathbf{x}_{L,k}]$ in a fault-tolerant fashion in presence of storage and computation faults. First, we encode the data $\widetilde{\mathbf{X}}$ into $\mathbf{X} = [\mathbf{X}_1, \dots \mathbf{X}_N] = [\mathbf{x}_{1,1}, \dots \mathbf{x}_{N,k}]$ and for $i \in [N]$, store each segment of data $\mathbf{X}_i = [\mathbf{x}_{i,1}, \dots \mathbf{x}_{i,k}]$ in the i-th variable agent in a bipartite multi-core system with parameters N,M,k,d,$\mathcal{G}(N, M, d_v, d_c)$,$\mathbf{A}_{ij}$,$\mathbf{b}_{ij}$,$i \in [N]$,$j \in [M]$. The vectors $\mathbf{X} = [\mathbf{X}_1, \dots \mathbf{X}_N] = [\mathbf{x}_{1,1}, \dots \mathbf{x}_{N,k}]$ are referred to as coded data. At the beginning, the data are encoded, and thus all check agents are 'satisfied' (see (9.1)).

However, storage faults may result in unsatisfied check agents. Therefore, we use a distributed decoding algorithm to correct errors. Note that the computations in the decoding algorithm are subject to computation faults. During the 0-th iteration, the variable agents broadcast their stored data to the connected check agents. During the t-th iteration, all the check agents compute in parallel, and then all the variable agents compute in parallel.

Assumption 16. The k-by-$(k+1)$ dimensional matrices $\mathbf{A}_{ij}, i \in [N], j \in [M]$ are designed in such a way that the row spaces of any two matrices $\mathbf{A}_{i_1 j}$ and $\mathbf{A}_{i_2 j}$ are not completely the same and no one is contained inside the other (which is always satisfied because we have assumed that each $\mathbf{A}_{ij}$ has full row rank). This property can be achieved, e.g., if all these matrices are chosen with random entries. See Fig. 9.1 for details.

We call a check agent that is connected to one faulty variable agent a 'single-ton', call a check agent that is connected to multiple faulty variable agents a 'multi-ton', and call a check agent that is connected to no faulty variable check agent a 'zero-ton'.

Lemma 9.3.1. *Under Assumption 14 and Assumption 16, the check agent p_j can determine the position and value of a wrong message in the check agent's neighborhood and send the correct message back if and only if the check agent is a single-ton.*

Proof. Recall that the data are initially encoded in such a way that $\sum\limits_{v_i \in \mathcal{N}(p_j)} \mathbf{X}_i \mathbf{A}_{ij} = \mathbf{0}\, \forall j$. Suppose there is a single storage error in one of the variable agents $v_i \in \mathcal{N}(p_j)$, i.e.,

[2]In fact, when the storage noise $\mathbf{N}_i$ has no point mass and when all parity-check solvers of the equation (9.3) are fault-free, all verified messages $\widetilde{\mathbf{M}}^{(t)}_{p_l \to v_i}$ have the same value with probability 1.

Algorithm 8 Fault-Tolerant Distributed Storage

- From variable agent to check agent:
  - **Iteration 0**: For each variable agent v_i, the stored data $\mathbf{X}_i(0) = \mathbf{X}_i$. For v_i and each parity check agent $p_j \in \mathcal{N}(v_i)$, $\mathbf{M}^{(0)}_{v_i \to p_j} = \mathbf{X}_i(0)$ is transmitted from v_i to p_j.
  - **Iteration** $t \geq 1$: For each variable agent v_i, when at least one of the incoming messages $\widetilde{\mathbf{M}}^{(t)}_{p_j \to v_i}, p_j \in \mathcal{N}(v_i)$ is not the error message 'e', set the stored data $\mathbf{X}_i(t)$ to the majority of these messages[2]. Otherwise, do not change the stored data, i.e., $\mathbf{X}_i(t) = \mathbf{X}_i(t-1)$. Then, for each parity check agent $p_j \in \mathcal{N}(v_i)$, transmit $\mathbf{M}^{(t)}_{v_i \to p_j} = \mathbf{X}_i(t)$ to p_j.
- From check agent to variable agent:
  - **Iteration** $t \geq 1$: Each check agent p_j computes the syndrome

$$\mathbf{Z}_j = \sum_{v_l \in \mathcal{N}(p_j)} \mathbf{M}^{(t-1)}_{v_l \to p_j} \mathbf{A}_{lj}. \tag{9.2}$$

1. If $\mathbf{Z}_j = \mathbf{0}_{d \times (k+1)}$, the check agent sends back $\widetilde{\mathbf{M}}^{(t)}_{p_j \to v_l} = \mathbf{M}^{(t-1)}_{v_l \to p_j}$ to each variable agent v_l in $\mathcal{N}(p_j)$.
2. Suppose $\mathbf{Z}_j \neq \mathbf{0}_{d \times (k+1)}$ and there exists a $v_i \in \mathcal{N}(p_j)$ such that $\mathbf{Z}_j \mathbf{b}_{ij} = \mathbf{0}_d$ (which means that the rows in $\mathbf{Z}_j$ are in the row space of $\mathbf{A}_{ij}$). Then, p_j computes the solution $\mathbf{M}$ to the equation

$$\left(\mathbf{M}^{(t-1)}_{v_i \to p_j} - \mathbf{M}\right) \mathbf{A}_{ij} = \mathbf{Z}_j, \tag{9.3}$$

   and sends back $\widetilde{\mathbf{M}}^{(t)}_{p_j \to v_i} = \mathbf{M}$ to v_i. For other variable agents $v_l \neq v_i$, p_j sends back $\widetilde{\mathbf{M}}^{(t)}_{p_j \to v_l} = \mathbf{M}^{(t-1)}_{v_l \to p_j}$.
3. Suppose $\mathbf{Z}_j \neq \mathbf{0}_{d \times (k+1)}$ and $\mathbf{Z}_j \mathbf{b}_{ij} \neq \mathbf{0}_d$ for all $v_i \in \mathcal{N}(p_j)$, p_j broadcasts an error message 'e' to all of its neighbors.

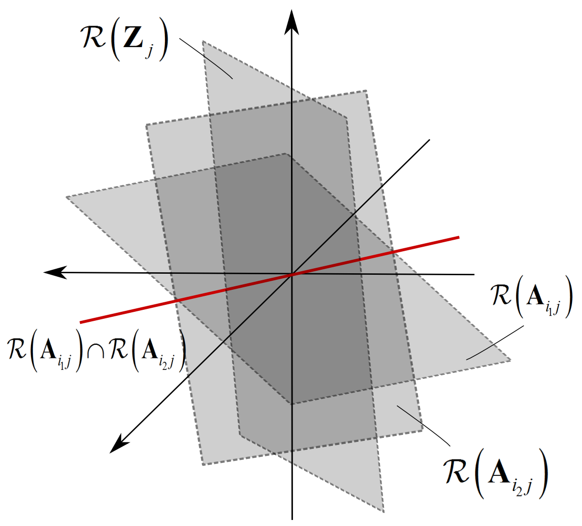

$\mathcal{R}(\mathbf{Z}_j)$
$\mathcal{R}(\mathbf{A}_{i_1 j})$
$\mathcal{R}(\mathbf{A}_{i_1 j}) \cap \mathcal{R}(\mathbf{A}_{i_2 j})$
$\mathcal{R}(\mathbf{A}_{i_2 j})$

Figure 9.1: The row spaces of $\mathbf{A}_{i_1 j}$ and $\mathbf{A}_{i_2 j}$ are not completely the same and no one is contained inside the other. In this way, if the row space of $\mathbf{Z}_j$, the parity check (syndrome) message of the j-th parity check agent, is the same as or contained inside the row space of the row space of a particular weight matrix $\mathbf{A}_{ij}$, one can determine that the fault is from the i-th variable agent. Since the intersection of the row spaces of $\mathbf{A}_{i_1 j}$ and $\mathbf{A}_{i_2 j}$ has low dimension (the red line), the event that the row space of $\mathbf{Z}_j$ is inside the intersection happens with probability 0 for randomly chosen $\mathbf{A}_{i_1 j}$ and $\mathbf{A}_{i_2 j}$ and random storage and computation faults.

$\mathbf{M}^{(t-1)}_{v_i \to p_j} = \mathbf{X}_i + \mathbf{N}_i$. Then, the parity check in (9.2) satisfies

$$\begin{aligned}\mathbf{Z}_j &= \sum_{v_l \in \mathcal{N}(p_j)} \mathbf{M}^{(t-1)}_{v_l \to p_j} \mathbf{A}_{lj} \\ &= \sum_{v_l \in \mathcal{N}(p_j)} \mathbf{X}_l \mathbf{A}_{lj} + \mathbf{N}_i \mathbf{A}_{ij} = \mathbf{N}_i \mathbf{A}_{ij}.\end{aligned} \tag{9.4}$$

In this case, after solving the equation (9.3), in which $\mathbf{M}^{(t-1)}_{v_i \to p_j} = \mathbf{X}_i + \mathbf{N}_i$ and $\mathbf{Z}_j = \mathbf{N}_i \mathbf{A}_{ij}$, the check agent p_j obtains the solution $\mathbf{M} = \mathbf{X}_i$ and sends it back to v_i. If there are at least two faulty variable agents v_{i_1} and v_{i_2} in $\mathcal{N}(p_j)$, the parity check $\mathbf{Z}_j = \mathbf{N}_{i_1} \mathbf{A}_{i_1 j} + \mathbf{N}_{i_2} \mathbf{A}_{i_2 j}$. Under Assumption 14 and Assumption 16, with probability 1, the rows of $\mathbf{Z}_j$ are not in the row space of any $\mathbf{A}_{ij}$. In this case, the error message 'e' is broadcast from p_j to all its neighbors, which conveys that there are too many errors in $\mathcal{N}(p_j)$. That is, the check agent p_j can determine the position and value of a wrong message if and only if the check agent is a single-ton. $\square$

9.3.1 Analysis of the fault-tolerant decoding algorithm

For the t-th time slot, suppose some storage faults happen at the beginning of the time slot. Then, the t-th iteration of the decoding algorithm is carried out. During the decoding iteration, the majority computation at the variable agents may be faulty, and the computation of M in (9.3) at check agents may also be faulty. Denote by $\mathcal{V}_t$ the set of variable agents that are storing incorrect data (not equal to the original data $\mathbf{X}_i$) just before the t-th decoding iteration. Denote by e_t^{var}, e_t^{chk} and e_t^{sto} respectively the number of computation faults at variable agents (the majority computation outputs the incorrect value), the number of computation faults at check agents (the computation of M in (9.3) is incorrect) and the number of storage faults.

Theorem 9.3.2. *(Fault-tolerant Distributed Storage) Suppose the data* $\widetilde{\mathbf{X}} = [\mathbf{X}_1, \dots \mathbf{X}_L]$ *are encoded into* $\mathbf{X} = [\mathbf{X}_1, \dots \mathbf{X}_N]$ *and stored in a bipartite expander multi-core system (*N,M,k,d, $\mathcal{G}(N, M, d_v, d_c, \alpha, \delta)$, $\mathbf{A}_{ij}$, $\mathbf{b}_{ij}$,$i \in [N]$,$j \in [M]$*). Suppose Assumption 14 and Assumption 16 are satisfied. Suppose the number of computation faults* e_t^{var} *and* e_t^{chk} *and the number of storage faults* e_t^{sto} *satisfy*

$$e_t^{var} + e_t^{chk} + e_t^{sto} \leq (2\delta - 1)\alpha N \; \forall t \geq 1. \tag{9.5}$$

Then, the set of variable agents that are storing incorrect data $\mathcal{V}_t$ *satisfies*

$$|\mathcal{V}_t| < \alpha N \; \forall t \geq 1. \tag{9.6}$$

Proof. See Appendix G.1. □

Remark 23. In Section 9.3, we introduced the verification-based decoding algorithm, which can be implemented in parallel for all agents. To further increase the computation speed, as shown in the algorithm, one may simultaneously load k consecutive data points in one variable agent to the cache of the processing unit in the check agent (which is called "coalesced access [207]"). Moreover, if the Tanner graph is quasi-cyclic [48], coalesced access can be carried out efficiently for the entire data stored in a set of consecutive variable agents to a set of consecutive check agents.

Remark 24. When errors are rare, the majority-based update rule at each variable node is not necessary, because the decoding algorithm is based on verifications, and pinpointing one error only requires one parity check. This indicates that, we may consider an asynchronous decoding algorithm, in which each check agent sets a Poisson clock. When its Poisson clock runs out, the check agent sends queries to all connected variable agents, and corrects the data stored in these agents.

Remark 25. The decoding algorithm analysis is based on the assumption that the decoding neighborhood of each variable node in the bipartite graph is tree-like, which again depends on the assumption that N, the number of agents, is large enough. However, this assumption may not be very practical. To address this problem, we may use Protograph-based bipartite graph [69] to allocate the data. In this case, each agent has all the data that are stored in one node of the Protograph, instead of the data that are stored in one node of the entire bipartite graph, and hence the number of agents can be reduced to the number of variable nodes in the small Protograph.

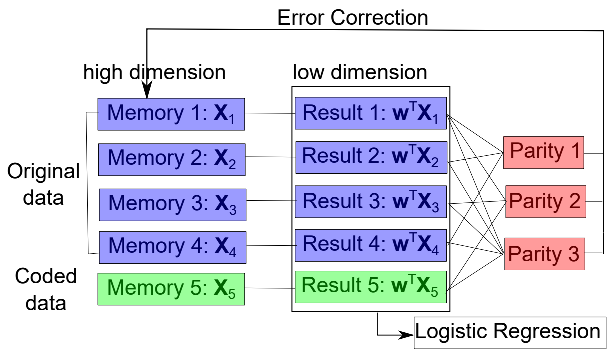


Figure 9.2: The data are split into data blocks and stored in distributed variable agents. The data and are encoded such that (9.1) holds for all parity check agents. Using the intermediate results $\mathbf{w}_t^\top \mathbf{X}_i$ in logistic regression, the parity check agents can identify wrong memory blocks and correct errors. During the computation of logistic regression, the memory faults are repeatedly suppressed by error correction, and hence the convergence of logistic regression can be ensured.

9.4 Main result: distributed logistic regression in the presence of storage and computation faults

9.4.1 A review of distributed logistic regression

Here we provide a brief overview of the logistic regression algorithm for binary classification [29, Section 4.3.2]. Suppose we want to solve a logistic regression problem where the raw data $\widetilde{\mathbf{X}} = [\mathbf{X}_1, \dots \mathbf{X}_L] = [\mathbf{x}_{1,1}, \dots \mathbf{x}_{L,k}]$ are stored distributedly at L agents, and $\mathbf{X}_i$ are stored at the i-th variable agent. Each d-dimensional column vector in $\mathbf{X}_i = [\mathbf{x}_{i,1}, \dots, \mathbf{x}_{i,k}]$ represents one instance of observations of a collection of d features. In total, we have Lk instances of observations of the d features, and each agent stores k instances. Also, we assume that the i-th agent has labels $\mathbf{y}_i^\top = [y_{i,1}, \dots, y_{i,k}]$, where each entry is in $\{0, 1\}$. Denote by $\mathbf{y} = [\mathbf{y}_1; \mathbf{y}_2; \dots; \mathbf{y}_L]$ the collection of all labels. We want to solve the logistic regression problem with ridge regression

$$\max_{\mathbf{w}\in\mathbf{R}^d} \log p(\mathbf{y}|\widetilde{\mathbf{X}}, \mathbf{w}) - \lambda \|\mathbf{w}\|_2^2, \tag{9.7}$$

under the assumption that

$$p(y = 0|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(\mathbf{w}^\top \mathbf{x})}, \tag{9.8}$$

where $\mathbf{x}$ and y denote an arbitrary data vector and its corresponding label.

Problem (9.7) can be solved recursively using a gradient descent method. The gradient update equation can be compactly written as

$$\begin{aligned} \mathbf{w}_{t+1} &= (1 - 2\varepsilon\lambda)\mathbf{w}_t - \varepsilon \nabla L(\mathbf{w}_t) \\ &= (1 - 2\varepsilon\lambda)\mathbf{w}_t - \varepsilon \widetilde{\mathbf{X}} \left[\mathbf{y} - \sigma(\widetilde{\mathbf{X}}^\top \mathbf{w}_t)\right], \end{aligned} \tag{9.9}$$

where $\sigma(\cdot)$ is the entry-wise sigmoid function. This can also be written in a distributed form as

$$\begin{aligned}\nabla L(\mathbf{w}_t) &= \sum_{i=1}^{L} \nabla \mathbf{w}_t^{(i)} \\ &:= \sum_{i=1}^{L} \mathbf{X}_i \left[\mathbf{y}_i - \sigma(\mathbf{X}_i^\top \mathbf{w}_t)\right],\end{aligned} \tag{9.10}$$

which can be implemented in a distributed way.

9.4.2 Coding-inspired fault-tolerance of distributed logistic regression

The details of the fault-tolerant distributed logistic regression algorithm are provided in Algorithm 9. The main difference in the decoding stage between Algorithm 8 and Algorithm 9 is that the parity checks are computed using the linear transform of the data $\mathbf{w}_t^\top \mathbf{X}_i$, instead of the data $\mathbf{X}_i$ themselves. To understand why decoding can indeed correct faults, we note that when no faults happen, the following parity-check equations are satisfied:

$$\sum_{v_i \in \mathcal{N}(p_j)} \mathbf{w}_t^\top \mathbf{X}_i \mathbf{A}_{ij} = \mathbf{0}, j \in [M]. \tag{9.11}$$

Note that the parity check $\mathbf{Z}_j = \sum_{v_i \in \mathcal{N}(p_j)} \mathbf{w}_t^\top \mathbf{M}_{v_l \to p_j}^{(t-1)} \mathbf{A}_{ij}$ in Algorithm 9 is a $1 \times (k+1)$ row vector, which is different from the $d \times (k+1)$ matrix in Algorithm 8. This helps reduce the computational complexity of decoding by a factor of d, which is useful in high-dimensional problems.

Remark 26. In order to reduce the network-traffic for repairing faulty storage nodes, one can introduce a fault indicator after detecting a fault at a check agent, instead of immediately recovering the data stored in the corresponding variable node. This is because a single mismatch between the computation result $\mathbf{w}_t^\top \mathbf{X}_i$ and the parity-check equations may be due to computation fault at this particular time slot, instead of a storage fault. However, if two consecutive mismatches happen at the variable agent, we should replace data, because the probability that a storage fault happens dominates the probability of two consecutive computation faults (under the assumption that the probability of storage errors and the probability of computation errors are comparable).

9.4.3 Analysis of the fault-tolerant distributed logistic regression algorithm

We assume that at the t-th time slot of the computation of logistic regression, there may be five types of faults: (1) the stored data at a variable agent $\mathbf{X}_i$ may be changed due to

[2] For the sake of clarity, we assume that this recovery is fault-free. One can easily generalize this to fault-prone recovery, and carry out analysis for fault-prone recovery parallel to the proof for distributed storage in Theorem 9.3.2.

Algorithm 9 Fault-Tolerant Distributed Logistic Regression

- **Initialization:** Encode the raw data $\widetilde{\mathbf{X}} = [\mathbf{X}_1, \dots \mathbf{X}_L]$ into $\mathbf{X} = [\mathbf{X}_1, \dots \mathbf{X}_N]$ and distribute the parity symbols $[\mathbf{X}_{L+1}, \dots \mathbf{X}_N]$ into $N - L$ new agents, such that all check agents defined in Section 9.2 are satisfied. At the central agent, set $\mathbf{w}_0$ using some rough estimate and broadcast it to all the variable agents.
- **Iteration** t**:** At each variable agent v_i, compute $\nabla \mathbf{w}_t^{(i)}$ in the following steps:
  - Compute $\mathbf{w}_t^\top \mathbf{X}_i$.
  - Carry out one iteration of Algorithm 8 for the computed results $\mathbf{w}_t^\top \mathbf{X}_i$.
  - When a fault at a variable agent v_i is detected by a check agent p_j (i.e., the second case in Algorithm 8 when $\mathbf{Z}_j \neq \mathbf{0}$ and $\mathbf{Z}_j \mathbf{b}_{ij} = \mathbf{0}$), recover the data in the storage of v_i using local parity-check equations[3].
  - Plug in the one-step decoded result of $\mathbf{w}_t^\top \mathbf{X}_i$ into the computation of $\nabla \mathbf{w}_t^{(i)} = \mathbf{X}_i \left[\mathbf{y}_i - \sigma(\mathbf{X}_i^\top \mathbf{w}_t)\right]$ at the i-th variable agent. Send back the computed result $\nabla \mathbf{w}_t^{(i)}$ to the central agent, even if the computation result may still not be completely correct.
- **Update:** At the central agent, compute the sum of all $\nabla \mathbf{w}_t^{(i)}$, $1 \leq i \leq L$, and update the estimate using (9.10). Then, the central agent broadcasts the updated estimate $\mathbf{w}_{t+1}$ to all variable agents.

a storage fault; (2) the computation of $\mathbf{w}_t^\top \mathbf{X}_i$ may be faulty due to a computation fault; (3) the computation at the check agents may be faulty due to a computation fault (and a faulty message may be generated that indicates a variable agent stores wrong value); (4) the majority-based computation at the variable agents may be faulty due to a computation fault; (5) the computation of the gradient descent update $\nabla \mathbf{w}_t^{(i)} = \mathbf{X}_i \left[\mathbf{y}_i - \sigma(\mathbf{X}_i^\top \mathbf{w}_t)\right]$ may be faulty due to a computation fault. We denote by e_t^{sto}, e_t^{loc}, e_t^{chk}, e_t^{var}, and e_t^{updt} the number of the five types of faults. As we have discussed, we assume that the recovery stage, which is after the detection of a fault, is fault-free for the sake of clarity.

Theorem 9.4.1. *(Fault-tolerant Distributed Logistic Regression) Suppose the data* $\widetilde{\mathbf{X}} = [\mathbf{X}_1, \dots \mathbf{X}_L]$ *are encoded into* $\mathbf{X} = [\mathbf{X}_1, \dots \mathbf{X}_N]$ *and stored in a bipartite multi-core system* $(N,M,k,d,\mathcal{G}(N, M, d_v.d_c),\mathbf{A}_{ij}, \mathbf{b}_{ij}, i \in [N], j \in [M])$*. Suppose* $\|\mathbf{x}\| \leq L_x$ *for each column* $\mathbf{x}$ *in the data matrix* $\widetilde{\mathbf{X}}$*. Suppose Assumption 14, 15 and 16 are satisfied. Suppose the fault-free update rule* (9.9) *satisfies* $\|\mathbf{w}\| \leq L_u$*. Suppose* L_ϕ *is a constant and* η *is defined as*

$$\begin{aligned}\eta :=& \left(\frac{1}{4}\sqrt{k} L_x (L_\phi + L_u) + 2k\right) \cdot \alpha L_e \\ &+ \max\left(\frac{1}{4}\sqrt{k} L_x, 1\right) \cdot \beta L_e,\end{aligned} \tag{9.12}$$

Suppose the number of computation faults e_t^{loc} *and* e_t^{chk} *and the number of storage faults* e_t^{sto} *satisfy*

$$d_v e_t^{loc} + e_t^{sto} + e_t^{chk} \leq (2\delta - 1)\alpha N \ \forall t \geq 1. \tag{9.13}$$

Then, the set of variable agents that are storing incorrect data $\mathcal{V}_t$ satisfies

$$|\mathcal{V}_t| < \alpha N, \ \forall t \geq 1. \tag{9.14}$$

Further, suppose the error bound satisfies $L_\phi = \frac{\eta N}{2\lambda}$, and suppose the number of computation faults e_t^{loc}, e_t^{chk}, e_t^{var} and e_t^{updt} satisfy

$$e_t^{loc} + e_t^{chk} + e_t^{var} + e_t^{updt} \leq \beta N. \tag{9.15}$$

Then, the difference between the result of the coded fault-tolerant distributed logistic regression and the result in the fault-free case is bounded in ℓ_2-norm by L_ϕ.

Proof. See Appendix G.2. □

Chapter 10

Case study 2: fault-tolerant convolution with noiseless decoding

10.1 Introduction

The goal of this chapter is to design signal processing systems−in particular, a set of parallel filters operating on the same input−in an error-resilient fashion. Among these linear filters, some filters may suffer from a complete failure and may provide completely arbitrary outputs. Our goal is to obtain the exact filtering output signal despite these failures.

Building on the real-number error control coding from compressive sensing [43, 44], we construct $m-n$ redundant filters in addition to the original n parallel filters in order to enable error correction. We provide a method to detect the faulty filters and correct errors using the linear programming decoding method [44]. We only need the output signals from all filters at a particular time instant t for decoding. We show that all faulty filters can be corrected provided that the errors are sparse, i.e., the number of faulty filters is smaller than some threshold value.

The decoding technique used in this chapter are assumed fault-free. However, as we show in the analysis of computational complexity, the complexity of decoding is much smaller than the complexity of linear filtering. Therefore, we may use higher power for decoding to eliminate faults. Note that we can actually implement an error-prone decoding for linear filtering using ideas from Chapter 9. We use the notation $\mathbf{H}_{\mathcal{T}}$ to denote the submatrix composed of columns of the p-by-m matrix $\mathbf{H}$, which have column indices from the set $\mathcal{T} \subset \{1, 2, \ldots, m\}$. We use the notation diag$(\mathbf{H})$ to denote the diagonal matrix generated from the vector $\mathbf{H}$. We use a linear programming decoding approach proposed by Candes and Tao [44] to detect filter failures, the details of which were presented in Section 2.2.5.

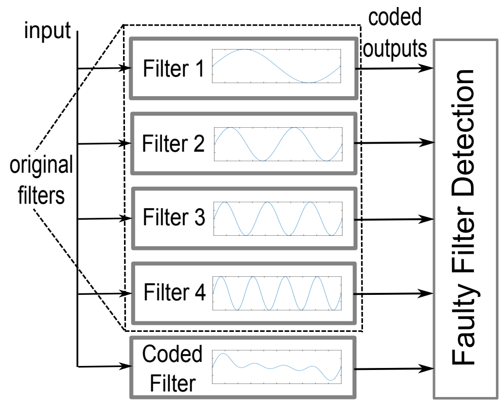


Figure 10.1: Redundant filters are constructed parallel to the original filters, such that the outputs from these filters are in a coded form.

10.2 System model and problem formulation

In this chapter, we would like to achieve fault-tolerance in a group of unreliable linear filters. We assume that the same signal $r(t)$ is the input to a group of linear filters with impulse responses $h_i(t)$, $i = 1, \ldots, n$. The output signals are written as

$$\hat{x}_i(t) = r(t) \star h_i(t). \tag{10.1}$$

The model in (10.1) applies to both digital and analog filters, where t takes discrete values in digital filters. Some implementation issues of digital filters are discussed in Remark 27.

One application of this model is the classical frequency division multiplexing in a multi-user communication system. In this special context, the signal $r(t)$ is the received signal composed of signals from n users and is demodulated by a set of linear filters from the carrier frequency. For example, we may define $x_i(t)$ as the bandlimited baseband signal transmitted by the i-th user. Then, each baseband signal $x_i(t)$ is shifted onto a specific channel (carrier) and turned into a carrier signal $\tilde{x}_i(t)$. One way of generating $\tilde{x}_i(t)$ is to multiply $x_i(t)$ with a specific carrier frequency signal $g_i(t) = \cos 2\pi f_i t$. Finally, the n carrier signals are summed up to form the transmitted signal $r(t)$. In this special context, different $h_i(t)$'s can be viewed as the transfer functions of band-pass filters with different central frequencies.

We assume that the n linear filters are error-prone, i.e., a small portion of these n filters generate completely faulty signals, while the remaining filters are error-free. We would like to find the faulty filters by examining the output signals from all the n filters and correct the erroneous signals. As will be clear in the following sections, the fault-detection algorithm that we propose is a one-time examination, which only requires the outputs $\hat{x}_i(t)$ at only one instant of time t, and which requires much lower computational complexity compared to linear filtering[1]. Therefore, we assume that the

[1]A faulty filter may coincidentally yield a correct output at this particular time t, which prevents this filter from being detected. However, this event has zero probability under mild assumptions on the error statistics.

decoding algorithm can be implemented using error-free computing components.

We will introduce redundancy by adding $m-n$ redundant filters whose impulse responses (or transfer functions) are linear combinations of the impulse responses (or transfer functions) of the original filters. In particular, for digital filters, redundant filters can be constructed by directly computing linear combinations of the tap coefficients of the n original filters. For simplicity, we assume that the complexity of building these redundant filters is the same as that of building each of the original filters. We acknowledge that this assumption might be unrealistic in some applications, and understanding appropriate application domains for such filters, as well as characterizing filter complexity by familiar metrics (e.g. number of taps or number of poles and zeros) is a meaningful future direction. We assume there are at most S faulty filters out of the total m filters.

10.3 Main result: coded parallel linear filters for fault detection

We use a coding-inspired algorithm to detect these faulty filters by adding some 'redundant filters'. In particular, we encode the n linear filters into m linear filters using the generator matrix $\mathbf{G}_{m\times n}=[\mathbf{I}_{n\times n},\mathbf{P}_{n\times p}]^\top$ of a systematic (m,n) linear block code, where $m=n+p$. This encoding process can be written as

$$[h_1(t),\ldots,h_n(t),h_{n+1}(t),\ldots,h_m(t)]^\top=\mathbf{G}_{m\times n}\cdot[h_1(t),\ldots,h_n(t)]^\top. \tag{10.2}$$

The transfer functions of all coded filters can be written as

$$[H_1(s),\ldots,H_n(s),H_{n+1}(s),\ldots,H_m(s)]^\top=\mathbf{G}_{m\times n}\cdot[H_1(s),\ldots,H_n(s)]^\top, \tag{10.3}$$

where the argument s should be changed to z for digital filters.

If all the linear filters are error-free, the ideal output signals can be written as

$$\begin{aligned}[\hat{x}_1(t),\ldots,\hat{x}_m(t)]^\top=&[h_1(t),\ldots,h_m(t)]^\top\star r(t)\\ =&\mathbf{G}_{m\times n}\cdot[h_1(t),\ldots,h_n(t)]^\top\star r(t)\\ =&\mathbf{G}_{m\times n}\cdot[\hat{x}_1(t),\ldots,\hat{x}_n(t)]^\top.\end{aligned} \tag{10.4}$$

However, since the faulty filters generate arbitrary output signals, the true output signals can be written as

$$\begin{aligned}[y_1(t),\ldots,y_m(t)]^\top&=[\hat{x}_1(t),\ldots,\hat{x}_m(t)]^\top+[e_1(t),\ldots,e_m(t)]^\top\\ &=\mathbf{G}_{m\times n}\cdot[\hat{x}_1(t),\ldots,\hat{x}_n(t)]^\top+[e_1(t),\ldots,e_m(t)]^\top,\end{aligned} \tag{10.5}$$

where the error vector $[e_1(t),\ldots,e_m(t)]$ is supported on a sparse set of cardinality at most S, i.e., at most S functions among $e_1(t),\ldots,e_m(t)$ are non-zero functions. We can write (10.5) in a compact form

$$\mathbf{y}(t)=\mathbf{G}_{m\times n}\cdot\hat{\mathbf{x}}(t)+\mathbf{e}(t). \tag{10.6}$$

Remark 27. In some digital filters, the convolution operations (see (10.1)) are computed using IFFT, FFT and entry-wise vector products. One might question the implementation of coded filters in this situation. However, we note that the convolution results $\hat{x}_i(t), \forall i$, are identical and independent of convolution operation details, if all operations are error-free. That is to say, (10.4) holds and is independent of convolution implementation details. Further, in our problem, the faulty filters are allowed to generate arbitrary outputs, which means that (10.5) also holds and is independent of implementation details. However, all filters must be constructed by different devices. This is because if all filters are implemented using a shared faulty FFT computation unit, all outputs $\hat{x}_i(t), \forall i$ are faulty and the errors are too many to be corrected. Fault-tolerant FFT has also been studied in the literature [125, 263].

Equation (10.6), for a particular time instant t, has exactly the same form of the real-number decoding problem in Section 2.2.5. From Lemma 2.2.5, if we view t as the time that the fault-detection algorithm is implemented, we readily obtain the following result.

Theorem 10.3.1. *As long as there exists a matrix $\mathbf{H}_{p\times m}$ such that (i) the restricted isometry properties* (2.15) *and* (2.16) *are satisfied; (ii) the condition* (2.17) *in Lemma 2.2.5 is satisfied; (iii) $\mathbf{H}_{p\times m}\mathbf{G}_{m\times n} = \mathbf{0}$, the signal vector $\hat{\mathbf{x}}(t)$ in* (10.6) *for a particular time t, can be recovered exactly using an ℓ_1 minimization program*

$$\hat{\mathbf{x}}(t) = \arg\min_{\mathbf{g}\in\mathbb{R}^n} \|\mathbf{y}(t) - \mathbf{G}\mathbf{g}\|_{\ell_1}, \tag{10.7}$$

provided that the cardinality of the support of the error vector S is within some constant fraction α of m.

The constant α for random Gaussian matrices is determined by solving an inequality in [44]. Readers are referred to equation (3.23) in [44] for details (note that the constant α is written as r in [44]). The theoretical bound on α is conservative, while simulations often provide much better results (see [44] and Section 10.4 of this chapter).

From Theorem 10.3.1, using $\hat{\mathbf{x}}(t)$ and $\mathbf{y}(t)$ for a specific time instant t, we can recover the error vector $\mathbf{e}(t)$. By examining the support of $\mathbf{e}(t)$, we obtain the locations of the faulty filters. Suppose the support of $\mathbf{e}(t)$, i.e., the set of obtained indexes of the faulty filters is $\mathcal{U} = \{i_1, i_2, \ldots, i_k\} \subset \{1, 2, \ldots, m\}$. Denote by $\mathbf{H}_{\mathcal{U}}$ the submatrix of $\mathbf{H}_{p\times m}$ that is composed of the columns with indexes in $\mathcal{U}$, and denote by $\mathbf{H}_{\bar{\mathcal{U}}}$ the submatrix of $\mathbf{H}_{p\times m}$ that is composed of the columns with indexes in $\bar{\mathcal{U}} = \{1, 2, \ldots, m\} \setminus \mathcal{U}$. Denote by $\mathbf{y}_{\mathcal{U}}(t)$ the subvector of $\mathbf{y}(t)$ composed of the entries with indexes in $\mathcal{U}$, and denote by $\mathbf{y}_{\bar{\mathcal{U}}}(t)$ the subvector of $\mathbf{y}(t)$ composed of the entries with indexes in $\bar{\mathcal{U}}$. Then, $\mathbf{y}_{\bar{\mathcal{U}}}(t)$ is the correct output, while $\mathbf{y}_{\mathcal{U}}(t)$ contains errors. By solving the following equation,

$$(\mathbf{H}_{\mathcal{U}})_{p\times k}\hat{\mathbf{y}}_{\mathcal{U}}^{\top}(t) + \mathbf{H}_{\bar{\mathcal{U}}}\mathbf{y}_{\bar{\mathcal{U}}}^{\top}(t) = \mathbf{0}, \tag{10.8}$$

we obtain the fault-free output $[\hat{\mathbf{y}}_{\mathcal{U}}(t), \mathbf{y}_{\bar{\mathcal{U}}}(t)]$. From the condition of Theorem 10.3.1, $(\mathbf{H}_{\mathcal{U}})_{p\times k}$ satisfies $k = |\mathcal{U}| < S$, and hence $\mathbf{H}_{\mathcal{U}}$ has full column rank (see the RIP assumption (**A.1**)). Thus, we can always find a $k \times k$ submatrix $(\mathbf{K})_{k\times k}$ of $(\mathbf{H}_{\mathcal{U}})_{p\times k}$. Suppose the corresponding rows of $(\mathbf{K})_{k\times k}$ in $(\mathbf{H}_{\mathcal{U}})_{p\times k}$ corresponds to the row set $\mathcal{T}$. Then,

$$\hat{\mathbf{y}}_{\mathcal{U}}^{\top}(t) = \mathbf{K}^{-1}(\mathbf{H}_{\bar{\mathcal{U}}})_{\mathcal{T}}\mathbf{y}_{\bar{\mathcal{U}}}^{\top}(t), \tag{10.9}$$

where $(\mathbf{H}_{\bar{\mathcal{U}}})_{\mathcal{T}}$ is the submatrix of $\mathbf{H}_{\bar{\mathcal{U}}}$ with the row set $\mathcal{T}$.

Remark 28. The performance of the proposed encoded filtering scheme depends on the design of the sensing matrix $\mathbf{H}_{p\times m}$ (often called the parity check matrix in coding theory), which is shown to satisfy the RIP constraints (A.1) and (A.2) [44]. In our simulations, we use random Gaussian matrices to generate $\mathbf{H}_{p\times m}$ and compute the systematic form of the annihilating matrix $\mathbf{G}_{m\times n}$. At the first glance, it may seem that the parity check matrix of a BCH code, i.e., a Vandermonde matrix is a good choice, because any p columns of a p-by-m Vandermonde matrix are linearly independent (and hence the LHS of the p-RIP condition (2.15) is satisfied for some constant δ_k), and the real-number BCH codes have been successfully designed [266]. However, it is reported in [11] that the constant δ_k approaches 1 quickly as then ratio k/n decreases, and k-by-k submatrices of a Vandermonde matrix suffer from numerical issues.

10.4 Simulation results

The number of original filters is $n = 100$. These filters are coded into $m = n/R$ filters using a real-number linear code with a parity check matrix generated from random Gaussian matrices. The received signal $r(t)$ is composed of $n = 100$ different sinusoidal components with coefficients $x_i, 1 \leq i \leq 100$ (which is like modulation of many delta-signals with sinusoids) and is sampled into a vector with length 10^5. The coefficients x_i of different sinusoids (the delta-signals) are recovered by using matched filters, i.e., the original filters. However, some randomly chosen matched filters are faulty and may generate arbitrary outputs. For simplicity, in this simulation, we assume that the output of a faulty filter takes the form $\hat{x}_i + e_i$, where $\hat{x}_i$ is the correct coefficient to be computed and e_i is an additive noise term. We abuse notation and use signal-to-noise ratio (SNR) to denote $\frac{\mathbb{E}[\hat{x}_i^2]}{\mathbb{E}[e_i^2]}$. We use $\log$-SNR in simulations.

In Figure 10.2 and 10.3, we present simulation results which show the relationship between successful detections (defined as detecting and identifying all faulty filters) and the total number of faulty filters. The two simulation results show two promising properties of the coded filters: (1) the higher the redundancy, the higher the ratio of fault-tolerance; (2) fault-tolerance is insensitive to the statistical properties of noise. To obtain a single data point, we run 50 simulations with distinct random failures and distinct random Gaussian matrices.

10.5 Conclusions and future directions

In this chapter, we consider the problem of designing reliable real-number parallel filters using noisy hardware. We show that by using linear-programming decoding, we can tolerate a large amount of faulty filters. An interesting result to be investigated is whether linear programming decoding can be implemented even if the decoding computation components themselves suffer from failures. Apart from linear programming decoding,

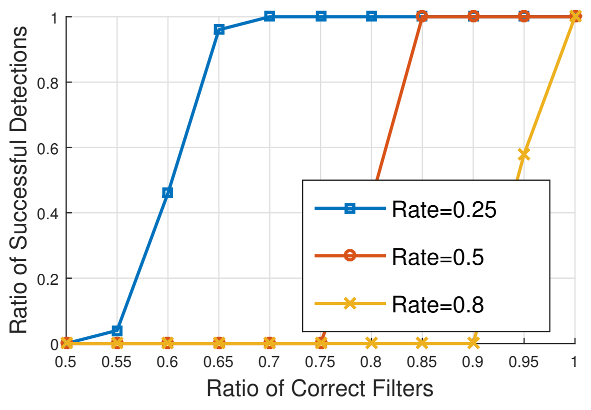


Figure 10.2: These are the simulation results for coded filters with different coding rates. Coded filters with higher redundancy can tolerate a higher ratio of faulty filters.

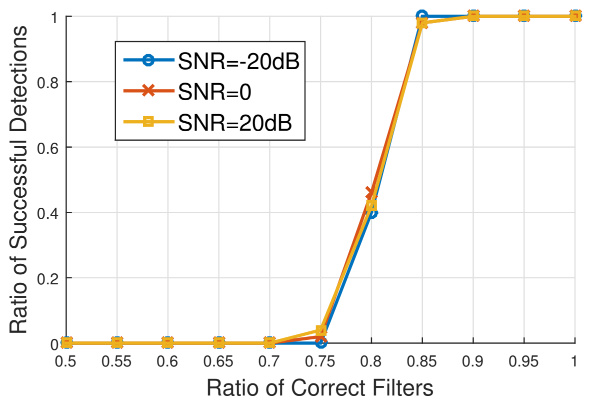


Figure 10.3: These are the simulation results for coded filters with different magnitudes faults when $R = 1/2$. The simulation results are insensitive to the fault magnitudes.

some other works on real-number error control coding are also applicable, such as real-number BCH codes [106] and LDPC codes [235]. A thorough performance comparison between these different codes is certainly of interest.

The parallel digital filtering model in this chapter (defined by (10.1)) can be adapted to different multi-user communication systems other than frequency division multiplexing. One example is the zero-forcing beamforming in a multiple access channel, which requires the projection of the received signal to different directions in the signal space. Coded linear filtering can be used in this problem to detect if some projections are faulty. This problem will be addressed in future works.

Chapter 11

Concluding remarks and future directions

This thesis develops the methodology, algorithms, and analytical tools of coding-based robust computation techniques. As we have mentioned in the motivation, robustness is a timeless issue, and it connects tightly with the overall performance of distributed systems. For example, one may reduce the energy consumption of circuits by deliberately sacrificing the robustness of each computation component (see Chapter 3). Another example is that by addressing the straggler's problem, one can reduce the overall computation time. From this perspective, the thesis is aimed to pave the way for understanding the interaction between information-theoretic techniques that are suitable for robustness and performance of today's computation systems. In this section, instead of reviewing the methods that we have discussed in each chapter, we would like to briefly overview the critical questions that we have tried to address during the thesis and provide the mental map again on the central theme of the thesis.

Can reliable computation be performed using entirely unreliable logic gates? Note that the error probability of the output is always lower-bounded by the last gate's error probability ϵ. Therefore, we can only define *reliability* as being able to make the output error probability close to ϵ (which we bound by 2ϵ in the ENCODED scheme in Chapter 3). In [253], it was shown that any function computable by l logic gates can be computed using $\mathcal{O}(l \log l)$ unreliable logic gates. One may think this has provided a satisfying answer to the above question. However, a $\log l$ factor in the increase of logic gates does not make sense in the saturation of Moore's law: if the entire semiconductor industry has tried their best to shrink the size of each transistor, how can we introduce such scaling-sense larger redundancy? The answer provided by the paper [278] is more satisfying in that a constant factor can bound the number of redundant computations. However, all of these techniques cannot make a real impact unless we can experimentally model the power-reliability tradeoffs of voltage scaling to give more insights to the designer. It would be useful to connect to the study on energy consumption on wiring and decoding [98, 99] as well, to choose the best codes for noisy decoding and observe in experiments if the predicted gains due to coding can indeed be achieved. For example, in [97], a new measure of the efficiency of circuits is proposed, which concerns the wiring length. Perhaps the final answer to the question is a circuit-level implementation that can showcase the energy reduction in computation while not sacrificing the accuracy of computation results.

The problem of computation with noisy gates is of considerable practical and intellectual interest. It is widely accepted that biological systems operate with noisy computational elements, and yet provide excellent performance at low energy. In engineered systems, with the saturation of Dennard's scaling and Moore's law, new device technologies are being used to design circuits that are invariably error-prone. A comprehensive understanding of reliability-resource tradeoffs in error-correction coding in computing could give these novel technologies (e.g. carbon nanotubes and mechanical switches) a better chance to compete with established ones (i.e., CMOS). To that end, it will be crucial to identify what causes faults in these novel technologies so that they can be modeled and analyzed, and appropriate codes are designed for them. Intellectually, it is interesting (and widely acknowledged) that the remarkable gains that coding brings to communications, especially at long-range, are not easy to obtain in computational settings. The theoretical reasoning for this thus far rests on simplistic models and has slightly loose bounds [96]. Improved strategies and improved outer limits will go a long way in characterizing how significant these gains can be.

We would also like to remark on the applicability of the gate-level error model in today's large scale systems. The current trend in coded computing focuses on "processor-level" (rather than gate-level) noise, e.g. it is assumed in many works that the product of input s with each column of $\mathbf{A}$ is "erasure-prone". However, there is an increasing trend in distributed systems community to consider "soft-errors" that are undetectable [46]. More importantly, there is an increasing need for understanding scalability when the number of (fixed memory) processors increases *for a fixed total problem size* (to understand the limits of gains with parallelization of a problem). This is called "strong scaling" [127, Chapter 9], whereas "weak scaling" allows for increasing problem size and number of processors while keeping the memory of each processor fixed. When the number of processors increases to the level that each processor, with a small amount of memory, only takes care of a small amount of work, it becomes more useful to consider the problem in the gate-level model: for strong scaling with soft errors, errors will accumulate and cause the resulting output to be far from the correct output.

Since we are talking about the error accumulation in computation systems, the natural question to ask is what is the suitable measure of error accumulation in computation systems. In Chapter 4, we showed that one such measure for the problem of data summarization and network consensus in the problem of multi-stage computing is the accumulation of distortion due to successive quantizations. We have proved theoretical bounds that are infinitely tighter than classical cut-set based bounds using this new measure of information loss. It is also interesting to investigate the generalization of the distortion accumulation effect and the inequalities developed in Chapter 4 to other computation and inference problems, especially in a computation DAG (directed acyclic graph) and in the case when data is not stored at all nodes. In fact, one can quickly obtain loose upper bounds for simple non-tree networks. For instance, for an achievable distortion bound in non-tree networks, a simple extension could be to the case of a DAG with only one source node with message $\mathbf{x}$ and only two paths to the sink node. In this case, if the mean-square error on one path is D_1 and the mean-square error on the other path is D_2, an achievable (if suboptimal) variance of estimating the source message using these two messages is $\min\{D_1, D_2\}$. This is achieved by either choosing the first message or the second message, and equality is achieved when the two messages are the same. Therefore, one can obtain (loose) upper bounds on the accumulation of distortion using our achievability results. However, because of apparent looseness in the bound, we may not achieve an asymptotically tight result, as we obtained in Theorem 4.3.1. Another new direction is the possible extension of distortion accumulation to non-Gaussian vectors using the Wasserstein distance as a distance metric [199], although we suspect that a simple form of distortion accumulation may not be easily obtained.

A crucial problem that is not considered in Chapter 4 is that sources at different nodes can be correlated. We expect that the concept of graph entropy will play a vital role in the correlated-source version of our problem [189]. We believe that the distortion accumulation phenomenon discussed here brings out a complementary and different issue that also needs to be understood for this comprehensive understanding (Theorem 4.3.1): even when all information sources are independent, the distortion of computing and compressing intermediate results *must* accumulate along the tree. This is mainly due to the new information introduced at intermediate nodes that needs to be incorporated into the intermediate results as the distributed computing proceeds along the edges of the tree network. [81] extends the notion of conditional graph entropy to distributed computing on tree networks, demonstrating that it is necessary to understand this concept to obtain a comprehensive information theory of distributed computing. However, in Chapter 4, the problem that we consider is fundamentally different: we effectively consider multi-stage computation, and the distortion accumulation effect is due to the successive quantization in multiple stages. We believe that there is a need to study examples that incorporate both distortion accumulation and graph entropy but in multi-stage problems.

Thus, we brought out the third question: what things can change if we consider computation problems that go beyond one stage? E.g., what if we consider iterative computing problems such as gradient descent and PageRank? In fact, the results that we have presented in Chapter 6, 7, and 5 are all about utilizing multi-stage or iterative computing problems to obtain a better design of coding-based algorithms. In Chapter 6, we show that by treating the intermediate results at stragglers that have not converged as additive noise, we can incorporate these results into the decoding of final results and reduce the computation error by orders of magnitude. In Chapter 7, we show that by carefully incorporating the intermediate computation result from the previous iteration, we can improve the number of tolerable erasures of sparse error-correcting codes by orders of magnitude. These two chapters point out that for the problem of iterative computing, there is usually a *good structure* that one can use to improve the performance of error correction. Essentially, this is because the problems that we considered in the thesis are mostly about *contraction* systems. One may argue that combining information in multi-modal or non-convex problems may not achieve the expected benefits in contraction systems anymore. However, as the practitioners have shown in the case of federated learning [138, 172], even for complicated non-convex problems, combining the learned models by simple averaging can still provide good convergence results. Thus, it is promising that we can extend the principles in Chapter 6 and 7 to more general settings. In Chapter 5, we show that when considering the problem of elastic computing, we have the new opportunity of designing adaptive schemes that can flexibly change the configuration when elastic events happen. This benefit cannot be achieved in one-stage computing problems, because if we consider one-stage problems where nodes can be taken away, we face the same problem as machine failures. Therefore, considering multi-stage problems can also lead to novel problems and new understandings of computation platforms. In fact, Chapter 3 and 4 are also closely related to multi-stage problems. In Chapter 3, we essentially show that by partitioning a one-stage problem into multiple stages, we can repeatedly suppress errors caused at the gate level, and make the computation with entirely fault-prone components reliable. Thus, the principle of considering multi-stage problems can also bring benefits in the reversed way in that, even if we consider single-stage problems, we can transform them into a multi-stage version and bring in new techniques and improvements. Finally, in Chapter 4, we show that by quantifying the distortion accumulation in a network with multiple stages, we can hope to obtain information-theoretical lower bounds that are infinitely tighter than existing ones, thus providing a better understanding of information propagation in distributed computing problems. Therefore, we have seen from multiple angles that the multi-stage computing can bring in new problem formulations and techniques to the robust computation literature, and it is of critical importance to understand it.

We want to end this thesis by going back to the fundamental physics perspective and the Landauer's principle (see Chapter 0). Of course, the principle of Landauer and the result on the minimum energy consumption for erasing a bit of information [147] is not for deriving any immediately practical bound on the energy consumption of today's computation systems (it was mainly for understanding Maxwell's demon). However, the principle does suggest another interesting point on the understanding of computing systems, which is that it is often useful to connect a computation problem to an information-theoretic issue and provide achievable schemes and fundamental limits in the transformed problem. Another simple example is the comparison-based lower bound [33] on sorting algorithms. The central theme of this thesis is similar, which is that the fundamental understanding of computing systems can be obtained through information-theoretic techniques and analytical frameworks. However, computation problems, even the simplest ones on matrix operations, are far beyond the scope of classical information theory. Therefore, it is useful to put in some effort to extend the method of information processing and coding to that of computation. And last but not least, it is always helpful to connect these results to real-system implementations and new computation platforms, as we have shown in Chapter 5.

Chapter 12

Bibliography

[1] Soft errors in electronic memory – a white paper. 2.3.3

[2] AWS Spot Instances. `https://aws.amazon.com/ec2/spot/`, 2018. 1.2.1

[3] AWS Spot Instances Prices. `https://aws.amazon.com/ec2/spot/pricing/`, 2018. 1.2.1

[4] Azure Batch. `https://docs.microsoft.com/en-us/azure/batch/batch-low-pri-vms`, 2018. 1.2.1

[5] Azure Batch Pricing. `https://azure.microsoft.com/en-us/pricing/details/batch/`, 2018. 1.2.1

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016. 5.1

[7] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2018. 1.2.2

[8] R. A. Abdallah and N. R. Shanbhag. An energy-efficient ecg processor in 45-nm cmos using statistical error compensation. *IEEE Journal of Solid-State Circuits*, 48(11):2882–2893, 2013. 2.3.9

[9] M. F. Aktas, P. Peng, and E. Soljanin. Effective straggler mitigation: Which clones should attack and when? *arXiv preprint arXiv:1710.00748*, 2017. 1.1, 2.3.1

[10] M. F. Aktas, P. Peng, and E. Soljanin. Straggler mitigation by delayed relaunch of tasks. *arXiv preprint arXiv:1710.00414*, 2017. 1.1, 2.3.1, 2.3.3

[11] A. Amini and F. Marvasti. Deterministic construction of binary, bipolar, and ternary compressed sensing matrices. *IEEE Transactions on Information Theory*, 57(4):2360–2370, 4 2011. 28

[12] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Effective straggler mitigation: Attack of the clones. In *NSDI*, volume 13, pages 185–198, 2013. 1.1, 2.3.1

[13] G. Ananthanarayanan, S. Kandula, A. G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in map-reduce clusters using mantri. In *Osdi*, volume 10, page 24, 2010. 0

[14] V. Anantharam, A. Gohari, S. Kamath, and C. Nair. On hypercontractivity and a data processing inequality. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 3022–3026, 6 2014. 2.3.11

[15] C. Anfinson and F. Luk. A linear algebraic model of algorithm-based fault tolerance. *IEEE Transactions on Computers*, 37(12):1599–1604, 12 1988. 2.3.2

[16] R. Appuswamy and M. Franceschetti. Computing linear functions by linear coding over networks. *IEEE Transactions on Information Theory*, 60(1):422–431, 1 2014. 2.3.12

[17] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger. Network coding for computing: Cut-set bounds. *IEEE Transactions on Information Theory*, 57(2):1015–1030, 2011. 2.3.12

[18] M. A. Attia and R. Tandon. Near optimal coded data shuffling for distributed learning. *arXiv preprint arXiv:1801.01875*, 2018. 2.3.3

[19] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran. Straggler-proofing massive-scale distributed matrix multiplication with d-dimensional product codes. In *IEEE International Symposium on Information Theory (ISIT)*, pages 1993–1997, 2018. 2.3.3

[20] M. Bakshi, S. Jaggi, S. Cai, and M. Chen. SHO-FA: Robust compressive sensing with order-optimal complexity, measurements, and bits. In *50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 786–793, 2012. 2.3.7

[21] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications*, 32(3):866–901, 2011. 2.3.6

[22] S. Banerjee, P. Gupta, and S. Shakkottai. Towards a queueing-based framework for in-network function computation. *Queueing Systems*, 72(3-4):219–250, 2012. 2.3.12

[23] M. Barnett, L. Shuler, R. van De Geijn, S. Gupta, D. G. Payne, and J. Watts. Interprocessor collective communication library (intercom). In *Proceedings of IEEE Scalable High Performance Computing Conference*, pages 357–364. IEEE, 1994. 5.4.3

[24] S. Basu, V. Saxena, R. Panja, and A. Verma. Balancing stragglers against staleness in distributed deep learning. In *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, pages 12–21. IEEE, 2018. 0

[25] F. L. Bauer. Das verfahren der treppeniteration und verwandte verfahren zur lösung algebraischer eigenwertprobleme. *Zeitschrift für angewandte Mathematik und Physik ZAMP*, 8(3):214–235, 1957. 7.4.1

[26] Bergman et al. Exascale computing study: Technology challenges in achieving exascale systems. *DARPA IPTO*, 2008. 1.2.1, 2.3.3

[27] M. W. Berry, D. Mezher, B. Philippe, and A. Sameh. Parallel algorithms for the singular value decomposition. *Statistics Textbooks and Monographs*, 184(117):31,

2006. 2.3.5, 7.4.1

[28] D. Bertozzi, L. Benini, and G. De Micheli. Error control schemes for on-chip communication links: the energy-reliability tradeoff. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(6):818–831, 6 2005. 2.3.7

[29] C. M. Bishop. Pattern recognition. *Machine Learning*, 2006. 9.4.1

[30] R. Bitar, P. Parag, and S. El Rouayheb. Minimizing latency for secure distributed computing. In *IEEE International Symposium on Information Theory (ISIT)*, pages 2900–2904, 2017. 2.3.3

[31] C. G. Blake and F. R. Kschischang. Energy of decoding algorithms. In *13th Canadian Workshop on Information Theory (CWIT)*, pages 1–5, 6 2013. 2.3.10

[32] C. G. Blake and F. R. Kschischang. Energy consumption of vlsi decoders. *Information Theory, IEEE Transactions on*, 61(6):3185–3198, 6 2015. 2.3.10

[33] A. Blum. Comparison-based lower bounds for sorting, 2011. 11

[34] B. Bollobás. Random graphs. In *Modern Graph Theory*, volume 184 of *Graduate Texts in Mathematics*, pages 215–252. Springer New York, 1998. 8.1.1, 8.2.2

[35] J. Bolz, I. Farmer, E. Grinspun, and P. Schröoder. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. In *ACM Transactions on Graphics*, volume 22, pages 917–924. ACM, 2003. 6.1

[36] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005. (document), 0, 1.2, 3.1

[37] A. Bouteiller, T. Herault, G. Bosilca, P. Du, and J. Dongarra. Algorithm-based fault tolerance for dense matrix factorizations, multiple failures and accuracy. *ACM Transactions on Parallel Computing*, 1(2):10, 2015. 2.3.2

[38] K. Bowman, J. Tschanz, C. Wilkerson, T. K. S.-L. Lu, V. De, and S. Borkar. Circuit techniques for dynamic variation tolerance. In *Proceedings of the 46th ACM Annual Design Automation Conference*, pages 4–7, 2007. 2.3.9

[39] D. Burshtein. On the error correction of regular LDPC codes using the flipping algorithm. *IEEE Trans. Inf. Theory*, 54(2):517–530, 2008. 2.2.2, 3.2.3, A.3, A.3, A.3.1, A.3, A.3

[40] D. Burshtein and G. Miller. Expander graph arguments for message-passing algorithms. *IEEE Trans. Inf. Theory*, 47(2):782–790, 2001. 2

[41] W. H. Butler, T. Mewes, C. K. A. Mewes, P. B. Visscher, W. H. Rippard, S. E. Russek, and R. Heindl. Switching distributions for perpendicular spin-torque devices within the macrospin approximation. *IEEE Trans. Magn.*, 48(12):4684–4700, 2012. 3.1

[42] F. Calmon, Y. Polyanskiy, and Y. Wu. Strong data processing inequalities in power-constrained Gaussian channels. In *Proceedings of the IEEE International Symposium on Information Theory*, 2015. 2.3.11

[43] E. Candes and P. Randall. Highly robust error correction by convex programming. *IEEE Transactions on Information Theory*, 54(7):2829–2840, 7 2008. 2.3.7, 10.1

[44] E. Candes and T. Tao. Decoding by linear programming. *IEEE Transactions on Information Theory*, 51(12):4203–4215, 12 2005. 2.2.5, 2.2.5, 2.2.5, 2.2.5, 2.3.7, 10.1, 10.3, 28

[45] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson. Randomness conductors and constant-degree lossless expanders. In *Proc. 34th ACM Symp. Theory Comput.*, pages 659–668. ACM, 2002. 2

[46] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1):5–28, 2014. 11

[47] Z. Charles and D. Papailiopoulos. Gradient coding using the stochastic block model. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1998–2002. IEEE, 2018. (document), 1.3.2, 2.3.4, 7.5.4, 7.12, 7.5.4

[48] L. Chen, J. Xu, I. Djurdjevic, and S. Lin. Near-shannon-limit quasi-cyclic low-density parity-check codes. *IEEE Transactions on Communications*, 52(7):1038–1042, 2004. 23

[49] S. Chen, R. Varma, A. Sandryhaila, and J. Kovačević. Discrete signal processing on graphs: Sampling theory. *IEEE Transactions on Signal Processing*, 63(24):6510–6523, 2015. 2.3.5, 6.2.3

[50] S. Chen, Y. Yang, C. Faloutsos, and J. Kovacevic. Monitoring manhattan's traffic at 5 intersections? In *Global Conference on Signal and Information Processing (GlobalSIP)*, 2016. 6.2.3

[51] Z. Chen. Optimal real number codes for fault tolerant matrix operations. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 29:1–29:10, New York, NY, USA, 2009. ACM. 2.3.2

[52] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pages 493–507, 1952. 8.4.3, A.1, A.1

[53] S. K. Chilappagari and B. Vasic. Fault tolerant memories based on expander graphs. In *Proceedings of IEEE Information Theory Workshop (ITW)*, 2007. 2.3.8

[54] J. W. Choi, B. Shim, A. Singer, and N. I. Cho. Low-power filtering via minimum power soft error cancellation. *IEEE Transactions on Signal Processing*, 55(10):5084–5096, 10 2007. 2.3.9

[55] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. *ACM SIGCOMM Computer Communication Review*, 41(4):98–109, 2011. 2.3.3

[56] B.-G. Chun, T. Condie, Y. Chen, B. Cho, A. Chung, C. Curino, C. Douglas, M. Interlandi, B. Jeon, J. S. Jeong, G. Lee, Y. Lee, T. Majestro, D. Malkhi, S. Matusevych, B. Myers, M. Mykhailova, S. Narayanamurthy, J. Noor, R. Ramakrishnan, S. Rao, R. Sears, B. Sezgin, T. Um, J. Wang, M. Weimer, and Y. Yang. Apache REEF: Retain-

able evaluator execution framework. *ACM Trans. Comput. Syst.*, 35(2):5:1–5:31, Oct. 2017. 1.3.1, 5.5

[57] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9(3):251–280, 1990. 6.3.5

[58] T. H. Cormen. *Introduction to algorithms*. MIT press, 2009. 8.2.2

[59] T. M. Cover and J. A. Thomas. *Elements of Information Theory, 2nd Edition*. John Wiley & Sons, 2006. 4.1, 4.3.2, 8, 9, B.2.1

[60] P. Cuff, H.-I. Su, and A. El Gamal. Cascade multiterminal source coding. In *IEEE International Symposium on Information Theory*, pages 1199–1203. IEEE, 2009. 1.1, 1.3.3, 2.3.12, 4.1, 4.3.3

[61] R. Danilak. Why energy is a big and rapidly growing problem for data centers, 2017. 1.2.2

[62] J. Dean. Software engineering advice from building large-scale distributed systems. *CS295 Lecture at Stanford University, July*, 2007. 1.2.1, 1.2.2, 2.3.3

[63] J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013. 1.2.1, 2.3.1, 2.3.3

[64] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential qr and lu factorizations. *SIAM Journal on Scientific Computing*, 34(1):A206–A239, 2012. 2.3.6

[65] R. H. Dennard, V. L. Rideout, E. Bassous, and A. R. Leblanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974. 1.2.1, 3.1

[66] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *IEEE Transactions on Information Theory*, 56(9):4539–4551, 2010. 1.2.3

[67] A. G. Dimakis, S. Kar, J. M. F. Moura, M. G. Rabbat, and A. Scaglione. Gossip algorithms for distributed signal processing. *Proceedings of the IEEE*, 98(11):1847–1864, 11 2010. 2.3.12

[68] C. Ding, C. Karlsson, H. Liu, T. Davies, and Z. Chen. Matrix multiplication on GPUs with on-line fault tolerance. In *Proceedings of the IEEE 9th International Symposium on Parallel and Distributed Processing with Applications*, pages 311–317, 5 2011. 2.3.2

[69] D. Divsalar, S. Dolinar, C. R. Jones, and K. Andrews. Capacity-approaching protograph codes. *IEEE Journal on Selected Areas in Communications*, 27(6):876–888, 2009. 25

[70] R. L. V. Dobrushin and S. I. Ortyukov. Upper bound on the redundancy of self-correcting arrangements of unreliable functional elements. *Probl. Inf. Transm.*, 13(3):56–76, 1977. 1.1, 2.3.7

[71] E. Dupraz, D. Declercq, B. Vasic, and V. Savin. Analysis and design of finite

alphabet iterative decoders robust to faulty hardware. *Communications, IEEE Transactions on*, 63(8):2797–2809, 2015. 2.3.8

[72] C. Dutta, Y. Kanoria, D. Manjunath, and J. Radhakrishnan. A tight lower bound for parity in noisy communication networks. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 1056–1065, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics. 2.3.12

[73] S. Dutta, Z. Bai, T. M. Low, and P. Grover. Codenet: Training large neural networks in presence of soft-errors. *Submitted*, 2018. 2.3.3

[74] S. Dutta, V. Cadambe, and P. Grover. Short-dot: Computing large linear transforms distributedly using coded short dot products. In *Advances In Neural Information Processing Systems*, pages 2092–2100, 2016. 2.3.3

[75] S. Dutta, V. Cadambe, and P. Grover. Coded convolution for parallel and distributed computing within a deadline. In *IEEE International Symposium on Information Theory (ISIT)*, pages 2403–2407, 2017. 2.3.3

[76] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD. *The 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018. 2.3.5

[77] E. Erkip and T. Cover. The efficiency of investment information. *IEEE Transactions on Information Theory*, 44(3):1026–1040, 5 1998. 2.3.11

[78] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. In *Proceedings of the 36th Annual IEEE/ACM International Symposium in Microarchitecture*, pages 7–18, 12 2003. 3.5

[79] W. S. Evans and L. J. Schulman. Signal propagation and noisy circuits. *IEEE Transactions on Information Theory*, 45(7):2367–2373, 1999. 0, 1.2.3, 2.3.11

[80] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover. On the optimal recovery threshold of coded matrix multiplication. In *55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1264–1270, 2017. 2.3.3

[81] S. Feizi and M. Médard. On network functional compression. *IEEE Transactions on Information Theory*, 60(9):5387–5401, 2014. 2.3.12, 11

[82] J. Feldman, T. Malkin, R. A. Servedio, C. Stein, and M. J. Wainwright. LP decoding corrects a constant fraction of errors. *IEEE Trans. Inf. Theory*, 53(1):82–89, 2007. 2

[83] N. S. Ferdinand and S. C. Draper. Anytime coding for distributed computation. In *Communication, Control, and Computing (Allerton)*, pages 954–960, 2016. 2.3.3

[84] K. Ferreira, J. Stearley, J. H. Laros III, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold. Evaluating the viability of process replication reliability for exascale systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 44. ACM, 2011. 2.3.1

[85] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28(13):2009, 2009. 1.2.1

[86] R. Gallager. Finding parity in a simple broadcast network. *IEEE Transactions on Information Theory*, 34(2):176–180, 3 1988. 1.3.3, 2.3.12, 8.1, 8.1.2, 8.3, 3

[87] R. G. Gallager. Low-density parity-check codes. *IRE Transactions on Information Theory*, 8(1):21–28, 1 1962. 2.2.2, 1, 2.2.2, 2.2.1, 2.3.8, 3.2.3

[88] R. G. Gallager. *Information theory and reliable communication*. John Wiley & Sons, 1968. 2.2.3, 2.2.2, 2.2.3

[89] A. E. Gamal and Y.-H. Kim. *Network information theory*. Cambridge University Press, 2011. 8

[90] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, and E. Hyytia. Reducing latency via redundant requests: Exact analysis. *ACM SIGMETRICS Performance Evaluation Review*, 43(1):347–360, 2015. 1.1, 2.3.1

[91] A. Geist. How to kill a supercomputer: Dirty power, cosmic rays, and bad solder. *IEEE Spectrum*, 10, 2016. 2.3.3

[92] A. Geist. Supercomputing's monster in the closet. *IEEE Spectrum*, 53(3):30–35, 2016. 2.3.3

[93] A. Giridhar and P. Kumar. Toward a theory of in-network computation in wireless sensor networks. *IEEE Communications Magazine*, 44(4):98–107, 4 2006. 2.3.12

[94] G. H. Golub and C. F. Van Loan. *Matrix computations*, volume 3. JHU Press, 2012. 2.3.5, 7.4.1, D.3.1

[95] N. Goyal, G. Kindler, and M. Saks. Lower bounds for the noisy broadcast problem. *SIAM Journal on Computing*, 37(6):1806–1841, 2008. 2.3.12, 8.1.2, 8.3

[96] P. Grover. Is 'shannon-capacity of noisy computing' zero? In *Proceedings of IEEE International Symposium on Information Theory*, pages 2854–2858, 6 2014. 2.3.10, 11

[97] P. Grover. Information friction and its implications on minimum energy required for communication. *IEEE Transactions on Information Theory*, 61(2):895–907, 2 2015. 2.3.10, 11

[98] P. Grover, A. Goldsmith, and A. Sahai. Fundamental limits on the power consumption of encoding and decoding. In *Proceedings of IEEE International Symposium on Information Theory*, pages 2716–2720, 2012. 2.3.10, 11

[99] P. Grover, K. Woyach, and A. Sahai. Towards a communication-theoretic understanding of system-level power consumption. *IEEE Journal of Selected Areas in Communication (JSAC) Special Issue on Energy-Efficient Wireless Communications*, 29(8):1744–1755, 9 2011. 2.3.10, 11

[100] V. Gupta, S. Kadhe, T. Courtade, M. W. Mahoney, and K. Ramchandran. Oversketched newton: Fast convex optimization for serverless systems. *arXiv preprint*

arXiv:1903.08857, 2019. 1.1

[101] V. Gupta, S. Wang, T. Courtade, and K. Ramchandran. Oversketch: Approximate matrix multiplication for the cloud. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 298–304. IEEE, 2018. 1.2.1

[102] V. Guruswami, C. Umans, and S. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. *J. ACM*, 56(4):20, 2009. 2

[103] J. Hachem, I.-H. Wang, C. Fragouli, and S. Diggavi. Coding with encoding uncertainty. In *Proceedings of IEEE International Symposium on Information Theory Proceedings (ISIT)*, pages 276–280, 7 2013. 2.3.8

[104] F. Haddadpour, Y. Yang, V. R. Cadambe, and P. Grover. Cross-iteration coded computing. *Allerton*, 2018. 1.3.2, 2.3.3

[105] F. Haddadpour, Y. Yang, M. Chaudhari, V. R. Cadambe, and P. Grover. Straggler-resilient and communication-efficient distributed iterative linear solver. *arXiv preprint arXiv:1806.06140*, 2018. 1.3.2, 2.3.6

[106] C. Hadjicostis. Nonconcurrent error detection and correction in fault-tolerant discrete-time LTI dynamic systems. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 50(1):45–55, 1 2003. 10.5

[107] C. N. Hadjicostis. Non-concurrent error detection and correction in discrete-time LTI dynamic systems. In *Proceedings of the 40th IEEE Conference on Decision and Control*, volume 2, pages 1899–1904, 2001. 2.3.9

[108] C. N. Hadjicostis. Nonconcurrent error detection and correction in fault-tolerant linear finite-state machines. *IEEE Transactions on Automatic Control*, 48(12):2133–2140, 2003. 2.3.7

[109] C. N. Hadjicostis and G. C. Verghese. Coding approaches to fault tolerance in linear dynamic systems. *IEEE Trans. on Information Theory*, 51(1):210–228, 2005. 1.1, 1.2.3, 2.2.4, 2.3.7, 3.6

[110] W. Halbawi, N. Azizan-Ruhi, F. Salehi, and B. Hassibi. Improving distributed gradient descent using reed-solomon codes. *arXiv preprint arXiv:1706.05436*, 2017. 1.3.2, 1.3.2, 2.3.3, 2.3.4

[111] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011. 2.3.5, 7.4.1, 7.4.1

[112] I. S. Haque and V. S. Pande. Hard data on soft errors: A large-scale assessment of real-world error rates in GPGPU. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 691–696, 2010. 1.2.1, 2.3.3

[113] T. H. Haveliwala. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*, pages 517–526. ACM, 2002. 2.3.5, 6.2.3, 6.2.3

[114] C. Heinze, B. McWilliams, and N. Meinshausen. Dual-loco: Distributing statistical estimation using random projections. In *Artificial Intelligence and Statistics*, pages 875–883, 2016. 2.3.6

[115] T. Herault and Y. Robert. *Fault-Tolerance Techniques for High Performance Computing*. Springer, 2015. 1.1, 2.3.1

[116] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011. 1.2.1, 5.1

[117] C.-H. Huang, Y. Li, and L. Dolecek. Belief propagation algorithms on noisy hardware. *IEEE Transactions on Communications*, 63(1):11–24, 2015. 2.3.8

[118] K. H. Huang. Algorithm-based fault tolerance for matrix operations. *IEEE transactions on computers*, 100(6):518–528, 1984. 0, 1.2.3, 2.2.4, 2.3.2

[119] J. Jeon. A generalized typicality for abstract alphabets. *arXiv:1401.6728v4*, 2015. 4.4.1

[120] H. Jeong, C. G. Blake, and P. Grover. Energy-adaptive polar codes: Trading off reliability and decoder circuit energy. In *Information Theory (ISIT), 2017 IEEE International Symposium on*, pages 2608–2612. IEEE, 2017. 2.3.10

[121] H. Jeong, T. M. Low, and P. Grover. Masterless coded computing: A fully-distributed coded FFT algorithm. *Communications, Control and Computing (Allerton)*, 2018. 2.3.3

[122] H. Ji, S. H. Weinberg, M. Li, J. Wang, and Y. Li. An apache spark implementation of block power method for computing dominant eigenvalues and eigenvectors of large-scale matrices. In *IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)*, pages 554–559, 2016. 2.3.5, 7.4.1

[123] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht. Occupy the cloud: Distributed computing for the 99%. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 445–451. ACM, 2017. 1.2.1

[124] C. Joo and N. Shroff. On the delay performance of in-network aggregation in lossy wireless sensor networks. *Networking, IEEE/ACM Transactions on*, 22(2):662–673, 4 2014. 2.3.12

[125] J.-Y. Jou and J. Abraham. Fault-tolerant FFT networks. *IEEE Transactions on Computers*, 37(5):548–561, 5 1988. 2.3.2, 27

[126] S. Kamath, D. Manjunath, and R. Mazumdar. On distributed function computation in structure-free random wireless networks. *IEEE Transactions on Information Theory*, 60(1):432–442, 1 2014. 2.3.12

[127] A. Kaminsky. BIG CPU, BIG DATA: Solving the world's toughest computational problems with parallel computing. 2016. 11

[128] S. Kannan and P. Viswanath. Multi-session function computation and multicasting in undirected graphs. *IEEE Journal on Selected Areas in Communications*, 31(4):702–713, 4 2013. 2.3.12

[129] Y. Kanoria and D. Manjunath. On distributed computation in noisy random planar networks. In *Proceedings of the 2007 IEEE International Symposium on Information*

Theory (ISIT), pages 626–630, 2007. 2.3.12

[130] C. Karakus, Y. Sun, and S. Diggavi. Encoded distributed optimization. In *Information Theory (ISIT), 2017 IEEE International Symposium on*, pages 2890–2894. IEEE, 2017. 2.3.3, 2.3.4

[131] C. Karakus, Y. Sun, S. Diggavi, and W. Yin. Straggler mitigation in distributed optimization through data encoding. In *Neural Information Processing Systems*, pages 5440–5448, 2017. 2.3.3, 2.3.4, 2.3.5

[132] N. Karamchandani, R. Appuswamy, and M. Franceschetti. Time and energy complexity of function computation over networks. *IEEE Transactions on Information Theory*, 57(12):7671–7684, 12 2011. 2.3.12, 8.1.2

[133] D. Kempe and F. McSherry. A decentralized algorithm for spectral analysis. *J. Comput. Syst. Sci.*, 74(1):70–83, 2008. 2.3.5, 7.4.1

[134] N. Khude, A. Kumar, and A. Karnik. Time and energy complexity of distributed computation of a class of functions in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 7(5):617–632, 5 2008. 2.3.12

[135] J. Kim, A. Paul, P. A. Crowell, S. J. Koester, S. S. Sapatnekar, J.-P. Wang, and C. H. Kim. Spin-based computing: device concepts, current status, and a case study on a high-performance microprocessor. *Proc. IEEE*, 103(1):106–130, 2015. 3.1

[136] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 361–372. IEEE Press, 2014. 2.3.3

[137] D. E. Knuth. *The Art of Computer Programming, volume 1: Fundamental Algorithms Addison-Wesley*. Addison-Wesley Professional, 1997. 2.2

[138] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016. 11

[139] J. Kosaian, K. Rashmi, and S. Venkataraman. Learning a code: Machine learning for approximate non-linear coded computation. *arXiv preprint arXiv:1806.01259*, 2018. 2.3.3

[140] H. Kowshik and P. R. Kumar. Optimal function computation in directed and undirected graphs. *IEEE Transactions on Information Theory*, 58(6):3407–3418, 2012. 2.3.12

[141] D. Krithivasan and S. S. Pradhan. Lattices for distributed source coding: Jointly gaussian sources and reconstruction of a linear function. *IEEE Transactions on Information Theory*, 55(12):5628–5651, 2009. 2.3.12

[142] E. Kushilevitz. Communication complexity. *Advances in Computers*, 44:331–360, 1997. 1.1, 8.1

[143] E. Kushilevitz and Y. Mansour. Computation in noisy radio networks. In *Proceedings of the 9th annual ACM-SIAM symposium on Discrete Algorithms, Society for*

Industrial and Applied Mathematics, volume 98, pages 236–243, 1998. 2.3.12

[144] A. V. Kuznetsov. Information storage in a memory assembled from unreliable components. *Probl. Inf. Transm.*, 9(3):100–114, 1973. 2.3.8

[145] F. Labeau, J.-C. Chiang, M. Kieffer, P. Duhamel, L. Vandendorpe, and B. Macq. Oversampled filter banks as error correcting codes: theory and impulse noise correction. *IEEE Transactions on Signal Processing*, 53(12):4619–4630, 12 2005. 2.3.9

[146] R. Landauer. Irreversibility and heat generation in the computing process. *IBM journal of research and development*, 5(3):183–191, 1961. 0

[147] R. Landauer. Minimal energy requirements in communication. *Science*, 272(5270):1914–1918, 1996. 0, 11

[148] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran. Speeding up distributed machine learning using codes. In *IEEE International Symposium on Information Theory*, pages 1143–1147, 2016. 2.3.3

[149] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 2018. 2.1, 2.2.4

[150] K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran. Coded computation for multicore setups. In *IEEE International Symposium on Information Theory (ISIT)*, pages 2413–2417, 2017. 2.3.3

[151] K. Lee, C. Suh, and K. Ramchandran. High-dimensional coded matrix multiplication. In *IEEE International Symposium on Information Theory (ISIT)*, pages 2418–2422, 2017. 2.3.3

[152] M. Lentmaier, D. Truhachev, K. Zigangirov, and D. Costello. An analysis of the block error probability performance of iterative decoding. *IEEE Transactions on Information Theory*, 51(11):3834–3855, 11 2005. 2.2.2, 2.3.8

[153] J. Leskovec and J. J. Mcauley. Learning to discover social circles in ego networks. In *Advances in neural information processing systems*, pages 539–547, 2012. 6.4.1, 7.5.1

[154] C. Li and H. Dai. Efficient in-network computing with noisy wireless channels. *IEEE Transactions on Mobile Computing*, 12(11):2167–2177, 11 2013. 2.3.12, 8.1.2

[155] J. Li, N. K. Sharma, D. R. Ports, and S. D. Gribble. Tales of the tail: Hardware, os, and application-level sources of tail latency. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14. ACM, 2014. 1.2.1, 2.3.3

[156] S. Li, M. Maddah-Ali, Q. Yu, and A. S. Avestimehr. A fundamental tradeoff between computation and communication in distributed computing. *IEEE Transactions on Information Theory*, 64(1):109–128, 2018. 2.3.3, 2.3.6

[157] S. Li, S. Supittayapornpong, M. A. Maddah-Ali, and A. S. Avestimehr. Coded TeraSort. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 389–398, 2017. 2.3.3

[158] X. Li, S. Pawar, and K. Ramchandran. Sub-linear time compressed sensing using

sparse-graph codes. In *IEEE International Symposium on Information Theory (ISIT)*, pages 1645–1649. IEEE, 2015. 2.3.7

[159] X. Li, K. Shen, M. C. Huang, and L. Chu. A Memory Soft Error Measurement on Production Systems. In *USENIX Annual Technical Conference*, page 275–280, 2007. 2.3.3

[160] F. Lin and W. W. Cohen. Power iteration clustering. In *International Conference on Machine Learning*, pages 655–662, 2010. 2.3.5

[161] M. Luby. Lt codes. In *Proceedings of the 2002 Annual Symposium on Foundations of Computer Science (FOCS)*, pages 271–280, 2002. 8.1.1

[162] R. Lucas, J. Ang, K. Bergman, S. Borkar, W. Carlson, L. Carrington, G. Chiu, R. Colwell, W. Dally, and J. Dongarra. Top ten exascale research challenges. *DOE ASCAC subcommittee report*, pages 1–86, 2014. 1.2.1, 2.3.3

[163] R. E. Lyons and W. Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, 6(2):200–209, 1962. 1.1, 2.3.1

[164] C. Ma, V. Smith, M. Jaggi, M. I. Jordan, P. Richtárik, and M. Takáč. Adding vs. averaging in distributed primal-dual optimization. *arXiv preprint arXiv:1502.03508*, 2015. 2.3.6

[165] K. Mahajan, M. Chowdhury, A. Akella, and S. Chawla. Dynamic query re-planning using QOOP. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 253–267, Carlsbad, CA, 2018. USENIX Association. 1.2.1

[166] M. W. Mahoney et al. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning*, 3(2):123–224, 2011. 1.1

[167] R. K. Maity, A. S. Rawat, and A. Mazumdar. Robust gradient descent via moment encoding with LDPC codes. *SysML Conference*, 2018. 2.3.3

[168] A. Mallick, M. Chaudhari, and G. Joshi. Rateless Codes for Near-Perfect Load Balancing in Distributed Matrix-Vector Multiplication. *arXiv preprint arXiv:1804.10331*, 2018. 2.3.3

[169] S. Manipatruni, D. E. Nikonov, and I. A. Young. Modeling and design of spintronic integrated circuits. *IEEE Trans. Circuits Syst. I, Reg. Papers*, 59(12):2801–2814, 2012. 3.1

[170] D. Marco, E. J. Duarte-Melo, M. Liu, and D. L. Neuhoff. On the many-to-one transport capacity of a dense wireless sensor network and the compressibility of its data. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (ISPN 2003)*, pages 1–16. Springer, 2003. 2.3.12

[171] R. Mcdonald, M. Mohri, N. Silberman, D. Walker, and G. S. Mann. Efficient large-scale distributed training of conditional maximum entropy models. In *Advances in Neural Information Processing Systems*, pages 1231–1239, 2009. 2.3.6

[172] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, et al. Communication-efficient

learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016. 1.2.2, 11

[173] B. McWilliams, C. Heinze, N. Meinshausen, G. Krummenacher, and H. P. Vanchinathan. Loco: Distributing ridge regression with random projections. *stat*, 1050:26, 2014. 2.3.6

[174] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. Mllib: Machine learning in Apache Spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016. 5.1

[175] M. Miranda. The threat of semiconductor variability: As transistors shrink, the problem of chip variability grows. *IEEE Spectrum*, 28, 2012. 2.3.3

[176] V. Misra and K. Viswanathan. Sequential functional quantization. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 2359–2363, 7 2013. 2.3.12, 4.1.1

[177] A. M. Mood, F. A. Graybill, and D. C. Boes. Introduction to the theory of statistics, 3rd edition. 1974. D.3.1

[178] T. Moscibroda. The worst-case capacity of wireless sensor networks. In *Proceedings of the 6th International Symposium on Information Processing in Sensor Networks (ISPN 2007)*, pages 1–10, 4 2007. 2.3.12

[179] S. K. Narang, A. Gadde, E. Sanou, and A. Ortega. Localized iterative methods for interpolation in graph structured data. In *Global Conference on Signal and Information Processing (GlobalSIP)*, pages 491–494. IEEE, 2013. 2.3.5, 6.2.3

[180] S. Narayanamurthy, M. Weimer, D. Mahajan, T. Condie, S. Sellamanickam, and S. S. Keerthi. Towards resource-elastic machine learning. In *NIPS 2013 BigLearn Workshop*, 2013. 1.2.1, 2.3.1, 5.1

[181] R. Nathanael and T.-J. K. Liu. *CMOS and Beyond: Logic Switches for Terascale Integrated Circuits*, chapter 11 Mechanical switches. Cambridge University Press, 2014. 1.2.1

[182] B. Nazer and M. Gastpar. Compute-and-forward: Harnessing interference through structured codes. *IEEE Transactions on Information Theory*, 57(10):6463–6486, 2011. 2.3.12

[183] V. P. Nelson. Fault-tolerant computing: Fundamental concepts. *IEEE Computer*, 23(7):19–25, 1990. 2.3.3

[184] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013. 2.3.6

[185] J. V. Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34:43–98, 1956. 2.3.7

[186] I. Newman. Computing in fault tolerance broadcast networks. In *Proceedings of 19th IEEE Annual Conference on Computational Complexity*, pages 113–122, 6 2004. 2.3.12

[187] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002. 7.4.1, 7.5.2

[188] J. Nielsen. Nielsen's law of internet bandwidth. *Online at http://www. useit. com/alertbox/980405. html*, 1998. 1.2.2

[189] A. Orlitsky and J. R. Roche. Coding for computing. *IEEE Transactions on Information Theory*, 47(3):903–917, 2001. 0, 1.1, 11

[190] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999. 1.3.2, 2.3.5, 6.1, 6.2.3, 6.2.3

[191] N. Patil, J. Deng, A. Lin, H.-S. Wong, and S. Mitra. Design methods for misaligned and mispositioned carbon-nanotube immune circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1725–1736, 2008. 1.2.1

[192] M. Petraglia and S. Mitra. Fault tolerant adaptive filter structure based on the generalized subband decomposition of FIR filters. In *IEEE International Symposium on Circuits and Systems*, volume 2, pages 141–144, 5 1994. 2.3.9

[193] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. *ACM SIGOPS Operating Systems Review*, 35(5):89–102, 2001. 1.2.1

[194] N. Pippenger. On networks of noisy gates. In *26th IEEE Annual Symposium on Foundations of Computer Science*, pages 30–38, 1985. 1.1, 2.3.7

[195] N. Pippenger. Reliable computation by formulae in the presence of noise. *IEEE Transactions on Information Theory*, 34(2):194–197, March 1988. 1.2.3

[196] N. Pippenger, G. Stamoulis, and J. Tsitsiklis. On a lower bound for the redundancy of reliable networks with noisy gates. *IEEE Transactions on Information Theory*, 37(3):639–643, 5 1991. 2.3.7, A.2

[197] J. S. Plank, K. Li, and M. A. Puening. Diskless checkpointing. *IEEE Trans. on Parallel and Distributed Systems*, 9(10):972–986, 1998. 1.1, 2.3.1

[198] Y. Polyanskiy and Y. Wu. Dissipation of information in channels with input constraints. *IEEE Trans. Inf. Theory*, 62(1):35–55, 2016. 0, 1.2.3, 2.3.11, 4.1

[199] Y. Polyanskiy and Y. Wu. Wasserstein continuity of entropy and outer bounds for interference channels. *IEEE Transactions on Information Theory*, 62(7):3992–4002, 7 2016. 11

[200] B. A. Prakash, A. Sridharan, M. Seshadri, S. Machiraju, and C. Faloutsos. Eigenspokes: surprising patterns and scalable community chipping in large graphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 435–448. Springer, 2010. (document), 7.4, 7.11

[201] S. Prakash, A. Reisizadeh, R. Pedarsani, and S. Avestimehr. Coded computing for distributed graph analytics. *ISIT*, 2018. 2.3.4

[202] E. Rachlin and J. E. Savage. A framework for coded computation. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 2342–2346, 7 2008. 2.3.7

[203] C. Radhakrishnan and W. Jenkins. Fault tolerance in transform-domain adaptive filters operating with real-valued signals. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(1):166–178, 1 2010. 2.3.9

[204] C. Radhakrishnan and A. C. Singer. Recursive least squares filtering under stochastic computational errors. In *Proceedings of IEEE Asilomar Conference on Signals, Systems and Computers*, pages 1529–1532, 2013. 2.3.9

[205] M. Raginsky. Logarithmic Sobolev inequalities and strong data processing theorems for discrete channels. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 419–423, 7 2013. 2.3.11

[206] M. Raginsky and I. Sason. Concentration of measure inequalities in information theory, communications, and coding. *Foundations and Trends®in Communications and Information Theory*, 10:1–246, 2013. 4.1.2

[207] R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, pages 873–880. ACM, 2009. 9.1, 23

[208] N. Raviv, I. Tamo, R. Tandon, and A. G. Dimakis. Gradient coding from cyclic MDS codes and expander graphs. *CoRR*, abs/1707.03858, 2017. 1.3.2, 1.3.2, 2.3.3, 2.3.4

[209] G. Redinbo. Generalized algorithm-based fault tolerance: error correction via Kalman estimation. *IEEE Transactions on Computers*, 47(6):639–655, 1998. 2.3.2, 2.3.9

[210] A. Reisizadehmobarakeh, S. Prakash, R. Pedarsani, and S. Avestimehr. Coded computation over heterogeneous clusters. *arXiv preprint arXiv:1701.05973*, 2017. 2.3.3

[211] S. Rendle, D. Fetterly, E. J. Shekita, and B.-y. Su. Robust large-scale machine learning in the cloud. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1125–1134. ACM, 2016. 1.2.1

[212] T. Richardson and R. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47(2):599–618, 2 2001. 2.3.8

[213] T. Richardson and R. Urbanke. *Modern coding theory*. Cambridge University Press, 2008. 2

[214] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986. 0, 18

[215] H. Rutishauser. Computational aspects of FL Bauer's simultaneous iteration method. *Numerische Mathematik*, 13(1):4–13, 1969. 1.3.2, 2.3.5, 7.4.1

[216] H. Rutishauser. Simultaneous iteration method for symmetric matrices. *Numerische Mathematik*, 16(3):205–223, 1970. 7.4.1

[217] Y. Saad. *Iterative methods for sparse linear systems*, volume 82. siam, 2003. 2.3.5, 6.1, 6.2.1

[218] A. Sandryhaila and J. Moura. Discrete signal processing on graphs. *IEEE transactions on signal processing*, 61(7):1644–1656, 2013. 6.2.3

[219] S. Sarvotham, D. Baron, and R. G. Baraniuk. Sudocodes-fast measurement and reconstruction of sparse signals. In *International Symposium on Information Theory*, pages 2804–2808, 2006. 2.3.7

[220] L. L. Scharf. *Statistical signal processing*, volume 98. Addison-Wesley Reading, MA, 1991. 4.1.1

[221] M. Sefidgaran and A. Tchamkerten. On cooperation in multi-terminal computation and rate distortion. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 766–770, 7 2012. 2.3.12

[222] M. Sefidgaran and A. Tchamkerten. Distributed function computation over a tree network. In *Information Theory Workshop*, pages 1–5. IEEE, 2013. 2.3.12

[223] A. Severinson, A. G. i Amat, and E. Rosnes. Block-diagonal and LT codes for distributed computing with straggling servers. *IEEE Transactions on Communications*, 2018. 2.3.3

[224] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(Feb):567–599, 2013. 1.2.2

[225] O. Shamir, N. Srebro, and T. Zhang. Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, pages 1000–1008, 2014. 2.3.6

[226] N. R. Shanbhag, S. Mitra, G. de Veciana, M. Orshansky, R. Marculescu, J. Roychowdhury, D. Jones, and J. M. Rabaey. The search for alternative computational paradigms. *IEEE Design and Test of Computers*, 25(4):334–343, 2008. 1.2.1

[227] C. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(4):623–656, 10 1948. 0, 1.1, 1.2.3

[228] B. Shim, S. Sridhara, and N. Shanbhag. Reliable low-power digital signal processing via reduced precision redundancy. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(5):497–510, 5 2004. 2.3.9

[229] M. M. Shulaker, G. Hills, N. Patil, H. Wei, H. Chen, H.-S. P. Wong, and S. Mitra. Carbon nanotube computer. *Nature*, 501(7468):526–530, 2013. 1.2.1

[230] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013. 6.2.3

[231] F. Simon. Capacity of a noisy function. In *Proceedings of the IEEE Information Theory Workshop (ITW)*, pages 1–5, 8 2010. 2.3.7

[232] F. Simon. On the capacity of noisy computations. In *Proceedings of the IEEE Information Theory Workshop (ITW)*, pages 185–189, 10 2011. 2.3.7

[233] M. Sipser and D. A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6), 1996. 1.2.3, 2.2.2, 2.3.8, A.3, A.3.2

[234] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434, 2017. 1.2.1

[235] N. Sommer, M. Feder, and O. Shalvi. Low-density lattice codes. *IEEE Transactions on Information Theory*, 54(4):1561–1585, 4 2008. 10.5

[236] Y. Song and N. Devroye. Lattice codes for the Gaussian relay channel: Decode-and-forward and compress-and-forward. *IEEE Transactions on Information Theory*, 59(8):4927–4948, 2013. 2.3.12

[237] R. Soundararajan and S. Vishwanath. Communicating linear functions of correlated gaussian sources over a MAC. *IEEE Transactions on Information Theory*, 58(3):1853–1860, 2012. 2.3.12

[238] D. A. Spielman. Highly fault-tolerant parallel computation. In *Proceedings of the IEEE 37th Annual Symposium on Foundations of Computer Science*, pages 154–163, 10 1996. 2.3.2, 2.3.7

[239] G. Stewart. Simultaneous iteration for computing invariant subspaces of non-hermitian matrices. *Numerische Mathematik*, 25(2):123–136, 1976. 2.3.5, 7.4.1

[240] H.-I. Su and A. El Gamal. Distributed lossy averaging. *IEEE Transactions on Information Theory*, 56(7):3422–3437, 7 2010. 1.1, 2.3.12, 4.1, 4.3.3, 9

[241] G. Suh, K. Lee, and C. Suh. Matrix sparsification for coded matrix multiplication. In *Communication, Control, and Computing (Allerton)*, pages 1271–1278, 2017. 2.3.3

[242] P. L. Suresh, M. Canini, S. Schmid, and A. Feldmann. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *12th USENIX Symposium on Networked Systems Design and Implementation*, pages 513–527. USENIX Association, 2015. 1.1, 2.3.1

[243] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3368–3376, 2017. 1.2.1, 1.3.2, 1.3.2, 2.3.3, 2.3.4, 2.3.6

[244] M. G. Taylor. Reliable computation in computing systems designed from unreliable components. *Bell System Technical Journal*, 47(10):2339–2366, 1968. 1.1, 2.2.4, 2.3.8

[245] M. G. Taylor. Reliable information storage in memories designed from unreliable components. *Bell System Technical Journal*, 47(10):2299–2337, 1968. 1.2.1, 1.2.3, 2.3.8

[246] C. D. Thompson. *A complexity theory for VLSI*. PhD thesis, Carnegie-Mellon University Pittsburg, PA, 1980. 0

[247] A. Tripathy and A. Ramamoorthy. Sum-networks from undirected graphs: con-

struction and capacity analysis. In *52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 651–658. IEEE, 2014. 2.3.12

[248] R. A. Van De Geijn and J. Watts. Summa: Scalable universal matrix multiplication algorithm. *Concurrency: Practice and Experience*, 9(4):255–274, 1997. 2.3.6, 7.2.1, 1

[249] L. R. Varshney. Performance of LDPC codes under faulty iterative decoding. *IEEE Transactions on Information Theory*, 57(7):4427–4444, 2011. 2.3.8

[250] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, and S. Seth. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5, 2013. 5.1

[251] K. Viswanathan. On the memory required to compute functions of streaming data. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 196–200, 6 2010. 2.3.12

[252] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007. 7.4, 7.5.2

[253] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In *Automata Studies*, pages 329–378. Princeton University Press, 1956. 0, 1.1, 1.2.1, 1.2.3, 2.3.7, 3.2.1, 11

[254] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Low latency via redundancy. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 283–294, 2013. 2.3.1

[255] A. Wagner. On distributed compression of linear functions. *IEEE Transactions on Information Theory*, 57(1):79–94, 1 2011. 2.3.12

[256] A. Wagner, S. Tavildar, and P. Viswanath. Rate region of the quadratic Gaussian two-encoder source-coding problem. *IEEE Transactions on Information Theory*, 54(5):1938–1961, 5 2008. 2.3.12

[257] A. Wald and J. Wolfowitz. Optimum character of the sequential probability ratio test. *The Annals of Mathematical Statistics*, 19(3):pp. 326–339, 1948. 2

[258] C.-Y. Wang, S.-W. Jeon, and M. Gastpar. Multi-round computation of type-threshold functions in collocated gaussian networks. In *Proceedings of 2013 IEEE International Symposium on Information Theory (ISIT)*, pages 2154–2158, 7 2013. 2.3.12

[259] D. Wang, G. Joshi, and G. Wornell. Efficient task replication for fast response times in parallel computation. In *ACM SIGMETRICS Performance Evaluation Review*, volume 42, pages 599–600. ACM, 2014. 1.1, 2.3.1

[260] D. Wang, G. Joshi, and G. Wornell. Using straggler replication to reduce latency in large-scale parallel computing. *ACM SIGMETRICS Performance Evaluation Review*, 43(3):7–11, 2015. 1.1, 2.3.1

[261] S. Wang, J. Liu, and N. Shroff. Coded sparse matrix multiplication. *arXiv preprint arXiv:1802.03430*, 2018. 2.3.3, 2.3.4

[262] S. Wang, J. Liu, N. Shroff, and P. Yang. Fundamental limits of coded linear transform. *arXiv preprint arXiv:1804.09791*, 2018. 2.3.3

[263] S.-J. Wang and N. K. Jha. Algorithm-based fault tolerance for FFT networks. *IEEE Transactions on Computers*, 43(7):849–854, 7 1994. 2.3.2, 27

[264] X. Wang, P. Liu, and Y. Gu. Local-set-based graph signal reconstruction. *IEEE Transactions on Signal Processing*, 63(9):2432–2444, 2015. 2.3.5

[265] D. Williams. *Probability with martingales*. Cambridge university press, 1991. 4.1.1

[266] J. Wolf. Redundancy, the discrete Fourier transform, and impulse noise cancellation. *IEEE Transactions on Communications*, 31(3):458–461, 3 1983. 28

[267] D. P. Woodruff et al. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014. 1.1

[268] Y. Wu. On the HWI inequality. A work in progress. 4.1.2

[269] A. Xu and M. Raginsky. A new information-theoretic lower bound for distributed function computation. In *IEEE International Symposium on Information Theory (ISIT)*, pages 2227–2231. IEEE, 2014. 2.3.12

[270] W. Xu and B. Hassibi. Efficient compressive sensing with deterministic guarantees using expander graphs. In *Information Theory Workshop*, pages 414–419, 2007. 2.3.7

[271] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz. Wrangler: Predictable and faster jobs using fewer resources. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14. ACM, 2014. 0

[272] Y. Yang, M. Chaudhari, P. Grover, and S. Kar. Coded iterative computing using substitute decoding. *arXiv preprint arXiv:1805.06046*, 2018. 1.3.2, 2.3.3, 5.1, 7.1, 7.2.3, 7.4

[273] Y. Yang, P. Grover, and S. Kar. Can a noisy encoder be used to communicate reliably? In *Communication, Control, and Computing (Allerton)*, pages 659–666, 2014. 2.3.2, 2.3.3, 2.3.8, 2.3.11, 3.1

[274] Y. Yang, P. Grover, and S. Kar. Information dissipation in noiseless lossy in-network function computation. In *53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 445–452. IEEE, 2015. 1.3.3, 2.3.12, 4.1

[275] Y. Yang, P. Grover, and S. Kar. Fault-tolerant distributed logistic regression using unreliable components. In *54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 940–947, 2016. 2.3.2, 9.1

[276] Y. Yang, P. Grover, and S. Kar. Fault-tolerant parallel linear filtering using compressive sensing. In *IEEE International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, pages 201–205, 2016. 2.3.3, 9.1

[277] Y. Yang, P. Grover, and S. Kar. Coded distributed computing for inverse problems. In *Advances in Neural Information Processing Systems (NIPS)*, pages 709–719, 2017. 1.3.2, 2.3.3, 5.1, 6.1, 6.2.3, 6.3.3, 6.3.4, 6.3.5, 6.3.10, 6.4.1, D.1

[278] Y. Yang, P. Grover, and S. Kar. Computing linear transformations with unreliable

components. *IEEE Transactions on Information Theory*, 63(6), 2017. 1.3.3, 2.3.2, 2.3.3, 2.3.10, 2.3.11, 3.1, 3.4, 4, 3.4.1, 3.5.1, 5.2.1, 11

[279] Y. Yang, P. Grover, and S. Kar. Rate distortion for lossy in-network linear function computation and consensus: Distortion accumulation and sequential reverse water-filling. *IEEE Trans. Inf. Theory*, 63(8):5179–5206, 2017. 1.3.3, 2.3.11, 2.3.12, 4.1, 4.1.2, 4.4.3, 4.4.3, 4.4.3, B.3.3, B.3.3

[280] Y. Yang, P. Grover, and S. Kar. Coding for a single sparse inverse problem. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1575–1579. IEEE, 2018. 2.3.3, 5.1, 7.1

[281] Y. Yang, M. Interlandi, P. Grover, S. Kar, S. Amizadeh, and M. Weimer. Coded elastic computing. *arXiv preprint arXiv:1812.06411*, 2019. 1.3.1, 2.3.3, 5.1, 5.2, 5.2.1, 11, 5.4.1, 5.4.5

[282] Y. Yang, S. Kar, and P. Grover. Energy efficient distributed coding for data collection in a noisy sparse network. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 2734–2738, 7 2016. 8.1, 8.6

[283] Y. Yang, S. Kar, and P. Grover. Graph codes for distributed instant message collection in an arbitrary noisy broadcast network. *IEEE Transactions on Information Theory*, 63(9):6059–6084, 2017. 1.3.3, 8.1, 8.1.1, 8.1, 8.3, 8.3.1, 2

[284] A. C.-C. Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 209–213. ACM, 1979. 0, 1.1

[285] S. M. T. Yazdi, H. Cho, and L. Dolecek. Gallager B decoder on noisy hardware. *IEEE Transactions on Communications*, 61(5):1660–1673, 5 2013. 2.3.8

[286] M. Ye and E. Abbe. Communication-computation efficient gradient coding. *arXiv preprint arXiv:1802.03475*, 2018. 2.3.3, 2.3.4, 2.3.6, 7.5.4, 17

[287] L. Ying, R. Srikant, and G. Dullerud. Distributed symmetric function computation in noisy wireless sensor networks. *IEEE Transactions on Information Theory*, 53(12):4826–4833, 12 2007. 2.3.12

[288] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr. Coded Fourier Transform. *Communication, Control, and Computing (Allerton)*, 2017. 2.3.3

[289] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication. *NIPS*, 2017. 2.3.3

[290] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr. Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding. *arXiv preprint:1801.07487*, 2018. 2.3.3

[291] Q. Yu, N. Raviv, J. So, and A. S. Avestimehr. Lagrange coded computing: Optimal design for resiliency, security and privacy. *Workshop on Systems for ML and Open Source Software at NeurIPS*, 2018. 2.3.3

[292] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010. 2.3.1

[293] F. Zhang and H. D. Pfister. Verification decoding of high-rate LDPC codes with applications in compressed sensing. *IEEE Transactions on Information Theory*, 58(8):5042–5058, 2012. 2.3.7

[294] Y. Zhang and X. Lin. Disco: Distributed optimization for self-concordant empirical loss. In *International conference on machine learning*, pages 362–370, 2015. 2.3.6

[295] Y. Zhang, M. J. Wainwright, and J. C. Duchi. Communication-efficient algorithms for statistical optimization. In *Advances in Neural Information Processing Systems*, pages 1502–1510, 2012. 2.3.6

[296] Z. Zhang, L. Cherkasova, and B. T. Loo. Performance modeling of mapreduce jobs in heterogeneous cloud environments. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pages 839–846. IEEE, 2013. 2.3.3

[297] C. Zhao, X. Bai, and S. Dey. Evaluating transient error effects in digital nanometer circuits. *IEEE Transactions on Reliability*, 56(3):381–391, 2007. 1.2.1, 2.3.3

[298] R. Zheng and R. Barton. Toward optimal data aggregation in random wireless sensor networks. In *Proceedings of the 26th IEEE International Conference on Computer Communications*, pages 249–257, 5 2007. 2.3.12

[299] X. Zhu, Z. Ghahramani, and J. D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *International Conference on Machine Learning*, pages 912–919, 2003. 2.3.5

[300] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603, 2010. 2.3.6

[301] V. A. Zorich and R. Cooke. *Mathematical analysis II*. Springer Science & Business Media, 2004. G.2

Appendix A

Theoretical proofs for Chapter 3

A.1 Proof of Theorem 3.4.2

In probabilistic settings, the number of errors at any stage could exceed $N\alpha_0$. In what follows, we use large deviation analysis to show that the probability of exceeding $N\alpha_0$ is small. First, we review the large deviation result for binomial distribution [52, page 502, Example 3].

Lemma A.1.1. *Let $X_i, i = 1, \ldots, N$ be N i.i.d. binary random variables with $\Pr[X_i = 1] = p$. Then*

$$\Pr\left[\frac{1}{N}\sum_{i=1}^{N} X_i > (p+\lambda)\right] < \exp\left[-D(p+\lambda\|p)N\right], \tag{A.1}$$

where $D(p+\lambda\|p) = (p+\lambda)\log_e \frac{p+\lambda}{p} + (1-p-\lambda)\log_e \frac{1-p-\lambda}{1-p}$. Further, if $p < \lambda$,

$$\Pr\left[\frac{1}{N}\sum_{i=1}^{N} X_i > (p+\lambda)\right] < \exp\left[-D(2\lambda\|\lambda)N\right]. \tag{A.2}$$

Proof. The inequality (A.1) is the large deviation bound for binomial distribution and is presented in [52, page 502, Example 3]. Note that $D(p+\lambda\|p)$ is monotone non-increasing for $p \in (0, \lambda)$. When $p < \lambda$, we have $D(p+\lambda\|p) > D(2\lambda\|\lambda)$. Therefore, (A.1) holds for $p < \lambda$. □

Then, Theorem 3.4.2 follows from Theorem 3.4.1 and Lemma A.1.1.

Using Theorem 3.4.1, we know that if the error fraction in all stages is bounded by the inequality (3.11), the final error fraction is at most $N\alpha_0$.

From Lemma A.1.1, we know that

$$\Pr(\alpha_{\text{and}} > p_{\text{and}} + \lambda) < \exp\left[-D(p_{\text{and}} + \lambda\|p_{\text{and}})N\right] < \exp[-D(2\lambda\|\lambda)N], \tag{A.3}$$

$$\Pr(\alpha_{\text{xor}} > p_{\text{xor}} + \lambda) < \exp\left[-D(p_{\text{xor}} + \lambda\|p_{\text{xor}})N\right] < \exp[-D(2\lambda\|\lambda)N], \tag{A.4}$$

$$\Pr(\alpha_{\text{maj}} > p_{\text{maj}} + \lambda) < \exp\left[-D(p_{\text{maj}} + \lambda\|p_{\text{maj}})N\right] < \exp[-D(2\lambda\|\lambda)N]. \tag{A.5}$$

Setting $\lambda = \frac{\theta\alpha_0/2}{(d_s-1)+[d_c(1-R)+1]+1}$ as in the condition (3.15), we have

$$(d_s-1)p_{\text{and}} + [d_c(1-R)+1]\,p_{\text{xor}} + p_{\text{maj}} < (d_s-1)\lambda + [d_c(1-R)+1]\,\lambda + \lambda = \frac{\theta\alpha_0}{2}.$$

Therefore,

$$\begin{aligned}
&\Pr\Big((d_s-1)\alpha_{\text{and}} + [d_c(1-R)+1]\,\alpha_{\text{xor}} + \alpha_{\text{maj}} > \theta\alpha_0\Big)\\
<&\Pr\Big((d_s-1)\alpha_{\text{and}} + [d_c(1-R)+1]\,\alpha_{\text{xor}} + \alpha_{\text{maj}}\\
&\qquad > (d_s-1)p_{\text{and}} + [d_c(1-R)+1]\,p_{\text{xor}} + p_{\text{maj}} + \frac{\theta\alpha_0}{2}\Big)\\
<&\Pr((d_s-1)\alpha_{\text{and}} > (d_s-1)p_{\text{and}} + (d_s-1)\lambda)\\
&+\Pr([d_c(1-R)+1]\,\alpha_{\text{xor}} > [d_c(1-R)+1]\,p_{\text{xor}}\\
&\qquad + [d_c(1-R)+1]\,\lambda) + \Pr(\alpha_{\text{maj}} > p_{\text{maj}} + \lambda)\\
=&\Pr(\alpha_{\text{and}} > p_{\text{and}} + \lambda) + \Pr(\alpha_{\text{xor}} > p_{\text{xor}} + \lambda)\\
&+\Pr\big(\alpha_{\text{maj}} > p_{\text{maj}} + \lambda\big) < 3\exp(-ND(2\lambda\|\lambda)),
\end{aligned}$$

where

$$\begin{aligned}
D(2\lambda\|\lambda) &= 2\lambda\log 2 + (1-2\lambda)\log\frac{1-2\lambda}{1-\lambda}\\
&= 2\lambda\log 2 - (1-2\lambda)\log\left(1+\frac{\lambda}{1-2\lambda}\right)\\
&= 2\lambda\log 2 - (1-2\lambda)\left(\frac{\lambda}{1-2\lambda}+\mathcal{O}(\lambda^2)\right)\\
&= (2\log 2 - 1)\,\lambda + \mathcal{O}(\lambda^2).
\end{aligned} \tag{A.6}$$

Since $(d_s-1)\alpha_{\text{and}} + \alpha_{\text{maj}} < (d_s-1)\alpha_{\text{and}} + [d_c(1-R)+1]\,\alpha_{\text{xor}} + \alpha_{\text{maj}}$, we also have

$$\Pr\big((d_s-1)\alpha_{\text{and}} + \alpha_{\text{maj}} > \theta\alpha_0\big) < 3\exp\left(-D(2\lambda\|\lambda)N\right).$$

Therefore, using the union bound for the L stages, the total error probability is upper bounded by $P_e^{\text{blk}} < 3L\exp\left(-D(2\lambda\|\lambda)N\right)$.

A.2 Proof of Theorem 3.4.3

Theorem 3.4.3 provides a lower bound on the number of operations by lower-bounding the operations done at the entrance stage of the noisy circuit, i.e., operations that have one of the L inputs $(s_1, s_2, ...s_L)$ as an argument. In order to prove Theorem 3.4.3, we need the following lemma (stated implicitly in [196, Proposition 1]) which characterizes the equivalence of a noisy-gate model and a noisy-wire model.

Lemma A.2.1. *For each unreliable gate from Gate Model I (D,ϵ) with error probability ϵ and fan-in number $\leq D$, its output variable can be stochastically simulated by (equivalent in distribution* *to) another unreliable gate $\tilde{g}$ that computes the same function but with the following property: each input wire flips the input independently with probability ϵ/D and the gate has additional output noise independent of input wire noise.*

Proof. For an arbitrary unreliable gate

$$y = g(u_1, u_2, ..., u_d) \oplus z_g, d \leq D,$$

consider another unreliable gate together with noisy wires

$$\tilde{y} = \tilde{g}(u_1, u_2, ..., u_d) \oplus \tilde{z}_g = g(u_1 \oplus w_1, u_2 \oplus w_2, ..., u_d \oplus w_d) \oplus \tilde{z}_g, d \leq D,$$

where w_j is the noise on the j-th input wire and takes value 1 with probability ϵ/D. The probability that all d wires convey the correct inputs is $(1 - \epsilon/D)^d > 1 - d\frac{\epsilon}{D} > 1 - \epsilon$. Therefore, if $\tilde{z}_g$ is 0 w.p.1, the error of $\tilde{g}$ will be smaller than ϵ. Thus, using standard continuity arguments, we can find a random variable $\tilde{z}_g$ which equals to 1 w.p. $\epsilon' < \epsilon$, while making $\tilde{y}$ and y equivalent in distribution. □

Based on this lemma, we know that a noisy network defined in Section 3.2.1 can always be replaced by another network, where each wire has an error probability $\frac{\epsilon}{D}$. Before a specific input s_k enters the noisy circuit, it is always transmitted along the wires connected to the entrance stage of the gates in the circuit. Because of the assumption that gates after the inputs are noisy, the bit will be 'sampled' by the noisy wires. For convenience of analysis, we assume each gate can only be used once so that the number of operations is equal to the number of unreliable gates. Now that each gate only computes once, each noisy wire can only carry information once as well. We assume each s_k is transmitted on T_k distinct wires. Then, the probability that the message on all T_k wires flips is

$$p_k = (\epsilon/D)^{T_k}. \tag{A.7}$$

Therefore, the error probability of the input bit s_k satisfies $P_{\text{in}}^k > p_k$. Since matrix $\mathbf{A}$ is assumed to have full row rank, if the linear transformation computation is noiseless, even a single input bit error leads to an output block error. Therefore, even when the linear transformation computation is noiseless, the output block error probability P_e^{blk} is greater than the input error probability P_{in}^k. Since the computation is noisy, P_e^{blk} is still greater than P_{in}^k, and hence is greater than p_k. Therefore, if $(\epsilon/D)^{T_k} = p_k > p_{\text{tar}}$, the block error probability $P_e^{\text{blk}} > P_{\text{in}}^k > p_k > p_{\text{tar}}$, which contradicts with the aim to make the block error probability smaller than p_{tar}. Thus,

$$p_{\text{tar}} > (\epsilon/D)^{T_k},$$

which means that for any bit s_k

$$T_k > \frac{\log 1/p_{\text{tar}}}{\log D/\epsilon}. \tag{A.8}$$

Therefore, the number of wires connected to each input bit must be at least $\frac{\log 1/p_{\text{tar}}}{\log D/\epsilon}$. Since the number of input bits is L, the total number of wires connected to all input bits is at least $\frac{L \log 1/p_{\text{tar}}}{\log D/\epsilon}$. Since we are using gates with bounded fan-in smaller than D, the number of gates is at least $\frac{L \log 1/p_{\text{tar}}}{D \log D/\epsilon}$, so does the number of operations. Since there are K output bits, the number of operations per output bit $\mathscr{N}_{\text{per-bit}} > \frac{L \log 1/p_{\text{tar}}}{KD \log D/\epsilon}$.

A.3 Codes that satisfy the noisy decoding requirement

The existence of codes that satisfy the requirement (A.3) in Section 2.2.2 follows from a result in [39]. We first present the result from [39].

Define $\beta_0, \beta_1, \beta_2, \beta_3$ respectively as the largest integer less than $d_v/2$, the largest integer less than or equal to $d_v/2$, the smallest integer greater than or equal to $d_v/2$, and the smallest integer greater than $d_v/2$. Create four real parameters γ_{12}, δ_{12}, π_0 and ω_0 that satisfy the following inequalities

$$(1-\theta)\alpha N \leq \gamma_{12} N + \delta_{12} N, \tag{A.9}$$

$$0 \leq \gamma_{12} N \leq \alpha N, \tag{A.10}$$

$$0 \leq \pi_0 (1-R) N \leq \omega_0 d_v N \leq \alpha d_v N, \tag{A.11}$$

$$\beta_3(\alpha - \gamma_{12}) N \leq \omega_0 d_v N \leq \min\left(\frac{d'}{d_c}\pi_0 d_v N, \gamma_{12}\beta_1 N + d_v(\alpha - \gamma_{12})N\right), \tag{A.12}$$

where d' is the largest odd number which is less than or equal to d_c, and

$$0 \leq \delta_{12} N \beta_2 \leq (\pi_0 - \omega_0) d_v N. \tag{A.13}$$

Define the following polynomials

$$F_0(x) \triangleq \sum_{j=0}^{\beta_0} \binom{d_v}{j} x^j, \tag{A.14}$$

$$F_1(x) \triangleq \sum_{j=0}^{\beta_1} \binom{d_v}{j} x^j, \tag{A.15}$$

$$F_2(x) \triangleq \sum_{j=\beta_2}^{d_v} \binom{d_v}{j} x^j, \tag{A.16}$$

$$F_3(x) \triangleq \sum_{j=\beta_3}^{d_v} \binom{d_v}{j} x^j, \tag{A.17}$$

$$G_o(x) \triangleq \sum_{j=1,3,\ldots,d'} \binom{d_c}{j} x^j, \tag{A.18}$$

$$G_e(x) \triangleq \sum_{j=0,2,\ldots,d''}^{c} \binom{d_c}{j} x^j, \tag{A.19}$$

where d'' is the largest even number less than or equal to d_c. Then we define

$$\begin{aligned}\psi(\alpha, \gamma_{12}, \delta_{12}, \pi_0, \omega_0) \triangleq\ & h(\gamma_{12}, \alpha - \gamma_{12}, \delta_{12}) + (1 - R)h(\pi_0) \\ & + t_1 + t_2 + u_1 + u_2 - d_v h(\omega_0, \alpha - \omega_0, \pi_0 - \omega_0),\end{aligned} \tag{A.20}$$

where $h(\cdot)$ is the entropy function defined as

$$h(\tau_1, \tau_2, \dots, \tau_i) = -\sum_{j=1}^{i} \tau_j \log \tau_j - \left(1 - \sum_{j=1}^{i} \tau_j\right) \log \left(1 - \sum_{j=1}^{i} \tau_j\right), \tag{A.21}$$

and

$$t_1 = \inf_{x>0} \left\{\gamma_{12} \log F_1 + (\alpha - \gamma_{12}) \log F_3 - \omega_0 d_v \log x\right\}, \tag{A.22}$$

$$t_2 = \inf_{x>0} \left\{\delta_{12} \log F_2 + (1 - \alpha - \delta_{12}) \log F_0 \ - (\pi_0 - \omega_0) d_v \log x\right\}, \tag{A.23}$$

$$u_1 = \inf_{x>0} \left\{\pi_0 (1 - R) \log G_o - \omega_0 d_v \log x\right\}, \tag{A.24}$$

$$u_2 = \inf_{x>0} \left\{(1 - \pi_0)(1 - R) \log G_e - (\alpha - \omega_0) d_v \log x\right\}. \tag{A.25}$$

The base of all the logarithms is e. Then, Theorem 1 of [39] and the last paragraph on page 521 of [39] implies the following result:
Lemma A.3.1. *([39, Theorem 1]) Consider the random ensemble of (d_v, d_c)-regular LDPC codes with $d_v > 4$ and block length N. Let α_0 be the smallest positive root of the function $f(\alpha)$ which is defined by*

$$f(\alpha) = \max_{\gamma_{12}, \delta_{12}, \pi_0, \omega_0} \psi(\alpha, \gamma_{12}, \delta_{12}, \pi_0, \omega_0), \tag{A.26}$$

where the maximization is over all values of $\gamma_{12}, \delta_{12}, \pi_0, \omega_0$ that satisfy (A.9)-(A.13). *Then, for any $\bar{\alpha}_0 < \alpha_0$, if N is sufficiently large, then except for almost all codes in this ensemble can correct at least $\theta\bar{\alpha}_0 N$ errors out of any arbitrary $\bar{\alpha}_0 N$ errors using one iteration of the PBF algorithm.*

Proof. Here we briefly summarize the proof in [39]. Denote by $\bar{p}_e(\bar{\alpha}_0 N)$ the fraction of (bad) codes in the (d_v, d_c)-regular ensemble that cannot correct a linear fraction $\theta\bar{\alpha}_0 N$ of all combinations of $\bar{\alpha}_0 N$ errors or less using one iteration of the PBF algorithm. Then, according to (38) in [39], $\bar{p}_e(\alpha_0 N)$ is upper-bounded by

$$\bar{p}_e(\bar{\alpha}_0 N) \le \sum_{\alpha N \le \bar{\alpha}_0 N} C(\alpha N)^{11/2} e^{N f(\alpha)}, \tag{A.27}$$

where the summation is over all integer values of $\alpha N \le \bar{\alpha}_0 N$, and $C = (2\pi)^{3/2} e^{1/3} \frac{d_v^{9/2} d_c^{3/2}}{\beta_2}$. Therefore, when $\bar{\alpha}_0$ is sufficiently small so that $f(\alpha) < 0$ for all $\alpha < \bar{\alpha}_0$, $\bar{p}_e(\bar{\alpha}_0 N) \to 0$ as $N \to \infty$, which means that almost all codes in the (d_v, d_c)-regular ensemble can correct θ fraction of all possible combinations of $\bar{\alpha}_0 N$ errors using one iteration of the PBF algorithm. □

Theorem 1 in [39] was stated for $\theta = 0$ and the original constraint corresponding to the constraint (A.9) ($(1-\theta)\alpha N \leq \gamma_{12}N + \delta_{12}N$) was $\alpha N \leq \gamma_{12}N + \delta_{12}N$. In this chapter,we use the result for $\theta = \text{constant} > 0$. This result can be obtained by directly changing the original constraint $\alpha N \leq \gamma_{12}N + \delta_{12}N$ in [39] to the new constraint $(1-\theta)\alpha N \leq \gamma_{12}N + \delta_{12}N$ (this direct change is also stated at the bottom of page 521 in [39] after the proof of Theorem 1). A refined bound for (A.27) can be obtained using (22)(23)(25) and (33) in [39], which shows

$$\begin{aligned} \bar{p}_e(\bar{\alpha}_0 N) \leq \sum_{\alpha N \leq \bar{\alpha}_0 N} \Bigg(& \sum_{\gamma_{12}N, \delta_{12}N, \pi_0(1-R)N, \omega_0 d_v N} \\ & (2\pi N d_v)^{3/2} e^{1/3} \sqrt{\omega_0(\alpha - \omega_0)(\pi_0 - \omega_0)} e^{N\psi(\alpha, \gamma_{12}, \delta_{12}, \pi_0, \omega_0)} \Bigg), \end{aligned} \tag{A.28}$$

where the outer summation is over all integer values of $\alpha N \leq \bar{\alpha}_0 N$, and the inner summation is over all integer values of $\gamma_{12}N, \delta_{12}N, \pi_0(1-R)N, \omega_0 d_v N$ that satisfy (A.9) to (A.13). We will use this refined bound to obtain finite-length result in the following example.

Example 1. One example of the parameter choice is $d_v = 9$, $d_c = 18$ and $\theta = 0.15$. In this case, we computed the first positive root of $f(\alpha) = 0$ using MATLAB and obtained $\alpha = 5.1 \cdot 10^{-4}$. This means that using one iteration of the PBF algorithm, we can correct a fraction $\theta = 0.15$ of $5.1 \cdot 10^{-4} \cdot N$ worst-case errors using a $(9, 18)$ regular LDPC code when N is sufficiently large. We can also use this result to obtain finite-length bounds (computing an upper bound on the fraction of bad codes using (A.28)). We obtained that at least 4.86% of $(9, 18)$ regular LDPC codes of length $N = 50,000$ in the random LDPC ensemble can reduce the number of errors by 15% using one iteration of the PBF algorithm, when the number of errors is smaller than or equal to 20, which corresponds to the case when $\alpha_0 = 0.0004$.

The existence of codes that satisfy requirement (A.3) can also be established using Expander LDPC codes. Here, we review some results on expander LDPCs [233].

Definition: (Expander Graph) An $(N, P, d_v, \gamma, \alpha)$ bipartite expander is a d_v-left-regular bipartite graph $\mathcal{G}(\mathcal{V}_L \cup \mathcal{V}_R, \mathcal{E})$ where $|\mathcal{V}_L| = N$ and $|\mathcal{V}_R| = P$. In this bipartite graph, it holds that $\forall \mathcal{S} \subset \mathcal{V}_L$ with $|\mathcal{S}| \leq \gamma N$, $\mathcal{N}(\mathcal{S}) \geq \alpha d_v |\mathcal{S}|$, where $\mathcal{N}(\mathcal{S})$ denotes the neighborhood of the set $\mathcal{S}$, i.e., the set of nodes in $\mathcal{V}_R$ connected to $\mathcal{S}$.

An $(N, P, d_v, \gamma, \alpha)$ expander LDPC code is a length-N LDPC code, where the Tanner graph of the code is the corresponding expander graph with $\mathcal{V}_L$ corresponding to the set of variable nodes and $\mathcal{V}_R$ the parity check nodes. We use $d_c = d_v N/P$ to denote the right-degree of the expander code.

Lemma A.3.2. *([233, Thm11]) Using an $(N, P, d_v, \gamma, \frac{3}{4} + \epsilon_e)$ regular expander LDPC code with parity check node degree $d_c = d_v N/P$, one can use one iteration of noiseless PBF algorithm to bring the fraction of errors down from α to $(1 - 4\epsilon_e)\alpha$ provided that the original corrupted codeword has at most $\gamma(1 + 4\epsilon_e)/2$ fraction of errors.*

Example 2. The construction of a good Expander code has been investigated for a long time. Constructive approaches for Expander codes can be found in [45, 102]. In [40, 82, 213], it is shown that random regular LDPC codes are expanders with high probability when the code length $N \to \infty$. In [213, Theorem 8.7] it is shown that, suppose γ_{max} is the positive solution of the equation

$$\frac{d_v - 1}{d_v} h_2(\gamma) - \frac{1}{d_c} h_2(\gamma d_c * (3/4 + \epsilon_e)) - \gamma(3/4 + \epsilon_e) d_c h_2 \left(\frac{1}{(3/4 + \epsilon_e) d_c} \right) = 0, \qquad \text{(A.29)}$$

then, for $3/4 + \epsilon_e < \frac{d_v - 1}{d_v}$ and $\gamma \in (0, \gamma_{\text{max}})$, a random regular (d_v, d_c) LDPC Tanner graph is a $(d_v, d_c, \gamma, \frac{3}{4} + \epsilon_e)$ expander with probability $1 - O(N^{-\beta})$, where $\beta = d_v [1 - (3/4 + \epsilon_e)] - 1$ is a constant greater than 0 when $3/4 + \epsilon_e < \frac{d_v - 1}{d_v}$ (which means that all sets of left nodes with cardinality smaller than γN have an expansion factor at least $\frac{3}{4} + \epsilon_e$). For $d_v = 16$, $d_c = 32$, and $\epsilon_e = 0.0375$ (which is equivalent to $4\epsilon_e = 0.15$, the same as $\theta = 0.15$ in Example 1), we use MATLAB to numerically solve the above equation and obtained $\gamma_{\text{max}} \approx 4.1 * 10^{-5}$, which means the fraction of errors α can be as large as $\gamma_{\text{max}}(1 + 4\epsilon_e)/2 = 2.3575 \cdot 10^{-5}$.

A.4 Proof of Theorem 3.5.1

We tune the energy supply such that

$$\max(p_{\text{and}}, p_{\text{xor}}, p_{\text{maj}}) \le \lambda = \frac{\theta \alpha_0 / 2}{[d_c(1 - R) + 1] + 2}, \qquad \text{(A.30)}$$

is satisfied for the first $L - L_{vs}$ stages (first phase), which ensures that

$$p_{\text{and}} + [d_c(1 - R) + 1]\, p_{\text{xor}} + p_{\text{maj}} \le \theta \alpha_0 / 2, \qquad \text{(A.31)}$$

is satisfied. We tune the energy supply such that

$$\max(p_{\text{and}}^{(i+1)}, p_{\text{xor}}^{(i+1)}, p_{\text{maj}}^{(i+1)}) \le \lambda^{(i+1)} = \frac{\theta \alpha_0 \left(1 - \frac{1}{2}\theta\right)^i / 4}{[d_c(1 - R) + 1] + 2}, \qquad \text{(A.32)}$$

is satisfied for the last L_{vs} stages (second phase), which ensures that

$$[d_c(1 - R) + 1] p_{\text{xor}}^{(i+1)} + p_{\text{maj}}^{(i+1)} + p_{\text{and}}^{(i+1)} \le \frac{1}{4} \theta \alpha_0 \left(1 - \frac{1}{2}\theta\right)^i, \qquad \text{(A.33)}$$

is satisfied (we have mentioned this in (3.33)). Since this version of ENCODED-V technique with dynamic voltage scaling has the same procedure and constant supply energy during the first $L - L_{\text{vs}}$ stages (first phase) as the ENCODED-F technique, from Theorem 3.4.2, we know that after the first $(L - L_{\text{vs}})$ stages, the output error fraction is smaller than α_0 with probability at least $1 - P_e^{\text{blk}}$, where $P_e^{\text{blk}} < 3(L - L_{\text{vs}}) \exp(-\lambda^* N)$ and λ^* is defined in (3.18).

We will prove that, after the i-th stage of the remaining L_{vs} stages, the error fraction is upper bounded by

$$\alpha_{\text{PBF}}^{(i)} \le \alpha_0 (1 - \theta/2)^i, \qquad \text{(A.34)}$$

with high probability. Thus, after L_{vs} iterations, we obtain

$$\alpha_{\text{PBF}}^{(L_{\text{vs}})} \leq \alpha_0(1-\theta/2)^{L_{\text{vs}}} \leq p_{\text{tar}}, \tag{A.35}$$

where the last step can be verified by plugging in (3.32).

The case for $i = 0$ is already true as argued above. Suppose (A.34) holds for some $i \geq 0$, then, we prove (A.34) also holds for the $(i+1)$-th stage of the second phase. Note that from (A.33), the probability that the number of new errors introduced during the PBF decoding at the $(i+1)$-th stage, which is $[d_c(1-R)+1]\alpha_{\text{xor}}^{(i+1)} + \alpha_{\text{maj}}^{(i+1)} + \alpha_{\text{and}}^{(i+1)}$, satisfies

$$\begin{aligned}
&\Pr\left([d_c(1-R)+1]\alpha_{\text{xor}}^{(i+1)} + \alpha_{\text{maj}}^{(i+1)} + \alpha_{\text{and}}^{(i+1)} > \frac{1}{2}\alpha_0\theta(1-\theta/2)^i\right) \\
\overset{(a)}{<}\ &\Pr\left([d_c(1-R)+1]\alpha_{\text{xor}}^{(i+1)} + \alpha_{\text{maj}}^{(i+1)} + \alpha_{\text{and}}^{(i+1)}\right. \\
&\qquad \left. > [d_c(1-R)+1]p_{\text{xor}}^{(i+1)} + p_{\text{maj}}^{(i+1)} + p_{\text{and}}^{(i+1)} + \frac{1}{4}\alpha_0\theta(1-\theta/2)^i\right) \\
<\ &\Pr\left(\alpha_{\text{and}}^{(i+1)} > p_{\text{and}}^{(i+1)} + \lambda^{(i+1)}\right) + \Pr\left(\alpha_{\text{xor}}^{(i+1)} > p_{\text{xor}}^{(i+1)} + \lambda^{(i+1)}\right) \\
&\quad + \Pr\left(\alpha_{\text{maj}}^{(i+1)} > p_{\text{maj}}^{(i+1)} + \lambda^{(i+1)}\right) \overset{(b)}{<} 3\exp\left(-\widetilde{\lambda}^{(i+1)}N\right),
\end{aligned} \tag{A.36}$$

where step (a) follows from (A.33), step (c) follows from the large deviation bound in Lemma A.1.1 and $\widetilde{\lambda}^{(i+1)}$ is defined in (3.36). Therefore, with probability at least $1 - 3\exp\left(-\widetilde{\lambda}^{(i+1)}N\right)$,

$$\begin{aligned}
\alpha_{\text{PBF}}^{(i+1)} &\leq \alpha_{\text{PBF}}^{(i)}(1-\theta) + [d_c(1-R)+1]\alpha_{\text{xor}}^{(i+1)} + \alpha_{\text{maj}}^{(i+1)} + \alpha_{\text{and}}^{(i+1)} \\
&\overset{(a)}{\leq} \alpha_0(1-\theta/2)^i(1-\theta) + \frac{1}{2}\alpha_0\theta(1-\theta/2)^i \\
&= \alpha_0(1-\theta/2)^{i+1},
\end{aligned} \tag{A.37}$$

where step (a) can be obtained by combining (A.37) and (A.34). Now that we have proved (A.34) for the $(i+1)$-th stage, we can carry out the math induction for all i that satisfies $1 \leq i \leq L_{\text{vs}}$. If (A.34) holds for all i, the final error fraction is smaller than p_{tar}. Thus, the overall probability that the final error fraction is greater than p_{tar} is upper bounded by the summation of $3(L - L_{\text{vs}})\exp(-\lambda^* N)$ in the first $L - L_{\text{vs}}$ stages and the RHS of (A.36) for the last L_{vs} stages, which is

$$P_e^{\text{blk}} < 3(L - L_{\text{vs}})\exp(-\lambda^* N) + 3\sum_{i=1}^{L_{\text{vs}}} \exp\left(-\widetilde{\lambda}^{(i+1)}N\right). \tag{A.38}$$

Thus, (3.35) is proved.

Finally, we compute the overall energy consumption. The energy consumed in the i-th stage can be written as

$$E_i = N\epsilon_{\text{and}}^{-1}\left(p_{\text{and}}^{(i)}\right) + N\epsilon_{\text{maj}}^{-1}\left(p_{\text{maj}}^{(i)}\right) + (N+P)\epsilon_{\text{xor}}^{-1}\left(p_{\text{xor}}^{(i)}\right). \tag{A.39}$$

By summing over all stages both in the first phase and the second phase and normalizing by the number of outputs K, the total energy consumption per output bit can be written as in (3.34).

A.5 Proof of Corollary 3.5.2

We choose

$$p_{\text{and}} = p_{\text{xor}} = p_{\text{maj}} = \lambda = \frac{\theta\alpha_0/2}{[d_c(1-R)+1]+2}, \tag{A.40}$$

in the first $L - L_{\text{vs}}$ stages and

$$p_{\text{and}}^{(i+1)} = p_{\text{xor}}^{(i+1)} = p_{\text{maj}}^{(i+1)} = \lambda^{(i+1)} = \frac{\theta\alpha_0\left(1-\frac{1}{2}\theta\right)^i/4}{[d_c(1-R)+1]+2}, \tag{A.41}$$

in the i-th stage of the last L_{vs} stages (defined in (3.32)).

By plugging in (A.40), (A.41) and $L_{\text{vs}} = \left\lceil \frac{\log\frac{2}{p_{\text{tar}}}+\log\alpha_0}{\log\frac{1}{1-\frac{1}{2}\theta}} \right\rceil$ into the error probability expression (3.35), we know that the ENCODED-V technique has output error fraction smaller than $\alpha_0(1-\theta/2)^{L_{\text{vs}}} \le \frac{1}{2}p_{\text{tar}}$ with probability at least $1 - P_e^{\text{blk}}$, where P_e^{blk} satisfies

$$P_e^{\text{blk}} < 3(L-L_{\text{vs}})\exp\left(-\lambda^* N\right) + 3L_{\text{vs}}\exp(-\widetilde{\lambda}^{(L_{\text{vs}}+1)}N), \tag{A.42}$$

where $\widetilde{\lambda}^{(i+1)} = D(2\lambda^{(i+1)}\|\lambda^{(i+1)}) = (2\log 2 - 1)\,\lambda^{(i+1)} + \mathcal{O}((\lambda^{(i+1)})^2)$ and $\lambda^* = D(2\lambda\|\lambda) = (2\log 2-1)\,\lambda + \mathcal{O}(\lambda^2)$. Since $\lambda^{(L_{\text{vs}}+1)} = \frac{\theta\alpha_0\left(1-\frac{1}{2}\theta\right)^{L_{\text{vs}}}/4}{[d_c(1-R)+1]+2}$ and $\alpha_0(1-\theta/2)^{L_{\text{vs}}-1} > \frac{1}{2}p_{\text{tar}}$, we have $\lambda^{(L_{\text{vs}}+1)} > \frac{\theta p_{\text{tar}}\left(1-\frac{1}{2}\theta\right)/8}{[d_c(1-R)+1]+2}$. Therefore,

$$P_e^{\text{blk}} < 3L\exp(-\theta^* N), \tag{A.43}$$

where $\theta^* = \min\left\{\lambda^*, D\left(2\frac{\theta p_{\text{tar}}\left(1-\frac{1}{2}\theta\right)/4}{[d_c(1-R)+1]+2}\middle\|\frac{\theta p_{\text{tar}}\left(1-\frac{1}{2}\theta\right)/4}{[d_c(1-R)+1]+2}\right)\right\}$.

Denote the output error fraction by δ_e^{frac}, which is a random variable supported on $[0,1]$. We know that $\Pr(\delta_e^{\text{frac}} > \frac{1}{2}p_{\text{tar}}) < P_e^{\text{blk}}$. Thus, the output bit error probability is upper bounded by

$$\begin{aligned}\mathbb{E}[\delta_e^{\text{frac}}] < {} & \Pr(\delta_e^{\text{frac}} > \frac{1}{2}p_{\text{tar}})\mathbb{E}\left[\delta_e^{\text{frac}}|\delta_e^{\text{frac}} > \frac{1}{2}p_{\text{tar}}\right] \\ & + \Pr(\delta_e^{\text{frac}} \le \frac{1}{2}p_{\text{tar}}))\mathbb{E}\left[\delta_e^{\text{frac}}|\delta_e^{\text{frac}} \le \frac{1}{2}p_{\text{tar}}\right] < P_e^{\text{blk}} + \frac{1}{2}p_{\text{tar}}.\end{aligned} \tag{A.44}$$

When $N > \frac{1}{\theta^*}\log\left(\frac{6L}{p_{\text{tar}}}\right)$, $P_e^{\text{blk}} < \frac{1}{2}p_{\text{tar}}$, and hence the output bit error probability $\mathbb{E}[\delta_e^{\text{frac}}]$ satisfies $\mathbb{E}[\delta_e^{\text{frac}}] < p_{\text{tar}}$.

In this corollary, we only examine the case when $\epsilon_{\text{and}}(u) = \epsilon_{\text{xor}}(u) = \epsilon_{\text{maj}}(u) = \epsilon(u)$ (either polynomial decay or exponential decay). We also choose the same gate error probabilities $p_{\text{and}} = p_{\text{xor}} = p_{\text{maj}} = \lambda$ and $p_{\text{and}}^{(i+1)} = p_{\text{xor}}^{(i+1)} = p_{\text{maj}}^{(i+1)} = \lambda^{(i+1)}$ for different types of unreliable gates (see (A.40) and (A.41)). Therefore, the energy consumption (3.34) is simplified to

$$E_{\text{per-bit}} \leq \frac{3N+P}{K}\left(L - L_{\text{vs}}\right)\epsilon^{-1}\left(\lambda\right) + \frac{3N+P}{K}\sum_{i=1}^{L_{\text{vs}}}\epsilon^{-1}\left(\lambda^{(i)}\right), \tag{A.45}$$

where $L_{\text{vs}} = \left\lceil \frac{\log \frac{2}{p_{\text{tar}}} + \log \alpha_0}{\log \frac{1}{1-\frac{1}{2}\theta}} \right\rceil$.

When the energy-reliability tradeoff function $\epsilon_{\text{and}}(u) = \epsilon_{\text{xor}}(u) = \epsilon_{\text{maj}}(u) = (\frac{1}{u})^c, c > 0$, the total energy consumption per bit

$$\begin{aligned} E_{\text{per-bit}} &\leq \frac{3N+P}{K}\lambda^{-\frac{1}{c}}\left[(L - L_{\text{vs}}) + \frac{\left(1-\frac{1}{2}\theta\right)^{-\frac{L_{\text{vs}}}{c}} - 1}{\left(1-\frac{1}{2}\theta\right)^{-\frac{1}{c}} - 1}\right] \\ &\leq \frac{3N+P}{K}\lambda^{-\frac{1}{c}}\left[(L - L_{\text{vs}}) + \frac{\left(1-\frac{1}{2}\theta\right)^{-\frac{1}{c}}\left(\frac{p_{\text{tar}}}{2\alpha_0}\right)^{-\frac{1}{c}} - 1}{\left(1-\frac{1}{2}\theta\right)^{-\frac{1}{c}} - 1}\right] \\ &= \Theta\left(\frac{N}{K}\max\left\{L, \left(\frac{1}{p_{\text{tar}}}\right)^{\frac{1}{c}}\right\}\right). \end{aligned} \tag{A.46}$$

When the energy-reliability tradeoff function $\epsilon_{\text{and}}(u) = \epsilon_{\text{xor}}(u) = \epsilon_{\text{maj}}(u) = \exp(-cu), c > 0$, the total energy consumption per bit

$$E_{\text{per-bit}} = \frac{3N+P}{cK}\left(L\log\frac{1}{\lambda} + \frac{1}{2}L_{\text{vs}}\left(L_{\text{vs}}+1\right)\log\frac{1}{1-\frac{1}{2}\theta}\right) = \Theta\left(\frac{N}{K}\max\left\{L, \log^2\frac{1}{p_{\text{tar}}}\right\}\right). \tag{A.47}$$

When the energy-reliability tradeoff function $\epsilon_{\text{and}}(u) = \epsilon_{\text{xor}}(u) = \epsilon_{\text{maj}}(u) = \exp(-c\sqrt{u}), c > 0$, the total energy consumption per bit

$$\begin{aligned} E_{\text{per-bit}} &= \frac{3N+P}{K}\left(L - L_{\text{vs}}\right)\left(\frac{1}{c}\log\frac{1}{\lambda}\right)^2 + \frac{3N+P}{K}\sum_{i=1}^{L_{\text{vs}}}\left(\frac{1}{c}\log\frac{1}{\lambda} + \frac{i-1}{c}\log\frac{1}{1-\frac{1}{2}\theta}\right)^2 \\ &= \frac{3N+P}{K}\left[L\left(\frac{1}{c}\log\frac{1}{\lambda}\right)^2 + \frac{2}{c}\log\frac{1}{\lambda}\cdot\frac{1}{c}\log\frac{1}{1-\frac{1}{2}\theta}\frac{(L_{\text{vs}}-1)L_{\text{vs}}}{2}\right. \\ &\qquad \left. + \frac{1}{c^2}\log^2\frac{1}{1-\frac{1}{2}\theta}\frac{1}{6}(L_{\text{vs}}-1)L_{\text{vs}}(2L_{\text{vs}}-1)\right] \\ &= \Theta\left(\frac{N}{K}\max\left\{L, \log^3\frac{1}{p_{\text{tar}}}\right\}\right). \end{aligned} \tag{A.48}$$

Appendix B

Theoretical proofs for Chapter 4

B.1 Proofs for Section 4.3

B.1.1 Proof of Theorem 4.3.1

We first examine the change of distortion on an arbitrary link $v_b \to v_a$ as shown in Fig. 4.1. Then, we prove this theorem by summing up all distortion on all links. By definition, we have

$$\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}} = \mathbb{E}\left[\mathbf{y}_{\mathcal{S}}|I_b\right], \tag{B.1}$$

where I_b denotes all available information at the node v_b. Similarly, we have

$$\widehat{\mathbf{y}}_{\mathcal{S},a}^{\text{mmse}} = \mathbb{E}\left[\mathbf{y}_{\mathcal{S}}|I_a\right]. \tag{B.2}$$

However, since the only information available at v_a to estimate $\mathbf{y}_{\mathcal{S}}$ is $M_{b\to a}$, because the data $\mathbf{x}_i$'s are uncorrelated, we have that

$$\widehat{\mathbf{y}}_{\mathcal{S},a}^{\text{mmse}} = \mathbb{E}\left[\mathbf{y}_{\mathcal{S}}|M_{b\to a}\right]. \tag{B.3}$$

It is certain that $M_{b\to a}$, the message bits transmitted from node v_b to node v_a, must be a function of all the available information in v_b. This means that $\sigma\left(M_{b\to a}\right) \subset \sigma\left(I_b\right)$, where $\sigma(\cdot)$ denotes the σ-algebra generated by the argument and $\sigma\left(I_b\right)$ denotes all the available information including the observations of all random variables at node v_b. Since $\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}}$, the conditional expectation estimate of $\mathbf{y}_{\mathcal{S}}$ given all the available information in v_b, is the projection of $\mathbf{y}_{\mathcal{S}}$ onto $\sigma\left(I_b\right)$ and $\widehat{\mathbf{y}}_{\mathcal{S},a}^{\text{mmse}}$ is the projection of $\mathbf{y}_{\mathcal{S}}$ onto $\sigma\left(M_{b\to a}\right) \subset \sigma\left(I_b\right)$, we have that $\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}} - \widehat{\mathbf{y}}_{\mathcal{S},a}^{\text{mmse}}$ is $\sigma\left(I_b\right)$-measurable. Therefore, using the orthogonality principle (Lemma 1), we can show that

$$\left(\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}} - \widehat{\mathbf{y}}_{\mathcal{S},a}^{\text{mmse}}\right) \perp \left(\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}} - \mathbf{y}_{\mathcal{S}}\right), \tag{B.4}$$

where the LHS is $\sigma\left(I_b\right)$-measurable, and the RHS is the projection error of the conditional expectation estimate $\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}}$ (Lemma 1 basically says that the projection error $\mathbb{E}[X|\mathcal{G}] - X$ between the original vector X and the projection (conditional expectation) $\mathbb{E}[X|\mathcal{G}]$ is uncorrelated of the sigma-algebra $\mathcal{G}$, i.e., all $\mathcal{G}$-measurable random variables). Therefore, using Pythagoras theorem and the observation that $\mathbb{E}[\widehat{\mathbf{y}}_{\mathcal{S},b}^{\mathrm{mmse}}] = \mathbb{E}[\widehat{\mathbf{y}}_{\mathcal{S},a}^{\mathrm{mmse}}] = \mathbb{E}[\mathbf{y}_{\mathcal{S}}] = \mathbf{0}_N$, we get

$$D_b^{\mathrm{Rx}} = D_b^{\mathrm{Tx}} + D_b^{\mathrm{Inc}}, \tag{B.5}$$

where, recall that $D_b^{\mathrm{Tx}} = \frac{1}{N}\mathbb{E}\left[\left\|\mathbf{y}_{\mathcal{S}} - \widehat{\mathbf{y}}_{\mathcal{S},b}^{\mathrm{mmse}}\right\|_2^2\right]$, $D_b^{\mathrm{Rx}} = \frac{1}{N}\mathbb{E}\left[\left\|\mathbf{y}_{\mathcal{S}} - \widehat{\mathbf{y}}_{\mathcal{S},a}^{\mathrm{mmse}}\right\|_2^2\right]$, and $D_b^{\mathrm{Inc}} = \frac{1}{N}\mathbb{E}\left[\left\|\widehat{\mathbf{y}}_{\mathcal{S},b}^{\mathrm{mmse}} - \widehat{\mathbf{y}}_{\mathcal{S},a}^{\mathrm{mmse}}\right\|_2^2\right]$. Since the link $v_b \to v_a$ is arbitrarily chosen, equation (B.5) can be generalized to all nodes, and hence (4.11) is proved.

Now, we show that the distortion D_b^{Tx} can be written as the sum of the distortions from the children of v_b. Without loss of generality, suppose the node v_b has d children $v_1, v_2, \ldots v_d$, as shown in Fig. 4.1. By definition, we have

$$\mathbf{y}_{\mathcal{S}} = \sum_{k=1}^{d} \mathbf{y}_{\mathcal{S}_k} + w_b \mathbf{x}_b. \tag{B.6}$$

By the definition of MMSE estimator, we have that

$$\widehat{\mathbf{y}}_{\mathcal{S},b}^{\mathrm{mmse}} = \mathbb{E}\left[\mathbf{y}_{\mathcal{S}} | I_b\right] = \mathbb{E}\left[\sum_{k=1}^{d} \mathbf{y}_{\mathcal{S}_k} + w_b \mathbf{x}_b | I_b\right] = \sum_{k=1}^{d} \widehat{\mathbf{y}}_{\mathcal{S}_k,b}^{\mathrm{mmse}} + w_b \mathbf{x}_b. \tag{B.7}$$

Therefore, we have

$$D_b^{\mathrm{Tx}} = \frac{1}{N}\mathbb{E}\left[\left\|\mathbf{y}_{\mathcal{S}} - \widehat{\mathbf{y}}_{\mathcal{S},b}^{\mathrm{mmse}}\right\|_2^2\right] \overset{(a)}{=} \frac{1}{N}\sum_{k=1}^{d} \mathbb{E}\left[\left\|\mathbf{y}_{\mathcal{S}_k} - \widehat{\mathbf{y}}_{\mathcal{S}_k,b}^{\mathrm{mmse}}\right\|_2^2\right] = \sum_{k=1}^{d} D_k^{\mathrm{Rx}}, \tag{B.8}$$

where (a) holds because different estimates $\widehat{\mathbf{y}}_{\mathcal{S}_k,b}^{\mathrm{mmse}}$ on different links $v_k \to v_b$ are independent of each other.

Combining (B.8) with (B.5), we have that

$$D_b^{\mathrm{Tx}} = \sum_{k=1}^{d} \left(D_k^{\mathrm{Tx}} + D_k^{\mathrm{Inc}}\right). \tag{B.9}$$

Using (B.9), we can prove (4.12) using induction in the tree (see Remark 5). Equation (4.13) is obtained by carrying out the induction in the tree until the sink node v_0.

B.1.2 Proof of Theorem 4.3.2

We still consider the specific set $\mathcal{S}$ as shown in Fig. 4.1. On the link $v_b \to v_a$, we have that

$$\begin{aligned}
NR_{b\to a} &\overset{(a)}{\geq} H(M_{b\to a}) \\
&\geq I(M_{b\to a}; \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b}) \\
&\overset{(b)}{=} I(M_{b\to a}, \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},a}; \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b}) \\
&\overset{(c)}{=} I(\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},a}; \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b}) + I(M_{b\to a}; \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b} | \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},a}) \\
&\geq I(\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},a}; \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b}) \\
&= h(\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b}) - h(\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b} | \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},a}) \\
&= h(\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b}) - h(\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b} - \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},a} | \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},a}) \\
&\geq h(\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b}) - h(\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b} - \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},a}) \\
&\overset{(d)}{\geq} h(\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b}) - \frac{N}{2}\log_2 2\pi e D^{\text{Inc}}_b,
\end{aligned} \tag{B.10}$$

where

(a) holds because $M_{b\to a}$ is a binary information sequence;

(b) holds because $\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},a}$ is a function of $M_{b\to a}$;

(c) follows from the chain rule for mutual information;

(d) holds because the entropy-maximizing distribution under variance constraint is Gaussian.

Now we only need to lower-bound $h(\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b})$. We know that

$$\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b} = \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S}\setminus\{b\},b} + w_b \mathbf{x}_b \tag{B.11}$$

$$\mathbf{y}_{\mathcal{S}} = \mathbf{y}_{\mathcal{S}\setminus\{b\}} + w_b \mathbf{x}_b. \tag{B.12}$$

Suppose $\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b} \sim r\left(x^N\right)$ and $\mathbf{y}_{\mathcal{S}} \sim s\left(x^N\right)$. Observe that (B.11) and (B.12) are in the form of the random variables in Lemma 4.1.2 with $t = w_b^2$ and $\mathbf{z} = \mathbf{x}_b \sim \mathcal{N}(\mathbf{0}_N, \mathbf{I}_N)$. Then, using Lemma 4.1.2, we have that

$$D\left(r||s\right) \leq \frac{1}{2w_b^2}\mathbb{E}\left[\left\|\mathbf{y}_{\mathcal{S}\setminus\{b\}} - \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S}\setminus\{b\},b}\right\|_2^2\right] = \frac{1}{2w_b^2}\mathbb{E}\left[\left\|\mathbf{y}_{\mathcal{S}} - \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S},b}\right\|_2^2\right] = \frac{ND_b^{\text{Tx}}}{2w_b^2}. \tag{B.13}$$

By definition, we have that

$$\mathbf{y}_{\mathcal{S}} \sim s\left(x^N\right) = \frac{1}{\left(\sqrt{2\pi}\sigma_{\mathcal{S}}\right)^N}\exp\left(-\frac{\left\|x^N\right\|_2^2}{2\sigma_{\mathcal{S}}^2}\right). \tag{B.14}$$

Therefore,

$$h(\mathbf{y}_{\mathcal{S}}) = \frac{N}{2}\log_2 2\pi e\sigma_{\mathcal{S}}^2. \tag{B.15}$$

The difference between $h(\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}})$ and $h(\mathbf{y}_{\mathcal{S}})$ is

$$\begin{aligned} h(\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}}) - h(\mathbf{y}_{\mathcal{S}}) &= -\int_{x^N \in R^N} r \log r dx^N + \int_{x^N \in R^N} s \log s dx^N \\ &= -\int_{x^N \in R^N} r \log \frac{r}{s} dx^N + \int_{x^N \in R^N} (s-r) \log s dx^N \\ &\overset{(a)}{=} -D\left(r||s\right) + \log_2 e \int_{x^N \in R^N} (s-r) \left(-\frac{\left\|x^N\right\|_2^2}{2\sigma_{\mathcal{S}}^2} \right) dx^N \\ &= -D\left(r||s\right) + \frac{\log_2 e}{2\sigma_{\mathcal{S}}^2} \mathbb{E}\left[\left\|\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}}\right\|_2^2 - \left\|\mathbf{y}_{\mathcal{S}}\right\|_2^2 \right], \end{aligned} \tag{B.16}$$

where we used (B.14) in step (a). The second term of the RHS can be bounded by

$$\begin{aligned} \left| \mathbb{E}\left[\left\|\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}}\right\|_2^2 - \left\|\mathbf{y}_{\mathcal{S}}\right\|_2^2 \right] \right| &= \left| \mathbb{E}\left[\left(\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}} - \mathbf{y}_{\mathcal{S}}\right)^\top \left(\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}} + \mathbf{y}_{\mathcal{S}}\right) \right] \right| \\ &\leq \sqrt{ \mathbb{E}\left[\left\|\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}} - \mathbf{y}_{\mathcal{S}}\right\|_2^2 \right] \mathbb{E}\left[\left\|\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}} + \mathbf{y}_{\mathcal{S}}\right\|_2^2 \right] } \\ &= \sqrt{ N D_b^{\text{Tx}} \mathbb{E}\left[\left\|\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}} - \mathbf{y}_{\mathcal{S}} + 2\mathbf{y}_{\mathcal{S}}\right\|_2^2 \right] } \\ &\leq \sqrt{ N D_b^{\text{Tx}} \cdot 2 \left\{ \mathbb{E}\left[4 \left\|\mathbf{y}_{\mathcal{S}}\right\|_2^2 \right] + \mathbb{E}\left[\left\|\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}} - \mathbf{y}_{\mathcal{S}}\right\|_2^2 \right] \right\} } \\ &= \sqrt{ 2 N D_b^{\text{Tx}} \left(4N\sigma_{\mathcal{S}}^2 + N D_b^{\text{Tx}} \right) }. \end{aligned} \tag{B.17}$$

Therefore, combining (B.13) and (B.15)-(B.17), we get

$$\begin{aligned} h(\widehat{\mathbf{y}}_{\mathcal{S},b}^{\text{mmse}}) &\geq h(\mathbf{y}_{\mathcal{S}}) - \frac{N D_b^{\text{Tx}}}{2w_b^2} - \frac{N \log_2 e}{2\sigma_{\mathcal{S}}^2} \sqrt{2 D_b^{\text{Tx}} \left(4\sigma_{\mathcal{S}}^2 + D_b^{\text{Tx}}\right)} \\ &= \frac{N}{2} \log_2 2\pi e \sigma_{\mathcal{S}}^2 - \frac{N D_b^{\text{Tx}}}{2w_b^2} - \frac{N \log_2 e}{2\sigma_{\mathcal{S}}^2} \sqrt{2 D_b^{\text{Tx}} \left(4\sigma_{\mathcal{S}}^2 + D_b^{\text{Tx}}\right)}. \end{aligned} \tag{B.18}$$

Plugging the above inequality into (B.10), we get

$$\begin{aligned} R_{b \to a} &\geq \frac{1}{2} \log_2 \frac{\sigma_{\mathcal{S}}^2}{D_b^{\text{Inc}}} - \frac{D_b^{\text{Tx}}}{2w_b^2} - \frac{\log_2 e}{2\sigma_{\mathcal{S}}^2} \sqrt{2 D_b^{\text{Tx}} \left(4\sigma_{\mathcal{S}}^2 + D_b^{\text{Tx}}\right)} \\ &= \frac{1}{2} \log_2 \frac{\sigma_{\mathcal{S}}^2}{D_b^{\text{Inc}}} - \mathcal{O}\left((D_b^{\text{Tx}})^{1/2} \right), \end{aligned} \tag{B.19}$$

in the limit of small D_b^{Tx}. Summing (B.19) over all links, we get

$$\begin{aligned} \sum_{i=1}^n R_{i \to \text{PN}(i)} &\geq \frac{1}{2} \sum_{i=1}^n \left[\log_2 \frac{\sigma_{\mathcal{S}_i}^2}{D_i^{\text{Inc}}} - \frac{D_i^{\text{Tx}}}{2w_i^2} - \frac{\log_2 e}{2\sigma_{\mathcal{S}_i}^2} \sqrt{2 D_i^{\text{Tx}} \left(4\sigma_{\mathcal{S}_i}^2 + D_i^{\text{Tx}}\right)} \right]. \\ &= \frac{1}{2} \sum_{i=1}^n \left[\log_2 \frac{\sigma_{\mathcal{S}_i}^2}{D_i^{\text{Rx}} - D_i^{\text{Tx}}} - \mathcal{O}\left((D_i^{\text{Tx}})^{1/2} \right) \right], \end{aligned} \tag{B.20}$$

in the limit of small $D_i^{\mathrm{Tx}}, \forall i$. The last equality in (B.20) can be obtained using $D_i^{\mathrm{Rx}} = D_i^{\mathrm{Tx}} + D_i^{\mathrm{Inc}}$ (see the distortion accumulation equation (4.11)). The optimization bound shown in (4.15) is basically the same bound (B.20) stated in an optimization form over the choices of the incremental distortions D_i^{Inc}. Now, we prove that the solution of the optimization satisfies (4.17), which finally leads to the order-sense bound (4.18). When the constraints in (4.15) are satisfied,

$$\begin{aligned} R &\geq \frac{1}{2}\sum_{i=1}^{n}\left[\log_2\frac{\sigma_{\mathcal{S}_i}^2}{D_i^{\mathrm{Inc}}} - \psi_i\left(D_i^{\mathrm{Tx}}\right)\right] \\ &\overset{(a)}{\geq} \frac{1}{2}\log_2\frac{\prod_{i=1}^{n}\sigma_{\mathcal{S}_i}^2}{\prod_{i=1}^{n}D_i^{\mathrm{Inc}}} - \frac{1}{2}\sum_{i=1}^{n}\psi_i\left(D_0^{\mathrm{mmse}}\right) \\ &\overset{(b)}{\geq} \frac{1}{2}\log_2\frac{\prod_{i=1}^{n}\sigma_{\mathcal{S}_i}^2}{\left(D_0^{\mathrm{mmse}}/n\right)^n} - \frac{1}{2}\sum_{i=1}^{n}\psi_i\left(D_0^{\mathrm{mmse}}\right) \\ &\overset{(c)}{\geq} \frac{1}{2}\log_2\frac{\prod_{i=1}^{n}\sigma_{\mathcal{S}_i}^2}{\left(D/n\right)^n} - \frac{1}{2}\sum_{i=1}^{n}\psi_i\left(D\right), \end{aligned} \tag{B.21}$$

where (a) holds because $D_i^{\mathrm{Tx}} < D_0^{\mathrm{mmse}}$ (which can be easily seen by comparing (4.12) and (4.13)) and the functions $\psi_i(\cdot)$, $i = 1, \ldots n$ are monotone, (b) follows from the constraint $D_0^{\mathrm{mmse}} = \sum_{i=1}^{n} D_i^{\mathrm{Inc}}$ in (4.13) and the fact that the arithmetic mean is greater or equal to the geometric mean, and (c) follows from the inequality $D_0^{\mathrm{mmse}} \leq D$ in (4.10). Further, using the fact that $\psi_i(D) = \frac{D}{2w_i^2} + \frac{\log_2 e}{2\sigma_{\mathcal{S}_i}^2}\sqrt{2D\left(4\sigma_{\mathcal{S}_i}^2 + D\right)} = \mathcal{O}(\sqrt{D})$, we obtain the lower bound (4.18).

B.1.3 Proof of Theorem 4.3.3

We still look at a specific set $\mathcal{S}$ as shown in Fig. 4.1. Then, we have

$$\begin{aligned} NR_{b\to a} &\geq H(M_{b\to a}) \\ &\geq I(M_{b\to a};\mathbf{y}_{\mathcal{S}}) \\ &= h(\mathbf{y}_{\mathcal{S}}) - h(\mathbf{y}_{\mathcal{S}}|M_{b\to a}) \\ &= h(\mathbf{y}_{\mathcal{S}}) - h(\mathbf{y}_{\mathcal{S}}|M_{b\to a}, \widehat{\mathbf{y}}_{\mathbf{S},a}^{\mathrm{mmse}}) \\ &= h(\mathbf{y}_{\mathcal{S}}) - h(\mathbf{y}_{\mathcal{S}} - \widehat{\mathbf{y}}_{\mathbf{S},a}^{\mathrm{mmse}}|\widehat{\mathbf{y}}_{\mathbf{S},a}^{\mathrm{mmse}}, M_{b\to a}) \\ &\geq h(\mathbf{y}_{\mathcal{S}}) - h(\mathbf{y}_{\mathcal{S}} - \widehat{\mathbf{y}}_{\mathbf{S},a}^{\mathrm{mmse}}) \\ &\geq \frac{N}{2}\log_2 2\pi e\sigma_{\mathcal{S}}^2 - \frac{N}{2}\log_2 2\pi e D_b^{\mathrm{Rx}} = \frac{N}{2}\log_2\frac{\sigma_{\mathcal{S}}^2}{D_b^{\mathrm{Rx}}}. \end{aligned} \tag{B.22}$$

Summing (B.22) over all links, we get the outer bound (4.19).

B.2 Proofs for Section 4.4

B.2.1 A review on Gaussian test channels

First, we elaborate on the details of Gaussian test channels. Suppose a transmitter has a source $X \sim \mathcal{N}(0, P)$ and wishes to send an approximate description $\widehat{X}$ to a receiver with distortion D. Then

$$R(D) = \min_{p(\widehat{x}|x):\mathbb{E}\left[(X-\widehat{X})^2\right]\le D} I(X;\widehat{X}) = \frac{1}{2}\log\frac{P}{D}, \forall P \ge D. \tag{B.23}$$

The "test channel" in this case is the inverse Gaussian channel

$$X = \widehat{X} + Z, \tag{B.24}$$

where $Z \sim \mathcal{N}(0, D)$ is an additive noise independent of $\widehat{X}$ (see [59, Theorem 10.3.2]). The test channel is useful for understanding orthogonality properties of codewords in random codebooks. To achieve the rate in (B.23), we can use a random code $\{\widehat{\mathbf{c}}(w) : w \in \{1, 2, \ldots 2^{NR}\}\}$ with joint typicality encoding and decoding, where each codeword $\widehat{\mathbf{c}}(w)$ is generated i.i.d. with each entry distributed as $\mathcal{N}(0, P - D)$. When $N \to \infty$, the rate (B.23) is asymptotically achieved.

B.2.2 Proof of (4.38)

We use induction in the tree (see Remark 5) to prove (4.38). For an arbitrary leaf v_l, we know that $\widehat{\sigma}_l^2 = \sigma_{\mathcal{S}_l}^2 = w_l^2$. For an arbitrary non-leaf node v_b, we have that (see (4.32))

$$\widehat{\sigma}_b^2 = \sum_{k=1}^{d} (\widehat{\sigma}_k^2 - d_b) + w_b^2. \tag{B.25}$$

By definition, we have

$$\mathbf{y}_{\mathcal{S}} = \sum_{k=1}^{d} \mathbf{y}_{\mathcal{S}_k} + w_b \mathbf{x}_b, \tag{B.26}$$

which means

$$\sigma_{\mathcal{S}_b}^2 = \sum_{k=1}^{d} \sigma_{\mathcal{S}_k}^2 + w_b w_b^2, \tag{B.27}$$

Comparing (B.25) and (B.27), we know that, if (4.38) holds at all children of v_b, it also holds at v_b. Thus, by induction in the tree, we can show that (4.38) is true.

B.2.3 Proof of Lemma 4.4.5

To simplify notation, for an arbitrary node v_b and its parent node $v_a = v_{\mathrm{PN}(b)}$ define

$$\widehat{\mathbf{s}}_b^* = \widehat{\mathbf{y}}_{\mathcal{S}_b,b}^{\mathrm{mmse}}, \tag{B.28}$$

$$\widehat{\mathbf{r}}_b^* = \widehat{\mathbf{y}}_{\mathcal{S}_b,\mathrm{PN}(b)}^{\mathrm{mmse}}. \tag{B.29}$$

Therefore, $\widehat{\mathbf{s}}_b^*$ is the MMSE estimate of the partial sum $\mathbf{y}_{\mathcal{S}_b}$ at the node v_b, while $\widehat{\mathbf{r}}_b^*$ is the MMSE estimate of the same variable, but at the parent-node $v_{\text{PN}(b)}$. In order to relate the Gaussian-code-based distortion and the MMSE-based distortion, we will prove that, the estimates based on the Gaussian code, i.e., the estimate $\widehat{\mathbf{s}}_b$ and the description $\widehat{\mathbf{r}}_b$, are very close to the MMSE estimates $\widehat{\mathbf{s}}_b^*$ and $\widehat{\mathbf{r}}_b^*$ in the sense of mean-square error[1]. We prove that as long as N is finite but sufficiently large, the gap between these two types of estimators can be arbitrarily small. Define

$$\Delta_b^{\text{Tx}} = \mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{s}}_b - \widehat{\mathbf{s}}_b^*\right\|_2^2\right], \tag{B.30}$$

$$\Delta_b^{\text{Rx}} = \mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}_b - \widehat{\mathbf{r}}_b^*\right\|_2^2\right]. \tag{B.31}$$

We will prove that $\Delta_b^{\text{Tx}} \to 0$ and $\Delta_b^{\text{Rx}} \to 0$ when $N \to \infty$. In particular, we will prove the following three statements:

Statement 1: For an arbitrary leaf v_l,

$$\Delta_l^{\text{Tx}} = 0. \tag{B.32}$$

Statement 2: For an arbitrary non-leaf node v_b and its d children $v_1, \ldots v_d$ (see Fig. 4.1),

$$\sqrt{\Delta_b^{\text{Tx}}} \le \sum_{k=1}^{d} \sqrt{\Delta_k^{\text{Rx}}}. \tag{B.33}$$

Statement 3: For an arbitrary node v_b,

$$\sqrt{\Delta_b^{\text{Rx}}} \le \sqrt{\theta_N} + \sqrt{\Delta_b^{\text{Tx}}}, \tag{B.34}$$

where $\lim_{N\to\infty} \theta_N = 0$.

Proof of Statement 1

For a leaf v_l, the random-coding-based estimate is $\widehat{\mathbf{s}}_l = w_l \mathbf{x}_l$, which is exactly the same as the MMSE estimate $\widehat{\mathbf{s}}_l^*$, since $\mathbf{x}_l$ is known to v_l. Therefore, $\Delta_l^{\text{Tx}} = 0$.

Proof of Statement 2

For a non-leaf node v_b and its children, we have that (see (4.40))

$$\widehat{\mathbf{s}}_b = \sum_{k=1}^{d} \mathbf{c}_k(M_{k\to b}) + w_b \mathbf{x}_b = \sum_{k=1}^{d} \widehat{\mathbf{r}}_b + w_b \mathbf{x}_b. \tag{B.35}$$

[1]Note that according to the intuitive explanation on test channels (see Remark 7), the estimates based on the Gaussian code and the MMSE estimations are indeed equal to each other when Gaussian test channels can be physically established.

Since the partial sum $\mathbf{y}_{\mathcal{S}_b} = \sum_{k=1}^{d} \mathbf{y}_{\mathcal{S}_k} + w_b \mathbf{x}_b$, we have that

$$\widehat{\mathbf{s}}_b^* = \mathbb{E}\left[\mathbf{y}_{\mathcal{S}_b} \left| I_b \right.\right] = \sum_{k=1}^{d} \mathbb{E}\left[\mathbf{y}_{\mathcal{S}_k} \left| I_b \right.\right] + w_b \mathbf{x}_b = \sum_{k=1}^{d} \widehat{\mathbf{r}}_k^* + w_b \mathbf{x}_b. \tag{B.36}$$

Thus, combining (B.35) and (B.36), we get

$$\Delta_b^{\text{Tx}} = \left[\|\widehat{\mathbf{s}}_b - \widehat{\mathbf{s}}_b^*\|_2^2\right] = \sum_{k=1}^{d} \mathbb{E}\left[\|\widehat{\mathbf{r}}_k - \widehat{\mathbf{r}}_k^*\|_2^2\right] = \sum_{k=1}^{d} \Delta_k^{\text{Rx}}, \tag{B.37}$$

which can be further relaxed by

$$\sqrt{\Delta_b^{\text{Tx}}} < \sum_{k=1}^{d} \sqrt{\Delta_k^{\text{Rx}}}. \tag{B.38}$$

Proof of Statement 3

Note that by (4.45), we have

$$\mathbb{E}\left[\frac{1}{N} \|\widehat{\mathbf{s}}_b - \widehat{\mathbf{r}}_b\|_2^2\right] \leq d_b + \varepsilon_N. \tag{B.39}$$

Define $\text{Dist}_b = \mathbb{E}_{\mathcal{C}_b}\left[\frac{1}{N}\left\|\mathbb{E}_{\mathcal{C}_b}\left[\widehat{\mathbf{s}}_b \left| \widehat{\mathbf{r}}_b \right.\right] - \widehat{\mathbf{s}}_b\right\|_2^2\right]$. We will prove that Dist_b is approximately greater than d_b (the explicit form is in (B.41)), which means that even the MMSE estimate $\mathbb{E}_{\mathcal{C}_b}\left[\widehat{\mathbf{s}}_b \left| \widehat{\mathbf{r}}_b \right.\right]$ cannot provide a much better description (in the sense of mean-square error) of $\widehat{\mathbf{s}}_b$ than the typicality-based estimate $\widehat{\mathbf{r}}_b$. Notice that the MMSE estimate $\mathbb{E}_{\mathcal{C}_b}\left[\widehat{\mathbf{s}}_b | \widehat{\mathbf{r}}_b\right]$ here should be defined for the chosen codebook $\mathcal{C}_b$ at v_b, since the receiver v_a also knows the codebook. The outer $\mathbb{E}$ in $\text{Dist}_b = \mathbb{E}_{\mathcal{C}_b}\left[\frac{1}{N}\left\|\mathbb{E}_{\mathcal{C}_b}\left[\widehat{\mathbf{s}}_b \left| \widehat{\mathbf{r}}_b \right.\right] - \widehat{\mathbf{s}}_b\right\|_2^2\right]$ is also conditioned on a given codebook $\mathcal{C}_b$ at node v_b. From (4.37) we have that

$$\begin{aligned}
\frac{N}{2}\log\frac{\widehat{\sigma}_b^2}{d_b} + N\delta_N =& NR_b \\
&\overset{(a)}{\geq} I(\widehat{\mathbf{s}}_b; \widehat{\mathbf{r}}_b) \\
&\overset{(b)}{\geq} I\left(\widehat{\mathbf{s}}_b; \mathbb{E}_{\mathcal{C}_b}\left[\widehat{\mathbf{s}}_b \left| \widehat{\mathbf{r}}_b \right.\right]\right) \\
&= h(\widehat{\mathbf{s}}_b) - h\left(\widehat{\mathbf{s}}_b | \mathbb{E}_{\mathcal{C}_b}\left[\widehat{\mathbf{s}}_b \left| \widehat{\mathbf{r}}_b \right.\right]\right) \\
&= h(\widehat{\mathbf{s}}_b) - h\left(\widehat{\mathbf{s}}_b - \mathbb{E}_{\mathcal{C}_b}\left[\widehat{\mathbf{s}}_b \left| \widehat{\mathbf{r}}_b \right.\right] \left| \mathbb{E}_{\mathcal{C}_b}\left[\widehat{\mathbf{s}}_b \left| \widehat{\mathbf{r}}_b \right.\right] \right.\right) \\
&\geq h(\widehat{\mathbf{s}}_b) - h\left(\widehat{\mathbf{s}}_b - \mathbb{E}_{\mathcal{C}_b}\left[\widehat{\mathbf{s}}_b \left| \widehat{\mathbf{r}}_b \right.\right]\right) \\
&\overset{(c)}{>} \frac{N}{2}\log_2 2\pi e \widehat{\sigma}_b^2 - N\beta_N - \frac{N}{2}\log_2 2\pi e \text{Dist}_b,
\end{aligned} \tag{B.40}$$

where (a) follows from the cut set bound, (b) follows from the data processing inequality, and (c) follows from Lemma 4.4.4. Notice that although the codebook $\mathcal{C}_b$ is fixed, other codebooks are not fixed, so the random vector $h(\widehat{\mathbf{s}}_b)$ still satisfies Lemma 4.4.4. Therefore,

$$\text{Dist}_b > 2^{-\delta_N-\beta_N} d_b = (1-\epsilon_N) d_b, \tag{B.41}$$

where $\lim_{N\to\infty} \epsilon_N = 0$. Since the inequality (B.41) holds for any given codebook $\mathcal{C}_b$, (B.41) also holds for the entire random codebook ensemble, in which case the outside $\mathbb{E}$ is again taken over the random codebook generation (which is in alignment with the definitions of other mean-square distortions in other parts of this section and all other sections). Combining (B.39) and (B.41) and the orthogonality principle

$$\left(\mathbb{E}\left[\widehat{\mathbf{s}}_b \left|\widehat{\mathbf{r}}_b\right.\right] - \widehat{\mathbf{s}}_b\right) \perp \left(\widehat{\mathbf{r}}_b - \mathbb{E}\left[\widehat{\mathbf{s}}_b \left|\widehat{\mathbf{r}}_b\right.\right]\right),$$

we get

$$\mathbb{E}\left[\frac{1}{N}\left\|\mathbb{E}\left[\widehat{\mathbf{s}}_b \left|\widehat{\mathbf{r}}_b\right.\right] - \widehat{\mathbf{r}}_b\right\|_2^2\right] \le d_b + \varepsilon_N - (1-\epsilon_N)d_b = \varepsilon_N + \epsilon_N d_b =: \theta_N, \tag{B.42}$$

where $\lim_{N\to\infty} \theta_N = 0$. Further, we have that

$$\widehat{\mathbf{r}}_b^* = \mathbb{E}\left[\mathbf{y}_{\mathcal{S}_b}\left|I_{\text{PN}(b)}\right.\right] = \mathbb{E}\left[\mathbf{y}_{\mathcal{S}_b}\left|\widehat{\mathbf{r}}_b\right.\right] \overset{(a)}{=} \mathbb{E}\left[\mathbb{E}\left[\mathbf{y}_{\mathcal{S}_b}|I_b\right]\left|\widehat{\mathbf{r}}_b\right.\right] = \mathbb{E}\left[\widehat{\mathbf{s}}_b^*\left|\widehat{\mathbf{r}}_b\right.\right], \tag{B.43}$$

where the equality (a) follows from the iterative expectation principle and the fact that $\widehat{\mathbf{r}}_b$ is a function of I_b. Therefore

$$\begin{aligned}
&\mathbb{E}\left[\left\|\mathbb{E}\left[\widehat{\mathbf{s}}_b\left|\widehat{\mathbf{r}}_b\right.\right] - \widehat{\mathbf{r}}_b^*\right\|_2^2\right] = \mathbb{E}\left[\left\|\mathbb{E}\left[\widehat{\mathbf{s}}_b - \widehat{\mathbf{s}}_b^*\left|\widehat{\mathbf{r}}_b\right.\right]\right\|_2^2\right] \\
&\overset{(a)}{\le} \mathbb{E}\left[\mathbb{E}\left[\left\|\widehat{\mathbf{s}}_b - \widehat{\mathbf{s}}_b^*\right\|_2^2\left|\widehat{\mathbf{r}}_b\right.\right]\right] = \mathbb{E}\left[\left\|\widehat{\mathbf{s}}_b - \widehat{\mathbf{s}}_b^*\right\|_2^2\right] = N\Delta_b^{\text{Tx}},
\end{aligned} \tag{B.44}$$

where inequality (a) follows from the Jensen's inequality. Thus, combining (B.42) and (B.44) and using the triangle inequality, we get

$$\begin{aligned}
\sqrt{\Delta_b^{\text{Rx}}} =& \sqrt{\mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}_b^* - \widehat{\mathbf{r}}_b\right\|_2^2\right]} \\
\le& \sqrt{\mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}_b^* - \mathbb{E}\left[\widehat{\mathbf{s}}_b\left|\widehat{\mathbf{r}}_b\right.\right]\right\|_2^2\right]} + \sqrt{\mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}_b - \mathbb{E}\left[\widehat{\mathbf{s}}_b\left|\widehat{\mathbf{r}}_b\right.\right]\right\|_2^2\right]} \\
\le& \sqrt{\theta_N} + \sqrt{\Delta_b^{\text{Tx}}}.
\end{aligned} \tag{B.45}$$

Using Statement 1-3 to prove Lemma 4.4.5

Using the three statements and using induction on the tree, we have that

$$\sup_{1\le b\le n} \sqrt{\Delta_b^{\text{Tx}}} \le \sup_{1\le b\le n} \sqrt{\Delta_b^{\text{Rx}}} \le n\sqrt{\theta_N}. \tag{B.46}$$

Thus, the conclusion (4.48) can be obtained by combining the orthogonality principle

$$\mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}_b^* - \widehat{\mathbf{s}}_b^*\right\|_2^2\right] = \mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}_b^* - \mathbf{y}_{\mathcal{S}_b}\right\|_2^2\right] - \mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{s}}_b^* - \mathbf{y}_{\mathcal{S}_b}\right\|_2^2\right] = D_b^{\mathrm{Rx}} - D_b^{\mathrm{Tx}} \tag{B.47}$$

and the triangle inequality, which is

$$\begin{aligned}\sqrt{\mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}_b^* - \widehat{\mathbf{s}}_b^*\right\|_2^2\right]} &\leq \sqrt{\mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{s}}_b - \widehat{\mathbf{s}}_b^*\right\|_2^2\right]} + \sqrt{\mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}_b - \widehat{\mathbf{r}}_b^*\right\|_2^2\right]} + \sqrt{\mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}_b - \widehat{\mathbf{s}}_b\right\|_2^2\right]} \\ &\leq \sqrt{d_b + \varepsilon_N} + 2n\sqrt{\theta_N},\end{aligned} \tag{B.48}$$

and

$$\begin{aligned}\sqrt{\mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}_b^* - \widehat{\mathbf{s}}_b^*\right\|_2^2\right]} &\geq \sqrt{\mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{s}}_b - \widehat{\mathbf{s}}_b^*\right\|_2^2\right]} - \sqrt{\mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}_b - \widehat{\mathbf{r}}_b^*\right\|_2^2\right]} - \sqrt{\mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}_b - \widehat{\mathbf{s}}_b\right\|_2^2\right]} \\ &\geq \sqrt{d_b - \varepsilon_N} - 2n\sqrt{\theta_N}.\end{aligned} \tag{B.49}$$

B.3 Proofs for Section 4.5

B.3.1 Proof of Theorem 4.5.1

We consider a general case in Fig. 4.1, where the set $\mathcal{S}$ represents $\mathcal{S}_{b\to a}$. Using exactly the same arguments from (B.1) to (B.4), we obtain

$$\left(\widehat{\mathbf{y}}^{\mathrm{mmse}}_{\mathcal{S}_{b\to a},b} - \widehat{\mathbf{y}}^{\mathrm{mmse}}_{\mathcal{S}_{b\to a},a}\right) \perp \left(\widehat{\mathbf{y}}^{\mathrm{mmse}}_{\mathcal{S}_{b\to a},b} - \mathbf{y}_{\mathcal{S}_{b\to a}}\right). \tag{B.50}$$

Therefore, using Pythagoras theorem, we get

$$D^{\mathrm{Rx}}_{i\to j} = D^{\mathrm{Tx}}_{i\to j} + D^{\mathrm{Inc}}_{i\to j}. \tag{B.51}$$

From the definition of an MMSE estimate, we have that

$$\widehat{\mathbf{y}}^{\mathrm{mmse}}_{\mathcal{S}_{b\to a},b} = \mathbb{E}\left[\mathbf{y}_{\mathcal{S}_{b\to a}} | I_b\right] = \mathbb{E}\left[\sum_{k=1}^{d} \mathbf{y}_{\mathcal{S}_{k\to b}} + w_b \mathbf{x}_b \,| I_b\right] = \sum_{k=1}^{d} \widehat{\mathbf{y}}^{\mathrm{mmse}}_{\mathcal{S}_{k\to b},b} + w_b \mathbf{x}_b. \tag{B.52}$$

Therefore

$$D^{\mathrm{Tx}}_{b\to a} = \mathbb{E}\left[\left(\mathbf{y}_{\mathcal{S}} - \widehat{\mathbf{y}}^{\mathrm{mmse}}_{\mathcal{S}_{b\to a},b}\right)^2\right] = \sum_{k=1}^{d} \mathbb{E}\left[\left(\mathbf{y}_{\mathcal{S}_{k\to b},b} - \widehat{\mathbf{y}}^{\mathrm{mmse}}_{\mathcal{S}_{k\to b},b}\right)^2\right] = \sum_{k=1}^{d} D^{\mathrm{Rx}}_{k\to b} + D^{\mathrm{Inc}}_{k\to b}. \tag{B.53}$$

Using induction on the edge set $\overrightarrow{\mathcal{T}}_k$ of the directed tree towards the root v_k, we get (4.64).

B.3.2 Proof of Theorem 4.5.2

The main part is to show that in Fig. 4.1

$$R_{b\to a} \geq \frac{1}{2}\log_2\frac{\sigma^2_{\mathcal{S}_{b\to a}}}{D^{\text{Inc}}_{b\to a}} - \mathcal{O}\left((D^{\text{Tx}}_{b\to a})^{1/2}\right), \tag{B.54}$$

which is a counterpart of (B.19). As long as (B.54) holds, the outer bound in Theorem 4.5.2 can be obtained by summing (B.54) over all links.

The proof of (B.54) can be obtained similarly as in the proof of (B.19). We know that the set $\mathcal{S}$ in Fig. 4.1 represents $\mathcal{S}_{b\to a} \subset \mathcal{V}$. Then, using the same derivations in (B.10), we get

$$NR_{b\to a} \geq h(\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S}_{b\to a},b}) - \frac{N}{2}\log_2 2\pi e D^{\text{Inc}}_{b\to a}. \tag{B.55}$$

Using Lemma 4.1.2 and the same derivations in (B.16) and (B.17), we get

$$\begin{aligned} h(\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S}_{b\to a},b}) - h(\mathbf{y}_{\mathcal{S}_{b\to a}}) &= -D\left(p||q\right) + \frac{\log_2 e}{2\sigma^2_{\mathcal{S}_{b\to a}}}\mathbb{E}\left[\left\|\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S}_{b\to a},b}\right\|_2^2 - \left\|\mathbf{y}_{\mathcal{S}_{b\to a}}\right\|_2^2\right] \\ &\geq -\frac{ND^{\text{Tx}}_{b\to a}}{2w_b^2} - \frac{N\log_2 e}{2\sigma^2_{\mathcal{S}_{b\to a}}}\sqrt{2D^{\text{Tx}}_{b\to a}\left(4\sigma^2_{\mathcal{S}_{b\to a}} + D^{\text{Tx}}_{b\to a}\right)}, \end{aligned} \tag{B.56}$$

where $p(\cdot)$ and $q(\cdot)$ are the pdfs of $\widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S}_{b\to a},b}$ and $\mathbf{y}_{\mathcal{S}_{b\to a}}$ respectively. Combining (B.55), (B.56) and the fact that $h(\mathbf{y}_{\mathcal{S}}) = \frac{1}{2}\log_2 2\pi e\sigma^2_{\mathcal{S}}$, we get

$$\begin{aligned} R_{b\to a} \geq& \frac{1}{2}\log_2\frac{\sigma^2_{\mathcal{S}_{b\to a}}}{D^{\text{Inc}}_{b\to a}} - \frac{D^{\text{Tx}}_{b\to a}}{2w_b^2} - \frac{\log_2 e}{2\sigma^2_{\mathcal{S}_{b\to a}}}\sqrt{2D^{\text{Tx}}_{b\to a}\left(4\sigma^2_{\mathcal{S}_{b\to a}} + D^{\text{Tx}}_{b\to a}\right)} \\ =& \frac{1}{2}\log_2\frac{\sigma^2_{\mathcal{S}_{b\to a}}}{D^{\text{Inc}}_{b\to a}} - \mathcal{O}\left((D^{\text{Tx}}_{b\to a})^{1/2}\right). \end{aligned} \tag{B.57}$$

This completes the proof.

B.3.3 Proof of Theorem 4.5.3

In this proof, we provide an achievable scheme for the Gaussian network consensus problem. We basically generalize the scheme for linear function computation in Section 4.4 to the network consensus problem. Therefore, we will first use Gaussian test channels to define some distribution functions that we will use in this section. Then, we will provide the encoding and decoding procedures for the Gaussian random codes. Finally, we will prove that this scheme achieves the sum rate inner bound (4.67).

Recall that at each node v_i, $\mathbf{y}_{\mathcal{S}_{i\to j}}$ denotes the partial weighted sum of all data at all descendants of v_i when the node v_j is viewed as the parent node of v_i. Denote by $\widehat{\mathbf{s}}_{i\to j}$ the estimate of the partial sum $\mathbf{y}_{\mathcal{S}_{i\to j}}$. Denote by $\widehat{\mathbf{r}}_{i\to j}$ the description of $\widehat{\mathbf{s}}_{i\to j}$ that is sent by v_i to v_j. The formal definition of the estimates and descriptions will be provided in the encoding and decoding procedures. Following the same procedures in Section 4.4, we first define some distribution functions using Gaussian test channels. These distribution functions will be defined such that the estimates $\widehat{\mathbf{s}}_{i\to j}$ and descriptions $\widehat{\mathbf{r}}_{i\to j}$ are typical with respect to them.

At each link $v_i \to v_j$, we define two scalar random variables $U^{\text{TC}}_{i\to j}$ and $V^{\text{TC}}_{i\to j}$. Define $\widehat{\sigma}^2_{i\to j}$ as the variance of $U^{\text{TC}}_{i\to j}$. When $U^{\text{TC}}_{i\to j}$ is given, $V^{\text{TC}}_{i\to j}$ is defined by the Gaussian test channel

$$U^{\text{TC}}_{i\to j} = V^{\text{TC}}_{i\to j} + Z_{i\to j}, \tag{B.58}$$

where $Z_{i\to j} \sim \mathcal{N}(0, d_{i\to j})$ is independent of $V^{\text{TC}}_{i\to j}$ and $d_{i\to j}$ is the distortion parameter, which can be tuned.

For any arbitrary leaf v_l, define

$$U^{\text{TC}}_{l\to n(l)} = w_l X_l, \tag{B.59}$$

where X_l denotes a random variable that has the same distribution as each entry of $\mathbf{x}_l$, and $v_{n(l)}$ denotes the only neighbor of the node v_l. For an arbitrary non-leaf node v_b and an arbitrary neighbor $v_a \in \mathcal{N}(v_b)$ as shown in Fig. 4.1, define

$$U^{\text{TC}}_{b\to a} = \sum_{v_k \in \mathcal{N}(v_b)\backslash\{v_a\}} V^{\text{TC}}_{k\to b} + w_b X_b, \tag{B.60}$$

where X_b denotes a random variable that has the same distribution as each entry of $\mathbf{x}_b$. Since the network is a tree, all descriptions $V^{\text{TC}}_{k\to b}$ at different neighbors v_k of v_b are independent of each other. Therefore,

$$\widehat{\sigma}^2_{b\to a} = \sum_{k=1}^{d} (\widehat{\sigma}^2_{k\to b} - d_{k\to b}) + w_b^2. \tag{B.61}$$

Define $\phi_{U^{\text{TC}}_{i\to j}}$ and $\phi_{V^{\text{TC}}_{i\to j}}$ as distribution functions of $U^{\text{TC}}_{i\to j}$ and $V^{\text{TC}}_{i\to j}$. We also use joint pdfs, where the meanings are always clear from the context. Note that Gaussian test channels and the calculations in (B.59) and (B.60) are all linear. Therefore, all pdfs $\phi_{U^{\text{TC}}_{i\to j}}$ and $\phi_{V^{\text{TC}}_{i\to j}}$ are Gaussian. Moreover, the pdfs $\phi_{U^{\text{TC}}_{i\to j}}$ and $\phi_{V^{\text{TC}}_{i\to j}}$ are tunable by changing the normalized distortions $d_{i\to j}$.

Remark 29. The random variable $U^{\text{TC}}_{i\to j}$ can be viewed intuitively as the estimate at the node v_i of the partial weighted sum $\mathbf{y}_{\mathcal{S}_{i\to j}}$ when test-channels can be physically established, while $V^{\text{TC}}_{i\to j}$ can be viewed as the description of $U^{\text{TC}}_{i\to j}$.

Before the computation starts, each node v_i generates $d(v_i)$ random codebooks $\mathcal{C}_{i\to j} = \{\mathbf{c}_{i\to j}(w) : w \in \{0, 1, \ldots 2^{NR_{i\to j}}\}\}, \forall j$ s.t. $v_j \in \mathcal{N}(v_i)$, where each codeword is generated i.i.d. according to distribution $\phi_{V^{\text{TC}}_{i\to j}}$. The rate is chosen such that

$$R_{i\to j} = I(U^{\text{TC}}_{i\to j}; V^{\text{TC}}_{i\to j}) + \delta_N = \frac{1}{2}\log\frac{\widehat{\sigma}^2_{i\to j}}{d_{i\to j}} + \delta_N, \tag{B.62}$$

where $U^{\text{TC}}_{i\to j}$ and $V^{\text{TC}}_{i\to j}$ are respectively the 'estimate' scalar random variable and the 'description' scalar random variable, and $\lim_{N\to\infty} \delta_N = 0$. Thus, the formula of the sum rate R in (4.61) can be proved by summing up the rates on all links in the network.

The codebook $\mathcal{C}_{i\to j}$ is revealed to the node v_j. During the computation, as shown in Fig 4.1, each node v_b, upon receiving description indexes $M_{1b}, M_{2b}, \dots M_{db}$ from the d neighbors $v_1, \dots v_d$ except the neighbor v_a, decodes these descriptions, computes the sum of them and the data vector generated at v_b

$$\widehat{\mathbf{s}}_{b\to a} = \sum_{k=1}^{d} \mathbf{c}_{k\to b}(M_{k\to b}) + w_b \mathbf{x}_b, \tag{B.63}$$

and re-encodes $\widehat{\mathbf{s}}_{b\to a}$ into a new description index $M_{b\to a} \in \{1, 2, \dots 2^{NR_{b\to a}}\}$ and sends the description index to the neighbor v_a with $NR_{b\to a}$ bits. We denote the reconstructed description by $\widehat{\mathbf{r}}_{b\to a} = \mathbf{c}_{b\to a}(M_{b\to a})$. The decoding and encoding at the node v_b are defined as follows.

- **Decoding:** In each codebook $\mathcal{C}_{k\to b}, \forall k$ s.t. $v_k \in \mathcal{N}(v_b)$, use the codeword $\mathbf{c}_{k\to b}(M_{k\to b})$ as the description $\widehat{\mathbf{r}}_{k\to b}$. If v_b has obtained all descriptions from all neighbors, it computes the sum of all descriptions and its own data as the estimate of $\mathbf{y}$:

$$\widehat{\mathbf{y}}_b = \sum_{v_k \in \mathcal{N}(v_b)} \widehat{\mathbf{r}}_{k\to b} + w_b \mathbf{x}_b. \tag{B.64}$$

- **Encoding:** For each neighbor $v_a \in \mathcal{N}(v_b)$, find the codeword $\mathbf{c}_{b\to a}(M_{b\to a}) \in \mathcal{C}_{b\to a} \setminus \{\mathbf{c}_{b\to a}(0)\}$ such that the two sequences $\widehat{\mathbf{s}}_{b\to a} = \sum_{k=1}^{d} \mathbf{c}_{k\to b}(M_{k\to b}) + w_b \mathbf{x}_b$ and $\widehat{\mathbf{r}}_{b\to a} = \mathbf{c}_{b\to a}(M_{b\to a})$ are jointly typical with respect to the distribution $\phi_{U^{\mathrm{TC}}_{b\to a}, V^{\mathrm{TC}}_{b\to a}}$. If there are more than one codewords that satisfy this condition, arbitrarily choose one of them. However, if $\widehat{\mathbf{s}}_{b\to a}$ is not typical with respect to the distribution $\phi_{U^{\mathrm{TC}}_{b\to a}}$, or if there is no codeword in $\mathcal{C}_{b\to a} \setminus \{\mathbf{c}_{b\to a}(0)\}$ that satisfies the joint typicality condition, send description index $M_{b\to a} = 0$.

Similar to the linear function computation case, the encoding step for network consensus may fail, because the estimate $\widehat{\mathbf{s}}_{b\to a} = \sum_{k=1}^{d} \mathbf{c}_{k\to b}(M_{k\to b}) + w_b \mathbf{x}_b$ may not be a typical sequence respect to pdf $\phi_{U^{\mathrm{TC}}_{b\to a}}$, or there may not exist codewords in $\mathcal{C}_{b\to a}$ that satisfy the typicality requirement. In this case, the description index $M_{b\to a} = 0$ is sent and this description is decoded to a predetermined random sequence $\mathbf{c}_{b\to a}(0)$ on the receiver side.

Lemma B.3.1 (Covering Lemma for Network Consensus). *Denote by $E_{i\to j} = 1$ the event that the encoding of the estimate $\widehat{\mathbf{s}}_{i\to j}$ at the node v_i is not successful. Then*

$$\lim_{N\to\infty} \sup_{(i,j)\in\mathcal{E}} \Pr(E_{i\to j} = 1) = 0, \tag{B.65}$$

where $\mathcal{E}$ denotes all links in the tree network $\mathcal{G} = (\mathcal{V}.\mathcal{E})$ ((i,j) and (j,i) are viewed as two links in the undirected graph $\mathcal{G}$), and the probability is taken over random data sampling and random codebook generation.

Proof. The proof of this lemma is almost the same as the proof for linear function computing case (see Appendix C-C in [279]). This is because the distributed computation algorithm used in this section can be viewed as a group of $n = |\mathcal{V}|$ linear function computations in n different directed trees $\vec{\mathcal{T}}_k, 1 \leq k \leq n$ towards n different roots (see definition of $\vec{\mathcal{T}}_k$ below equation (4.63)). Therefore, we can use the conditional typicality lemma and mathematical induction on each directed tree to obtain the conclusion. $\square$

Remark 30. The proofs for network consensus are also based on the induction on the tree (see Remark 5), except that we may often want to prove that some property P holds at all links $v_b \to v_a$ in the tree network. Firstly, we prove that P holds for all links $v_l \to v_n(l)$, where v_l is a leaf and $v_{n(l)}$ is the only neighbor of v_l. Secondly, we prove that, for an arbitrary node v_b with $d+1$ neighbors, denoted by $v_1, v_2, \ldots v_d$ and a special neighbor v_a, if P holds for all links $v_1 \to v_b, v_2 \to v_b, \ldots v_d \to v_b$, then the property holds for the link $v_b \to v_a$. It is obvious that these two arguments lead to the conclusion that P holds for all links in the tree network.

Lemma B.3.1 states that the estimate $\widehat{\mathbf{s}}_{b\to a}$ and the description $\widehat{\mathbf{r}}_{b\to a}$ are jointly typical with high probability for all links $v_b \to v_a$ in the tree network. The following Lemma B.3.2 and Lemma B.3.3 are counterparts of Lemma 4.4.3 and Lemma 4.4.4 in the linear function computation problem.

Lemma B.3.2. *For an arbitrary link $v_b \to v_a$, the description $\widehat{\mathbf{r}}_{b\to a} = \mathbf{c}_{b\to a}(M_{ba})$ and the estimate $\widehat{\mathbf{s}}_{b\to a}$ satisfy*

$$\left| \mathbb{E}\left[\frac{1}{N}\|\widehat{\mathbf{s}}_{b\to a}\|_2^2\right] - \widehat{\sigma}^2_{b\to a} \right| < \varepsilon_N, \tag{B.66}$$

$$\left| \mathbb{E}\left[\frac{1}{N}\|\widehat{\mathbf{r}}_{b\to a}\|_2^2\right] - (\widehat{\sigma}^2_{b\to a} - d_{b\to a}) \right| < \varepsilon_N, \tag{B.67}$$

$$\left| \mathbb{E}\left[\frac{1}{N}\|\widehat{\mathbf{r}}_{b\to a} - \widehat{\mathbf{s}}_{b\to a}\|_2^2\right] - d_{b\to a} \right| < \varepsilon_N, \tag{B.68}$$

where $\lim_{N\to\infty} \varepsilon_N = 0$.

Proof. Similar with the proof of Lemma B.3.1, the proof of this lemma can be derived similarly as the proof for the linear function computation case (see Appendix C-D in [279]), because the proof for the linear function computation case is mathematical induction in the tree network, while the network consensus computation scheme in this section can be viewed as a group of linear function computations on n different directed trees. $\square$

Lemma B.3.3. *For an arbitrary link $v_b \to v_a$, the description $\widehat{\mathbf{r}}_{b\to a} = \mathbf{c}_{b\to a}(M_{ba})$ and the estimate $\widehat{\mathbf{s}}_{b\to a}$ satisfy*

$$h(\widehat{\mathbf{s}}_{b\to a}) > \frac{N}{2}\log_2 2\pi e \widehat{\sigma}^2_{b\to a} - N\beta_N, \tag{B.69}$$

$$h(\widehat{\mathbf{r}}_{b\to a}) > \frac{N}{2}\log_2 2\pi e (\widehat{\sigma}^2_{b\to a} - d_{b\to a}) - N\beta_N, \tag{B.70}$$

where $\lim_{N\to\infty} \beta_N = 0$.

Proof. One can use the same argument as the one used in the proof of Lemma B.3.2. $\square$

The following lemma characterizes the relationship between the Gaussian-code-based distortion $d_{i\to j}$ (normalized distortion) and the MMSE-based distortion $D^{\text{Tx}}_{i\to j}$ for the Gaussian code.

Lemma B.3.4. *For an arbitrary link $v_i \to v_j$*

$$\sqrt{d_{i\to j} - \varepsilon_N} - \eta_N \leq \sqrt{D^{Rx}_{i\to j} - D^{Tx}_{i\to j}} \leq \sqrt{d_{i\to j} + \varepsilon_N} + \eta_N, \tag{B.71}$$

where $\lim\limits_{N\to\infty} \eta_N = 0$ *and* ε_N *is the same as in* (B.68).

Proof. The proof of this lemma essentially follows the same procedures with the ones in the proof for linear function computation in the Appendix B.2.3. We only provide the sketch of the proof. First, define

$$\widehat{\mathbf{s}}^*_{i\to j} = \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S}_{i\to j},i}, \tag{B.72}$$

$$\widehat{\mathbf{r}}^*_{i\to j} = \widehat{\mathbf{y}}^{\text{mmse}}_{\mathcal{S}_{i\to j},j}. \tag{B.73}$$

Therefore, $\widehat{\mathbf{s}}^*_{i\to j}$ is the MMSE estimate of the partial weighted sum $\mathbf{y}_{\mathcal{S}_{i\to j}}$ at node v_i, while $\widehat{\mathbf{r}}^*_{i\to j}$ is the MMSE estimate of the same weighted sum at node v_j. Define

$$\Delta^{\text{Tx}}_{i\to j} = \mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{s}}_{i\to j} - \widehat{\mathbf{s}}^*_{i\to j}\right\|_2^2\right], \tag{B.74}$$

$$\Delta^{\text{Rx}}_{i\to j} = \mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}_{i\to j} - \widehat{\mathbf{r}}^*_{i\to j}\right\|_2^2\right]. \tag{B.75}$$

We will prove that $\Delta^{\text{Tx}}_{i\to j} \to 0$ and $\Delta^{\text{Rx}}_{i\to j} \to 0$ when $N \to \infty$.

Using the same derivations with equation (B.35) to (B.37), we get

$$\Delta^{\text{Tx}}_{b\to a} = \sum_{k=1}^{d} \Delta^{\text{Rx}}_{k\to b}, \tag{B.76}$$

for an arbitrary link $v_b \to v_a$ and the neighborhood structure $\mathcal{N}(v_b) = \{v_1, \ldots v_d\} \cup \{v_a\}$ (see Figure 4.1). Using the same derivations with equation (B.39) to (B.45), we get

$$\sqrt{\Delta^{\text{Rx}}_{b\to a}} \leq \sqrt{\theta_N} + \sqrt{\Delta^{\text{Tx}}_{b\to a}}, \tag{B.77}$$

for an arbitrary link $v_b \to v_a$ and the constant $\lim_{N\to\infty}\theta_N = 0$. Using induction on n different directed tree networks, we get

$$\sup_{(i,j)\in\mathcal{E}} \sqrt{\Delta^{\text{Tx}}_{i\to j}} \leq \sup_{(i,j)\in\mathcal{E}} \sqrt{\Delta^{\text{Rx}}_{i\to j}} \leq n\sqrt{\theta_N}. \tag{B.78}$$

Using the triangle inequality, we get

$$\sqrt{\mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}^*_{i\to j} - \widehat{\mathbf{s}}^*_{i\to j}\right\|_2^2\right]} \leq \sqrt{d_{i\to j} + \varepsilon_N} + 2n\sqrt{\theta_N}, \tag{B.79}$$

and

$$\sqrt{\mathbb{E}\left[\frac{1}{N}\left\|\widehat{\mathbf{r}}^*_{i\to j}-\widehat{\mathbf{s}}^*_{i\to j}\right\|_2^2\right]} \geq \sqrt{d_{i\to j}-\varepsilon_N}-2n\sqrt{\theta_N}, \tag{B.80}$$

which conclude the proof. □

Using the same procedures from (4.50) to (4.55), one can prove that the overall distortion at one node, averaged over the random code ensemble satisfies

$$D_i^{\text{Total}} \leq \sum_{(i,j)\in\overrightarrow{\mathcal{T}}_k} d_{i\to j}+\epsilon_N. \tag{B.81}$$

Summing the above equations over all directed trees in the network, we have that (4.68) holds for the overall distortion averaged over the random code ensemble. Therefore, we can at least find one code for which (4.68) holds.

Appendix C

Theoretical proofs for Chapter 5

C.1 Proof of Theorem 5.3.1

Recall that we call the sub-blocks on the same row block (in Figure 5.1) of the P different blocks of data a group, and we use $\mathbf{X}_{k,j}^{\text{coded}}$ to represent the coded sub-block of data that is at the k-th machine and belongs to the j-th group. Let $\mathbf{X}^j$ be the collections of original data that belongs to the j-th group. For example, in Figure 5.1(b), $\mathbf{X}^6$ represents the collection of original data $\left[\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\right]^\top$. Then, from the cyclic way of using data, we can see that when each machine uses L sub-blocks, the overall number of used sub-blocks in each group is L. From Lemma 2.2.4, the results $\mathbf{u}_{t,k,j} = \mathbf{X}_{k,j}^{\text{coded}}\mathbf{w}_t$ for all the workers that use the coded data in the j-th group can be collected together to decode $\mathbf{X}^j\mathbf{w}_t$. The other claim can be seen from the figure, i.e., the area of the used data is always equal to the area of the original data, and the area of the used data is the same across all remaining machines. We prove this claim as follows. The size of the data in each of the L subsets is N/L. Thus, the used data at each machine is the same number $\frac{N}{L}\frac{L}{P-M} = N/(P-M)$. There are $P-M$ machines left, so the overall size of the used data is $\frac{N}{P-M}\cdot(P-M) = N$, which is the same as the original data.

C.2 Proof of Theorem 5.3.2

Now, we prove statement (a). Suppose an arbitrary set of $P-L$ machines fail. Since the algorithm is tolerant to any $P-L$ failures, the master node can still recover exactly all the results $\mathbf{X}\mathbf{w}$, no matter what $\mathbf{w}$ is. Therefore, we can choose $\mathbf{w}$ to be the elements of natural basis $\mathbf{w} = \mathbf{e}_i, i = 1, 2, \ldots, d$, and collect all the results $[\mathbf{X}\mathbf{e}_1, \mathbf{X}\mathbf{e}_2, \ldots, \mathbf{X}\mathbf{e}_d] = \mathbf{X}\mathbf{I}_d = \mathbf{X}$. Since data processing can only reduce entropy (from the data processing inequality), the entropy of the overall data stored at the remaining L machines is no less than the entropy of $\mathbf{X}$ which is NdH. This holds for any combination of L machines, i.e.,

$$H(\mathbf{X}_{i_1}, \mathbf{X}_{i_2}, \ldots, \mathbf{X}_{i_L}) \geq NdH, \forall 1 \leq i_1 < i_2 < \ldots < i_L \leq P. \tag{C.1}$$

Adding up the above equation for all combinations of L out of P machines, and by plugging in $\sum_{j=1}^{L} H(\mathbf{X}_{i_j}) \geq H(\mathbf{X}_{i_1}, \mathbf{X}_{i_2}, \ldots, \mathbf{X}_{i_L})$, we have $\sum_{k=1}^{P} H_k \geq \frac{NPdH}{L}$.

Then, we prove statement (b). Suppose (b) is not true. Then, it means that there exists a way to encode and store the encoded data in the memory of the machines, such that for any arbitrary vector $\mathbf{w}$, computing $\mathbf{Xw}$ only requires reading data of entropy strictly less than NdH. Since the overall number of stored data is finite, we can assume that the overall number of stored numbers is p. For an arbitrary subset $\mathcal{S}$ of the stored numbers such that $H(\mathcal{S}) < NdH$, we denote by $\mathcal{P}_{\mathcal{S}}$ the set of $\mathbf{w}$ such that $\mathbf{Xw}$ is able to be computed using only the numbers in $\mathcal{S}$. Then, we can see that the $\mathcal{P}_{\mathcal{S}}$ for an arbitrary subset $\mathcal{S}$ is a linear subspace of $\mathbb{R}^d$. This is because if $\mathbf{Xw}_1$ and $\mathbf{Xw}_2$ are both able to be computed using the numbers in $\mathcal{S}$, then, $a\mathbf{Xw}_1 + b\mathbf{Xw}_2 = \mathbf{X}(a\mathbf{w}_1 + b\mathbf{w}_2)$ is also able to be computed. Now, we prove that $\mathcal{P}_{\mathcal{S}}$ cannot be the entire $\mathbb{R}^d$ that $\mathbf{w}$ can take value from. This is because if $\mathcal{P}_{\mathcal{S}}$ is equal to $\mathbb{R}^d$, then $\mathbf{Xe}_1, \mathbf{Xe}_2, \ldots, \mathbf{Xe}_d$ are all able to be computed using the numbers in $\mathcal{S}$ ($\mathbf{e}_1, \ldots \mathbf{e}_d$ are the standard basis), which means $\mathbf{X}$ itself is able to be computed using $\mathcal{S}$. This is a clear contradiction to the data-processing inequality because $H(\mathbf{X}) = NdH$, while $H(\mathcal{S}) < NdH$. Thus, $\mathcal{P}_{\mathcal{S}}$ can at most be a linear space of dimension $d-1$ in $\mathbb{R}^d$. This means the union of $\mathcal{P}_{\mathcal{S}}$ for all $\mathcal{S}$ is a finite collection of linear spaces of dimension $d-1$ in $\mathbb{R}^d$, which cannot cover the entire $\mathbb{R}^d$. Thus, there must exist $\mathbf{w}$ in $\mathbb{R}^d$ such that $\mathbf{Xw}$ is not able to be computed by using data of entropy strictly less than NdH.

Appendix D

Theoretical proofs for Chapter 6

D.1 Proof of Theorem 6.3.1

We first introduce some notation and preliminary properties that we will use in this proof. The proof can be found in [277]. Denote by vec$(\mathbf{A})$ the vector that is composed of the concatenation of all columns in a matrix $\mathbf{A}$. For example, the vectorization of $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ is the column vector vec$(\mathbf{A}) = [1, 4, 2, 5, 3, 6]^\top$. We will also use the Kronecker product defined as

$$\mathbf{A}_{m\times n} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \dots & a_{1n}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & a_{m2}\mathbf{B} & \dots & a_{mn}\mathbf{B}. \end{bmatrix} \tag{D.1}$$

We now state some properties of the vectorization and Kronecker product.
Lemma D.1.1. *Property 1: if* $\mathbf{A} = \mathbf{B}\mathbf{C}$*, then*

$$vec(\mathbf{A}) = (\mathbf{C} \otimes \mathbf{I}_N) vec(\mathbf{B}). \tag{D.2}$$

Property 2: vectorization does not change the Frobenius norm, i.e.,

$$\|\mathbf{A}\| = \|vec(\mathbf{A})\|. \tag{D.3}$$

Property 3: The following mixed-product property holds

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{A} \cdot \mathbf{C}) \otimes (\mathbf{B} \cdot \mathbf{D}), \tag{D.4}$$

if one can form the matrices $\mathbf{AC}$ *and* $\mathbf{BD}$.
Property 4: If $\mathbf{A}$ *and* $\mathbf{B}$ *are both positive semi-definite,* $\mathbf{A} \otimes \mathbf{B}$ *is also positive semi-definite.*
Property 5: Suppose $\mathbf{C}$ *is positive semi-definite and* $\mathbf{A} \preceq \mathbf{B}$*. Then,*

$$\mathbf{A} \otimes \mathbf{C} \preceq \mathbf{B} \otimes \mathbf{C}. \tag{D.5}$$

Property 6: (commutative property) Suppose $\mathbf{A}_{m\times n}$ *and* $\mathbf{B}_{p\times q}$ *are two matrices. Then,*

$$(\mathbf{A}_{m\times n}\otimes\mathbf{I}_p)\cdot(\mathbf{I}_n\otimes\mathbf{B}_{p\times q})=(\mathbf{I}_m\otimes\mathbf{B}_{p\times q})\cdot(\mathbf{A}_{m\times n}\otimes\mathbf{I}_q). \tag{D.6}$$

Property 7: Suppose $\mathbf{A}$ *is an* $nN\times nN$ *matrix that can be written as*

$$\mathbf{A}_{nN\times nN}=\begin{bmatrix}\mathbf{A}_{11} & \mathbf{A}_{12} & \dots & \mathbf{A}_{1n}\\ \mathbf{A}_{21} & \mathbf{A}_{22} & \dots & \mathbf{A}_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ \mathbf{A}_{n1} & \mathbf{A}_{n2} & \dots & \mathbf{A}_{nn}\end{bmatrix}, \tag{D.7}$$

where each $\mathbf{A}_{ij}$ *is a square matrix of size* $N\times N$*. Then, for an arbitrary matrix* $\mathbf{L}$ *of size* $k\times n$*,*

$$\mathit{trace}\left[(\mathbf{L}\otimes\mathbf{I}_N)\cdot\mathbf{A}\cdot(\mathbf{L}\otimes\mathbf{I}_N)^\top\right]=\mathit{trace}\left[\mathbf{L}\cdot\begin{bmatrix}\mathit{trace}[\mathbf{A}_{11}] & \dots & \mathit{trace}[\mathbf{A}_{1n}]\\ \vdots & \ddots & \vdots\\ \mathit{trace}[\mathbf{A}_{n1}] & \dots & \mathit{trace}[\mathbf{A}_{nn}]\end{bmatrix}\cdot\mathbf{L}^\top\right]. \tag{D.8}$$

Computing the explicit form of the error matrix E

From (6.15), we have encoded the input $\mathbf{r}_i$ to the linear inverse problem in the following way:

$$[\mathbf{s}_1,\mathbf{s}_2,\dots,\mathbf{s}_n]=[\mathbf{r}_1,\mathbf{r}_2,\dots,\mathbf{r}_k]\cdot\mathbf{G}. \tag{D.9}$$

Since $\mathbf{x}_i^*$ is the solution to the linear inverse problem, we have

$$\mathbf{x}_i^*=\mathbf{C}_{\text{inv}}\mathbf{r}_i, \tag{D.10}$$

where $\mathbf{C}_{\text{inv}}$ is either the direct inverse in (6.2) for square linear inverse problems or the least-square solution in (6.4) for non-square inverse problems. Define $\mathbf{y}_i^*$ as the solution of the inverse problem with the encoded input $\mathbf{s}_i$. Then, we also have

$$\mathbf{y}_i^*=\mathbf{C}_{\text{inv}}\mathbf{s}_i. \tag{D.11}$$

Left-multiplying $\mathbf{C}_{\text{inv}}$ on both LHS and RHS of (D.9) and plugging in (D.10) and (D.11), we have

$$[\mathbf{y}_1^*,\mathbf{y}_2^*,\dots,\mathbf{y}_n^*]=[\mathbf{x}_1^*,\mathbf{x}_2^*,\dots,\mathbf{x}_k^*]\cdot\mathbf{G}=\mathbf{X}^*\cdot\mathbf{G}. \tag{D.12}$$

Define $\boldsymbol{\epsilon}_i^{(l)}=\mathbf{y}_i^{(l_i)}-\mathbf{y}_i^*$, which is the remaining error at the i-th worker after l_i iterations. From the explicit form (6.10) of the remaining error of the executed iterative algorithm, we have

$$\mathbf{y}_i^{(l_i)}=\mathbf{y}_i^*+\boldsymbol{\epsilon}_i^{(l_i)}=\mathbf{y}_i^*+\mathbf{B}^{l_i}\boldsymbol{\epsilon}_i^{(0)}. \tag{D.13}$$

Therefore,from the definition of $\mathbf{Y}^{(T_{\text{dl}})}$ (see (6.17)) and equation (D.12) and (D.13),

$$\begin{aligned}\mathbf{Y}^{(T_{\text{dl}})}=&[\mathbf{y}_1^{(l_1)},\mathbf{y}_2^{(l_2)},\dots,\mathbf{y}_n^{(l_n)}]\\ =&[\mathbf{y}_1^*,\mathbf{y}_2^*,\dots,\mathbf{y}_n^*]+[\boldsymbol{\epsilon}_1^{(l_1)},\boldsymbol{\epsilon}_2^{(l_2)},\dots,\boldsymbol{\epsilon}_n^{(l_n)}]\\ =&\mathbf{X}^*\cdot\mathbf{G}+[\mathbf{B}^{l_1}\boldsymbol{\epsilon}_1^{(0)},\dots,\mathbf{B}^{l_n}\boldsymbol{\epsilon}_n^{(0)}].\end{aligned} \tag{D.14}$$

Plugging in (6.18), we get the explicit form of $\mathbf{E} = \hat{\mathbf{X}}^\top - \mathbf{X}^*$:

$$\begin{aligned}\hat{\mathbf{X}}^\top &= (\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\mathbf{G}\mathbf{\Lambda}^{-1}(\mathbf{Y}^{(T_{\text{dl}})})^\top \\ &= (\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\mathbf{G}\mathbf{\Lambda}^{-1}\left[\mathbf{G}^\top(\mathbf{X}^*)^\top + [\mathbf{B}^{l_1}\boldsymbol{\epsilon}_1^{(0)}, \dots, \mathbf{B}^{l_n}\boldsymbol{\epsilon}_n^{(0)}]^\top\right] \\ &= (\mathbf{X}^*)^\top + (\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\mathbf{G}\mathbf{\Lambda}^{-1}\left[\mathbf{B}^{l_1}\boldsymbol{\epsilon}_1^{(0)}, \dots, \mathbf{B}^{l_n}\boldsymbol{\epsilon}_n^{(0)}\right]^\top.\end{aligned} \tag{D.15}$$

From (6.16), (D.12) and the definition $\boldsymbol{\epsilon}_i^{(0)} = \mathbf{y}_i^{(0)} - \mathbf{y}_i^*$ and $\mathbf{e}_i^{(l)} = \mathbf{x}_i^{(0)} - \mathbf{x}_i^*$, we have

$$[\boldsymbol{\epsilon}_1^{(0)}, \boldsymbol{\epsilon}_2^{(0)}, \dots, \boldsymbol{\epsilon}_n^{(0)}] = [\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \dots, \mathbf{e}_k^{(0)}] \cdot \mathbf{G}. \tag{D.16}$$

Vectorization of the error matrix E

From property 2 of Lemma D.1.1, vectorization does not change the Frobenius norm, so we have

$$\mathbb{E}[\|\mathbf{E}\|^2 \,|\mathbf{l}] = \mathbb{E}[\|\text{vec}(\mathbf{E})\|^2 \,|\mathbf{l}] = \mathbb{E}\left[\text{trace}\left(\text{vec}(\mathbf{E})\text{vec}(\mathbf{E})^\top\right)|\mathbf{l}\right]. \tag{D.17}$$

Therefore, to prove the conclusion of this theorem, i.e., $\mathbb{E}[\|\mathbf{E}\|^2 \,|\mathbf{l}] \le \sigma_{\max}(\mathbf{G}^\top\mathbf{G})\text{trace}\left[(\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\right]$, we only need to show

$$\mathbb{E}\left[\text{trace}\left(\text{vec}(\mathbf{E})\text{vec}(\mathbf{E})^\top\right)|\mathbf{l}\right] \le \sigma_{\max}(\mathbf{G}^\top\mathbf{G})\text{trace}\left[(\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\right]. \tag{D.18}$$

Express the mean-squared error using the vectorization form

Now we prove (D.18). From (D.15), we have

$$\mathbf{E}^\top = (\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\mathbf{G}\mathbf{\Lambda}^{-1}[\mathbf{B}^{l_1}\boldsymbol{\epsilon}_1^{(0)}, \dots, \mathbf{B}^{l_n}\boldsymbol{\epsilon}_n^{(0)}]^\top, \tag{D.19}$$

which is the same as

$$\mathbf{E} = [\mathbf{B}^{l_1}\boldsymbol{\epsilon}_1^{(0)}, \dots, \mathbf{B}^{l_n}\boldsymbol{\epsilon}_n^{(0)}] \cdot [(\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\mathbf{G}\mathbf{\Lambda}^{-1}]^\top. \tag{D.20}$$

From property 1 of Lemma D.1.1, (D.20) means

$$\begin{aligned}\text{vec}(\mathbf{E}) &= \left[(\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\mathbf{G}\mathbf{\Lambda}^{-1} \otimes \mathbf{I}_N\right] \cdot \text{vec}([\mathbf{B}^{l_1}\boldsymbol{\epsilon}_1^{(0)}, \dots, \mathbf{B}^{l_n}\boldsymbol{\epsilon}_n^{(0)}]) \\ &= \left[(\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\mathbf{G}\mathbf{\Lambda}^{-1} \otimes \mathbf{I}_N\right] \cdot \text{diag}[\mathbf{B}^{l_1}, \dots, \mathbf{B}^{l_n}] \cdot \text{vec}([\boldsymbol{\epsilon}_1^{(0)}, \dots, \boldsymbol{\epsilon}_n^{(0)}]).\end{aligned} \tag{D.21}$$

Define

$$\mathbf{L} = (\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\mathbf{G}\mathbf{\Lambda}^{-1}, \tag{D.22}$$

$$\mathbf{D} = \text{diag}[\mathbf{B}^{l_1}, \dots, \mathbf{B}^{l_n}], \tag{D.23}$$

and

$$\mathbf{E}_0 = \text{vec}([\boldsymbol{\epsilon}_1^{(0)}, \dots, \boldsymbol{\epsilon}_n^{(0)}]). \tag{D.24}$$

Then,

$$\text{vec}(\mathbf{E}) = (\mathbf{L} \otimes \mathbf{I}_N) \cdot \mathbf{D} \cdot \mathbf{E}_0. \tag{D.25}$$

Therefore,

$$\mathbb{E}\left[\text{trace}\left(\text{vec}(\mathbf{E})\text{vec}(\mathbf{E})^\top\right)|\mathbf{l}\right] = \text{trace}\left((\mathbf{L} \otimes \mathbf{I}_N \cdot \mathbf{D})\mathbb{E}[\mathbf{E}_0\mathbf{E}_0^\top|\mathbf{l}](\mathbf{L} \otimes \mathbf{I}_N \cdot \mathbf{D})^\top\right). \tag{D.26}$$

Bounding the term $\mathbb{E}[\mathbf{E}_0\mathbf{E}_0^\top|\mathbf{l}]$ using the maximum eigenvalue $\sigma_{\textbf{max}}(\mathbf{G}^\top\mathbf{G})$

Note that $\mathbf{E}_0 = \text{vec}([\boldsymbol{\epsilon}_1^{(0)}, \ldots, \boldsymbol{\epsilon}_n^{(0)}])$. From (D.16), we have

$$[\boldsymbol{\epsilon}_1^{(0)}, \boldsymbol{\epsilon}_2^{(0)}, \ldots, \boldsymbol{\epsilon}_n^{(0)}] = [\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \ldots, \mathbf{e}_k^{(0)}] \cdot \mathbf{G}. \tag{D.27}$$

Therefore, using property 1 of Lemma D.1.1, we have

$$\mathbf{E}_0 = (\mathbf{G}^\top \otimes \mathbf{I}_N) \cdot \text{vec}([\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \ldots, \mathbf{e}_k^{(0)}]). \tag{D.28}$$

From Assumption 5, the covariance of $\mathbf{e}_i^{(0)}$ is

$$\mathbb{E}[\mathbf{e}_i^{(0)}(\mathbf{e}_i^{(0)})^\top|\mathbf{l}] = \mathbf{C}_E, i = 1, \ldots, k. \tag{D.29}$$

Therefore, from (D.28), we have

$$\begin{aligned}
\mathbb{E}[\mathbf{E}_0\mathbf{E}_0^\top|\mathbf{l}] =&(\mathbf{G}^\top \otimes \mathbf{I}_N) \cdot \mathbb{E}[\text{vec}([\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \ldots, \mathbf{e}_k^{(0)}])\cdot \\
&\quad \text{vec}([\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \ldots, \mathbf{e}_k^{(0)}])^\top|\mathbf{l}] \cdot (\mathbf{G}^\top \otimes \mathbf{I}_N)^\top \\
&\overset{(a)}{=}(\mathbf{G}^\top \otimes \mathbf{I}_N) \cdot (\mathbf{I}_k \otimes \mathbf{C}_E) \cdot (\mathbf{G}^\top \otimes \mathbf{I}_N)^\top \\
&=(\mathbf{G}^\top \otimes \mathbf{I}_N) \cdot (\mathbf{I}_k \otimes \mathbf{C}_E) \cdot (\mathbf{G} \otimes \mathbf{I}_N) \\
&\overset{(b)}{=}(\mathbf{G}^\top \cdot \mathbf{I}_k \cdot \mathbf{G}) \otimes (\mathbf{I}_N \cdot \mathbf{C}_E \cdot \mathbf{I}_N) \\
&=\mathbf{G}^\top\mathbf{G} \otimes \mathbf{C}_E \\
&\overset{(c)}{\preceq}\sigma_{\max}(\mathbf{G}^\top\mathbf{G})\mathbf{I}_n \otimes \mathbf{C}_E,
\end{aligned} \tag{D.30}$$

where (a) is from (D.29), (b) and (c) follow respectively from property 3 and property 5 of Lemma D.1.1.

If $\mathbf{G}$ has orthonormal rows, the eigenvalues of $\mathbf{G}^\top\mathbf{G}$ (which is an $n \times n$ matrix) are all in $(0, 1]$. This is why we can remove the term $\sigma_{\max}(\mathbf{G}^\top\mathbf{G})$ in (6.23) when $\mathbf{G}$ has orthonormal rows. In what follows, we assume $\mathbf{G}$ has orthonormal rows, and the result when $\mathbf{G}$ does not have orthonormal rows follows naturally.

Assuming $\mathbf{G}$ has orthonormal rows, we have

$$\mathbb{E}[\mathbf{E}_0\mathbf{E}_0^\top|\mathbf{l}] \preceq \mathbf{I}_n \otimes \mathbf{C}_E. \tag{D.31}$$

Plugging (D.31) into (D.26), we have

$$\begin{aligned}
&\mathbb{E}\left[\text{trace}\left(\text{vec}(\mathbf{E})\text{vec}(\mathbf{E})^\top\right)|\mathbf{l}\right] \\
\leq&\text{trace}\left((\mathbf{L} \otimes \mathbf{I}_N) \cdot \mathbf{D}(\mathbf{I}_n \otimes \mathbf{C}_E)\mathbf{D}^\top(\mathbf{L} \otimes \mathbf{I}_N)^\top\right),
\end{aligned} \tag{D.32}$$

where $\mathbf{D} = \text{diag}[\mathbf{B}^{l_1}, \ldots, \mathbf{B}^{l_n}]$. Therefore,

$$\begin{aligned}
&\mathbf{D}(\mathbf{I}_n \otimes \mathbf{C}_E)\mathbf{D}^\top \\
=&\text{diag}[\mathbf{B}^{l_1}\mathbf{C}_E(\mathbf{B}^\top)^{l_1}, \ldots, \mathbf{B}^{l_n}\mathbf{C}_E(\mathbf{B}^\top)^{l_n}].
\end{aligned} \tag{D.33}$$

From the definition of $\mathbf{C}(l_i)$ in (6.20),

$$\mathbf{D}(\mathbf{I}_n \otimes \mathbf{C}_E)\mathbf{D}^\top = \text{diag}[\mathbf{C}(l_1), \ldots, \mathbf{C}(l_n)]. \tag{D.34}$$

Reducing the dimensionality of $\mathbf{D}(\mathbf{I}_n \otimes \mathbf{C}_E)\mathbf{D}^\top$ in the trace expression using property 7 in Lemma D.1.1

From Property 7 in Lemma D.1.1, we can simplify (D.35):

$$\begin{aligned}
&\mathbb{E}\left[\text{trace}\left(\text{vec}(\mathbf{E})\text{vec}(\mathbf{E})^\top\right)|\mathbf{l}\right]\\
\leq&\text{trace}\left((\mathbf{L}\otimes\mathbf{I}_N)\cdot\mathbf{D}(\mathbf{I}_n\otimes\mathbf{C}_E)\mathbf{D}^\top(\mathbf{L}\otimes\mathbf{I}_N)^\top\right)\\
\overset{(a)}{=}&\text{trace}\left((\mathbf{L}\otimes\mathbf{I}_N)\cdot\text{diag}[\mathbf{C}(l_1),\ldots,\mathbf{C}(l_n)](\mathbf{L}\otimes\mathbf{I}_N)^\top\right)\\
\overset{(b)}{=}&\text{trace}\left[\mathbf{L}\cdot\text{diag}[\text{trace}(\mathbf{C}(l_1)),\ldots,\text{trace}(\mathbf{C}(l_1))]\mathbf{L}^\top\right]\\
\overset{(c)}{=}&\text{trace}\left(\mathbf{L}\mathbf{\Lambda}\mathbf{L}^\top\right),
\end{aligned} \tag{D.35}$$

where (a) is from (D.34), (b) is from Property 7 and (c) is from the definition of $\mathbf{\Lambda}$ in (6.19). Equation (D.35) can be further simplified to

$$\begin{aligned}
&\mathbb{E}\left[\text{trace}\left(\text{vec}(\mathbf{E})\text{vec}(\mathbf{E})^\top\right)|\mathbf{l}\right]\leq\text{trace}\left(\mathbf{L}\mathbf{\Lambda}\mathbf{L}^\top\right)\\
\overset{(a)}{=}&\text{trace}\left((\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{\Lambda}((\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\mathbf{G}\mathbf{\Lambda}^{-1})^\top\right)\\
=&\text{trace}((\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}),
\end{aligned} \tag{D.36}$$

where (a) is from the definition of the decoding matrix $\mathbf{L} = (\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1}\mathbf{G}\mathbf{\Lambda}^{-1}$. Thus, we have completed the proof of Theorem 6.3.1 for the case when $\mathbf{G}$ has orthonormal rows. As we argued earlier, the proof when $\mathbf{G}$ does not have orthonormal rows follows immediately (see the text after (D.30)).

D.2 Proof of Corollary 6.3.2

First, note that

$$\mathbf{G} = [\mathbf{I}_k, \mathbf{0}_{k,n-k}]\,\mathbf{F}. \tag{D.37}$$

Therefore,

$$\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top = [\mathbf{I}_k, \mathbf{0}_{k,n-k}]\,\mathbf{F}\mathbf{\Lambda}^{-1}\mathbf{F}^\top\,[\mathbf{I}_k, \mathbf{0}_{k,n-k}]^\top \overset{(a)}{=} [\mathbf{I}_k, \mathbf{0}_{k,n-k}]\,(\mathbf{F}\mathbf{\Lambda}\mathbf{F}^\top)^{-1}\,[\mathbf{I}_k, \mathbf{0}_{k,n-k}]^\top, \tag{D.38}$$

where (a) is from $\mathbf{F}^\top\mathbf{F} = \mathbf{I}_n$. Now take the inverse of both sides of (6.24), we have

$$(\mathbf{F}\mathbf{\Lambda}\mathbf{F}^\top)^{-1} = \begin{bmatrix}(\mathbf{J}_1 - \mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top)^{-1} & * \\ * & *\end{bmatrix}_{n\times n}, \tag{D.39}$$

where $*$ is used as a substitute for matrices that are unimportant for our argument. Thus, comparing (D.38) and (D.39),

$$\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top = (\mathbf{J}_1 - \mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top)^{-1}, \tag{D.40}$$

which means

$$(\mathbf{G}\mathbf{\Lambda}^{-1}\mathbf{G}^\top)^{-1} = \mathbf{J}_1 - \mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top. \tag{D.41}$$

From (6.23) and (D.41), the theorem follows.

D.3 Proof of Theorem 6.3.4

In this section, we compute the residual error of the uncoded linear inverse algorithm. From (6.9), we have

$$\mathbf{e}_i^{(l+1)} = \mathbf{B}\mathbf{e}_i^{(l)}. \tag{D.42}$$

Therefore, in the uncoded scheme, the overall error is

$$\begin{aligned}
\mathbb{E}\left[\|\mathbf{E}_{\text{uncoded}}\|^2 | \mathbf{l}\right] &= \mathbb{E}\left[\left\|[\mathbf{e}_1^{(l_1)}, \mathbf{e}_2^{(l_2)} \ldots, \mathbf{e}_k^{(l_k)}]\right\|^2 | \mathbf{l}\right] = \sum_{i=1}^{k} \mathbb{E}\left[\left\|[\mathbf{e}_i^{(l_i)}]\right\|^2 | \mathbf{l}\right] \\
&= \sum_{i=1}^{k} \text{trace}\left(\mathbb{E}\left[\mathbf{e}_i^{(l_i)}(\mathbf{e}_i^{(l_i)})^\top | \mathbf{l}\right]\right) \overset{(a)}{=} \sum_{i=1}^{k} \text{trace}\left(\mathbb{E}\left[\mathbf{B}^{l_i}\mathbf{e}_i^{(0)}(\mathbf{B}^{l_i}\mathbf{e}_i^{(0)})^\top | \mathbf{l}\right]\right) \\
&= \sum_{i=1}^{k} \text{trace}\left(\mathbf{B}^{l_i}\mathbb{E}\left[\mathbf{e}_i^{(0)}(\mathbf{e}_i^{(0)})^\top | \mathbf{l}\right](\mathbf{B}^{l_i})^\top\right) = \sum_{i=1}^{k} \text{trace}\left(\mathbf{B}^{l_i} \cdot \mathbf{C}_E \cdot (\mathbf{B}^{l_i})^\top\right) \\
&= \sum_{i=1}^{k} \text{trace}\left(\mathbf{B}^{l_i}\mathbf{C}_E(\mathbf{B}^{l_i})^\top\right) \overset{(b)}{=} \sum_{i=1}^{k} \text{trace}\left(\mathbf{C}(l_i)\right),
\end{aligned} \tag{D.43}$$

where (a) is from (D.42) and (b) is from the definition of $\mathbf{C}(l_i)$ in (6.20). Thus, we have proved (6.32). To prove (6.33), we note that from the i.i.d. assumption of l_i,

$$\mathbb{E}_f\left[\|\mathbf{E}_{\text{uncoded}}\|^2\right] = \mathbb{E}_f\left[\sum_{i=1}^{k} \text{trace}\left(\mathbf{C}(l_i)\right)\right] = k\mathbb{E}_f[\text{trace}(\mathbf{C}(l_1))]. \tag{D.44}$$

D.3.1 Proof of Theorem 6.3.7

From Theorem 6.3.5,

$$\mathbb{E}_f\left[\|\mathbf{E}_{\text{uncoded}}\|^2\right] - \mathbb{E}_f\left[\|\mathbf{E}_{\text{coded}}\|^2\right] \geq \mathbb{E}_f[\text{trace}(\mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top)]. \tag{D.45}$$

We now argue that to show (6.41), we only need to show

$$\lim_{n\to\infty} \frac{1}{n-k}\mathbb{E}_f[\text{trace}(\mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top)] \geq \frac{\text{var}_f[\text{trace}(\mathbf{C}(l_1))]}{\mathbb{E}_f[\text{trace}(\mathbf{C}(l_1))]}, \tag{D.46}$$

because then, we have

$$\begin{aligned}
&\lim_{n\to\infty} \frac{1}{(n-k)}\left[\mathbb{E}_f\left[\|\mathbf{E}_{\text{uncoded}}\|^2\right] - \mathbb{E}_f\left[\|\mathbf{E}_{\text{coded}}\|^2\right]\right] \\
\overset{(a)}{\geq} &\lim_{n\to\infty} \frac{1}{(n-k)}\mathbb{E}_f[\text{trace}(\mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top)] \overset{(b)}{\geq} \frac{\text{var}_f[\text{trace}(\mathbf{C}(l_1))]}{\mathbb{E}_f[\text{trace}(\mathbf{C}(l_1))]},
\end{aligned} \tag{D.47}$$

where (a) follows from (D.45) and (b) follows from (D.46).

Also note that after we prove (6.41), then using (6.39), we have

$$\mathbb{E}_f\left[\|\mathbf{E}_{\text{uncoded}}\|^2\right] - \mathbb{E}_f\left[\left\|\mathbf{E}_{\text{rep}}\right\|^2\right] \leq (n-k)\mathbb{E}_f[\text{trace}(\mathbf{C}(l_1))], \tag{D.48}$$

so we have

$$\begin{aligned} &\lim_{n\to\infty}\frac{1}{(n-k)}\left[\mathbb{E}_f\left[\|\mathbf{E}_{\text{uncoded}}\|^2\right] - \mathbb{E}_f\left[\left\|\mathbf{E}_{\text{rep}}\right\|^2\right]\right] \\ \leq &\mathbb{E}_f[\text{trace}(\mathbf{C}(l_1))] \overset{(a)}{\leq} \frac{1}{\rho}\frac{\text{var}_f[\text{trace}(\mathbf{C}(l_1))]}{\mathbb{E}_f[\text{trace}(\mathbf{C}(l_1))]} \\ \leq &\frac{1}{\rho}\lim_{n\to\infty}\frac{1}{(n-k)}\left[\mathbb{E}_f\left[\|\mathbf{E}_{\text{uncoded}}\|^2\right] - \mathbb{E}_f\left[\|\mathbf{E}_{\text{coded}}\|^2\right]\right], \end{aligned} \tag{D.49}$$

which means coded computation beats uncoded computation. Note that step (a) holds because of the variance heavy-tail property.

Therefore, we only need to prove (D.46). The proof of (D.46) is divided into two steps, and intuition behind each step is provided along the proof. The main intuition is that the Fourier structure of the matrix $\mathbf{F}$ makes the matrix $\mathbf{J}_4$ concentrates around its mean value, which makes the most tricky term $\mathbb{E}_f[\text{trace}(\mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top)]$ analyzable.

Exploiting the Fourier structure to obtain a Toeplitz covariance matrix

First, we claim that when $\mathbf{F}_{n\times n}$ is the Fourier transform matrix, the matrix $\mathbf{F}\mathbf{\Lambda}\mathbf{F}^\top$ in (6.24)

$$\mathbf{F}\mathbf{\Lambda}\mathbf{F}^\top = \begin{bmatrix} \mathbf{J}_1 & \mathbf{J}_2 \\ \mathbf{J}_2^\top & \mathbf{J}_4 \end{bmatrix}_{n\times n}, \tag{D.50}$$

is a Toeplitz matrix composed of the Fourier coefficients of the sequence (vector) $s = [\text{trace}(\mathbf{C}(l_1)), \ldots, \text{trace}(\mathbf{C}(l_n))]$. In what follows, we use the simplified notation

$$s_j := \text{trace}(\mathbf{C}(l_{j+1})), j = 0, 1, \ldots, n-1. \tag{D.51}$$

Lemma D.3.1. *If*

$$\mathbf{F} = \left(\frac{w^{pq}}{\sqrt{n}}\right)_{p,q=0,1,\ldots,n-1}, \tag{D.52}$$

where $w = \exp(-2\pi i/n)$, *then*

$$\mathbf{F}\mathbf{\Lambda}\mathbf{F}^\top = \textit{Toeplitz}[\tilde{s}_p]_{p=0,1,\ldots,n-1}, \tag{D.53}$$

where

$$\tilde{s}_p = \frac{1}{n}\sum_{j=0}^{n-1} w^{-pj} s_j \tag{D.54}$$

Proof. The entry on the l-th row and the m-th column of $\mathbf{F}\mathbf{\Lambda}\mathbf{F}^\top$ is

$$[\mathbf{F}\mathbf{\Lambda}\mathbf{F}^\top]_{l,m} = \sum_{j=0}^{n-1} \frac{w^{lj}}{\sqrt{n}} \frac{w^{-mj}}{\sqrt{n}} s_j = \frac{1}{n} \sum_{j=0}^{n-1} w^{(l-m)j} s_j. \tag{D.55}$$

Thus, Lemma D.3.1 holds. $\square$

Therefore, the variance of all entries of $\mathbf{F}\mathbf{\Lambda}\mathbf{F}^\top$ is the same because

$$\text{var}_f[\tilde{s}_p] = \text{var}_f\left[\frac{1}{n}\sum_{j=0}^{n-1} w^{-pj} s_j\right] = \frac{1}{n}\text{var}_f\left[s_0\right] =: \frac{1}{n}v. \tag{D.56}$$

Further, the means of all diagonal entries of $\mathbf{F}\mathbf{\Lambda}\mathbf{F}^\top$ are

$$\mathbb{E}_f[\tilde{s}_0] = \mathbb{E}_f\left[s_0\right] =: \mu, \tag{D.57}$$

while the means of all off-diagonal entries are

$$\mathbb{E}_f[\tilde{s}_p] = \frac{1}{n}\sum_{j=0}^{n-1} w^{-pj}\mathbb{E}_f[s_j] = 0, \forall p \neq 0. \tag{D.58}$$

Using the concentration of $\mathbf{J}_4$ to obtain the error when $n \to \infty$

From an intuitive perspective, when $n \to \infty$, the submatrix $\mathbf{J}_4$ concentrates at $\mu\mathbf{I}_{n-k}$ (see the above computation on the mean and variance of all entries). In this case

$$\mathbb{E}_f[\text{trace}(\mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top)] \approx \frac{1}{\mu}\mathbb{E}_f[\text{trace}(\mathbf{J}_2\mathbf{J}_2^\top)] = \frac{1}{\mu}k(n-k)\text{var}[\tilde{s}_p] = \frac{n-k}{\mu}v \cdot \frac{k}{n}. \tag{D.59}$$

Therefore, we have

$$\lim_{n\to\infty} \frac{1}{n-k}\mathbb{E}_f[\text{trace}(\mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top)] = \frac{v}{\mu} = \frac{\text{var}_f[s_0]}{\mathbb{E}_f[s_0]}. \tag{D.60}$$

Now, we formalize the above intuitive statement. In fact, we will show a even stronger bound than the bound on the expected error.

Lemma D.3.2. *When $n - k = o(\sqrt{n})$, with high probability (in $1 - \mathcal{O}(\frac{(n-k)^2}{n})$),*

$$\frac{1}{n-k}\textit{trace}(\mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top) \geq \frac{1}{\mu+\epsilon}\left(\frac{k}{n}v - \epsilon\right), \tag{D.61}$$

for any $\epsilon > 0$.

After we prove Lemma D.3.2, we obtain a bound on expectation using the fact that

$$\frac{1}{n-k}\mathbb{E}_f[\text{trace}(\mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top)] \geq (1-\mathcal{O}(\frac{(n-k)^2}{n}))\frac{1}{\mu+\epsilon}\left(\frac{k}{n}v-\epsilon\right). \tag{D.62}$$

Thus, when $n \to \infty$ and $n-k = o(\sqrt{n})$,

$$\lim_{n\to\infty}\frac{1}{n-k}\mathbb{E}_f[\text{trace}(\mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top)] \geq \frac{v-\epsilon}{\mu+\epsilon} = \frac{\text{var}_f[s_0]-\epsilon}{\mathbb{E}_f[s_0]+\epsilon}, \tag{D.63}$$

for all $\epsilon > 0$, which completes the proof of Theorem 6.3.7.

The proof of Lemma D.3.2 relies on the concentration of trace$(\mathbf{J}_2\mathbf{J}_2^\top)$ and the concentration of $\mathbf{J}_4$. In particular, when we prove the concentration of $\mathbf{J}_4$, we use the Gershgorin circle theorem [94]. First, we show the following Lemma.

Lemma D.3.3. *When $n-k = o(n)$, with high probability (in $1-\mathcal{O}(\frac{n-k}{n})$)*

$$\frac{1}{n-k}\textit{trace}(\mathbf{J}_2\mathbf{J}_2^\top) \geq \frac{k}{n}v-\epsilon. \tag{D.64}$$

Proof. Since $(\mathbf{J}_2)_{k\times(n-k)} := [\mathbf{J}_{i,j}]$ ($\mathbf{J}_{i,j}$ represents the entry on the i-th row and the j-th column) is the upper-right submatrix of $\mathbf{F\Lambda F}^\top = \text{Toeplitz}[\tilde{s}_p]_{p=0,1,\dots,n-1}$,

$$\text{trace}(\mathbf{J}_2\mathbf{J}_2^\top) = \sum_{i=1}^{k}\sum_{j=1}^{n-k}|\mathbf{J}_{i,j}|^2 = \sum_{l=1}^{k}\sum_{m=k+1}^{n}|\tilde{s}_{m-l}|^2. \tag{D.65}$$

Since all entries in $\mathbf{J}_2$ have zero mean (because $l \neq m$ ever in (D.65) and from (D.58) all off-diagonal entries have zero mean) and have the same variance $\frac{v}{n}$ (see (D.56)),

$$\mathbb{E}_f\left[\frac{1}{n-k}\text{trace}(\mathbf{J}_2\mathbf{J}_2^\top)\right] = \frac{1}{n-k}\cdot k(n-k)\mathbb{E}_f[|\tilde{s}_1|^2] \overset{(a)}{=} \frac{1}{n-k}\cdot k(n-k)\text{var}_f[\tilde{s}_1] = \frac{k}{n}v, \tag{D.66}$$

where (a) holds because $\mathbb{E}_f[\tilde{s}_1] = 0$. To prove (D.64), we compute the variance of trace$(\mathbf{J}_2\mathbf{J}_2^\top)$ and use Chebyshev's inequality to bound the tail probability. Define

$$\mu_B := \mathbb{E}_f[\text{trace}(\mathbf{J}_2\mathbf{J}_2^\top)] \overset{(a)}{=} \frac{k(n-k)}{n}v, \tag{D.67}$$

where (a) follows from (D.66). From (D.65), we have

$$\text{trace}(\mathbf{J}_2\mathbf{J}_2^\top) \leq (n-k)\sum_{p=1}^{n-1}|\tilde{s}_p|^2 \overset{(a)}{=} (n-k)\left(\frac{1}{n}\sum_{j=0}^{n-1}s_j^2 - |\tilde{s}_0|^2\right), \tag{D.68}$$

where the last equality (a) holds due to Parseval's equality for the Fourier transform, which states that $\frac{1}{n}\sum_{j=0}^{n-1}s_j^2 = \sum_{p=0}^{n-1}|\tilde{s}_p|^2$. Then,

$$\begin{aligned}\text{var}_f\left[\frac{1}{n-k}\text{trace}(\mathbf{J}_2\mathbf{J}_2^\top)\right] &= \mathbb{E}_f\left[\left(\frac{1}{n-k}\text{trace}(\mathbf{J}_2\mathbf{J}_2^\top)\right)^2\right] - \mathbb{E}_f^2\left[\frac{1}{n-k}\text{trace}(\mathbf{J}_2\mathbf{J}_2^\top)\right]\\ \overset{(a)}{\leq}\mathbb{E}_f\left[\left(\frac{1}{n}\sum_{j=0}^{n-1}s_j^2-|\tilde{s}_0|^2\right)^2\right] - \frac{k^2}{n^2}v^2 &\overset{(b)}{=} \mathbb{E}_f\left[\left(\frac{1}{n}\sum_{j=0}^{n-1}s_j^2-(\frac{1}{n}\sum_{j=0}^{n-1}s_j)^2\right)^2\right] - \frac{k^2}{n^2}v^2,\end{aligned} \tag{D.69}$$

where (a) follows from (D.66) and (D.68) and (b) follows from (D.54). Note that

$$\frac{1}{n}\sum_{j=0}^{n-1} s_j^2 - (\frac{1}{n}\sum_{j=0}^{n-1} s_j)^2 = \frac{n-1}{n}s^2, \tag{D.70}$$

where

$$s^2 := \frac{1}{n-1}\sum_{j=0}^{n-1}(s_j - \bar{s})^2, \tag{D.71}$$

is the famous statistic called "unbiased sample variance", and its variance is (see Page 229, Theorem 2 in [177])

$$\mathrm{var}[s^2] = \frac{1}{n}\left(\mu_4 - \frac{n-3}{n-1}\mu_2^2\right), \tag{D.72}$$

where

$$\mu_4 = \mathbb{E}[(s_0-\mu)^4], \tag{D.73}$$

and

$$\mu_2 = \mathbb{E}[(s_0-\mu)^2] = \mathrm{var}[s_0] = v. \tag{D.74}$$

Also note that the sample variance is unbiased, which means

$$\mathbb{E}_f[s^2] = v. \tag{D.75}$$

Therefore, we have

$$\mathbb{E}_f[(s^2)^2] = \mathrm{var}[s^2] + (\mathbb{E}_f[s^2])^2 = \frac{1}{n}\left(\mu_4 - \frac{n-3}{n-1}v^2\right) + v^2, \tag{D.76}$$

so we have

$$\begin{aligned}
&\mathrm{var}_f\left[\frac{1}{n-k}\mathrm{trace}(\mathbf{J}_2\mathbf{J}_2^\top)\right] \overset{(a)}{\leq} \mathbb{E}_f\left[\left(\frac{1}{n}\sum_{j=0}^{n-1}s_j^2 - (\frac{1}{n}\sum_{j=0}^{n-1}s_j)^2\right)^2\right] - \frac{k^2}{n^2}v^2\\
&\overset{(b)}{=}\mathbb{E}_f\left[(\frac{n-1}{n}s^2)^2\right] - \frac{k^2}{n^2}v^2 = \frac{(n-1)^2}{n^2}\mathbb{E}_f[(s^2)^2] - \frac{k^2}{n^2}v^2\\
&\overset{(c)}{=}\frac{(n-1)^2}{n^2}\frac{1}{n}\left(\mu_4 - \frac{n-3}{n-1}v^2\right) + \frac{(n-1)^2-k^2}{n^2}v^2 = \mathcal{O}\left(\frac{1}{n}\right) + \frac{(n-1)^2-k^2}{n^2}v^2,
\end{aligned} \tag{D.77}$$

where (a) follows from (D.69), (b) follows from (D.70) and (c) follow from (D.76).

Note that we have computed the expectation of $\frac{1}{n-k}\mathrm{trace}(\mathbf{J}_2\mathbf{J}_2^\top)$, which is $\frac{k}{n}v$ (see (D.66)). Using the Chebyshev's inequality

$$\begin{aligned}
&\Pr\left(\left|\frac{1}{n-k}\mathrm{trace}(\mathbf{J}_2\mathbf{J}_2^\top) - \frac{k}{n}v\right| \geq \epsilon\right) \leq \frac{1}{\epsilon^2}\mathrm{var}\left[\frac{1}{n-k}\mathrm{trace}(\mathbf{J}_2\mathbf{J}_2^\top)\right]\\
&\overset{(a)}{\leq}\frac{1}{\epsilon^2}\mathcal{O}\left(\frac{1}{n}\right) + \frac{1}{\epsilon^2}\frac{(n-1)^2-k^2}{n^2}v^2 = \frac{1}{\epsilon^2}\mathcal{O}\left(\frac{1}{n}\right) + \frac{1}{\epsilon^2}\frac{(n-k-1)(n+k-1)}{n^2}v^2\\
&\overset{(b)}{<}\frac{1}{\epsilon^2}\mathcal{O}\left(\frac{1}{n}\right) + \frac{2}{\epsilon^2}\frac{n-k-1}{n}v^2 = \frac{1}{\epsilon^2}\mathcal{O}\left(\frac{n-k}{n}\right).
\end{aligned} \tag{D.78}$$

where (a) is from (D.77) and (b) is because $n+k-1<2n$. Therefore, the proof of (D.64) is over. $\square$

Next, we show that with high probability the largest eigenvalue of $(\mathbf{J}_4)_{(n-k)\times(n-k)}$ is smaller than $(1+\epsilon)\mu$. Note that the matrix $\mathbf{J}_4$ is a principle submatrix of the Toeplitz matrix $\mathbf{F}\boldsymbol{\Lambda}\mathbf{F}^\top = \text{Toeplitz}[\tilde{s}_p]_{p=0,1,\ldots,n-1}$, so $\mathbf{J}_4 = \text{Toeplitz}[\tilde{s}_p]_{p=0,1,\ldots,n-k-1}$ is also Toeplitz. Using the Gershgorin circle theorem, all eigenvalues of $\mathbf{J}_4 := [\tilde{\mathbf{J}}_{ij}]$ must lie in the union of $(n-k)$ circles, in which the i-th circle is centered at the diagonal entry $\tilde{\mathbf{J}}_{ii} = \tilde{s}_0$ and has radius $\sum_{j\neq i}|\tilde{\mathbf{J}}_{ij}| = \sum_{j\neq i}|\tilde{s}_{j-i}|$. These $(n-k)$ circles are all within the circle centered at $\tilde{s}_0$ with radius $2\sum_{p=1}^{n-k-1}|\tilde{s}_p|$. Therefore, the maximum eigenvalue of $\mathbf{J}_4$ satisfies

$$\sigma_{\max} < \tilde{s}_0 + 2\sum_{p=1}^{n-k-1}|\tilde{s}_p|. \tag{D.79}$$

Thus,

$$\Pr(\sigma_{\max} > \mu+\epsilon) < \Pr\left(\tilde{s}_0 + 2\sum_{p=1}^{n-k-1}|\tilde{s}_p| > \mu+\epsilon\right) = \Pr\left(\left(\tilde{s}_0 - \mu + 2\sum_{p=1}^{n-k-1}|\tilde{s}_p|\right)^2 > \epsilon^2\right)$$
$$\overset{(a)}{\leq} \frac{1}{\epsilon^2}\mathbb{E}\left[\left(\tilde{s}_0 - \mu + 2\sum_{p=1}^{n-k-1}|\tilde{s}_p|\right)^2\right] \overset{(b)}{\leq} \frac{1}{\epsilon^2}(2n-2k-1)^2\frac{v}{n} = \frac{1}{\epsilon^2}\mathcal{O}\left(\frac{(n-k)^2}{n}\right), \tag{D.80}$$

where (a) is from the Markov inequality and (b) is due to the fact that $\text{var}[\tilde{s}_p] = \frac{v}{n}$ for all p and $\mathbb{E}[\tilde{s}_0] = \mu$ and $\mathbb{E}[\tilde{s}_p] = 0$ for all $p \neq 0$.

From Lemma D.3.3 and (D.80), when $n\to\infty$ and $(n-k)^2 = o(n)$, with high probability (which is $1 - \frac{1}{\epsilon^2}\mathcal{O}\left(\frac{(n-k)^2}{n}\right)$),

$$\frac{1}{n-k}\text{trace}(\mathbf{J}_2\mathbf{J}_2^\top) \geq \frac{k}{n}v - \epsilon, \tag{D.81}$$

and at the same time

$$\mathbf{J}_4^{-1} \succeq \frac{1}{\mu+\epsilon}\mathbf{I}_{n-k}. \tag{D.82}$$

From concentration of $\text{trace}(\mathbf{J}_2\mathbf{J}_2^\top)$ and the lower bound of $\mathbf{J}_4^{-1}$, we have, with high probability,

$$\frac{1}{n-k}\text{trace}(\mathbf{J}_2\mathbf{J}_4^{-1}\mathbf{J}_2^\top) \geq \frac{1}{\mu+\epsilon}\left(\frac{k}{n}v - \epsilon\right), \tag{D.83}$$

for all ϵ. This concludes the proof of Lemma D.3.2 and hence completes the proof of Theorem 6.3.7 (see the details from after Lemma D.3.2 to equation (D.63)). This lemma is a formal statement of equality (D.60).

D.3.2 Computing the matrix Λ

Recall that the statistic $\hat{\gamma}_{m,l}$ is defined as

$$\hat{\gamma}_{m,l} = \frac{1}{m}\sum_{j=1}^{m} \left\|\mathbf{B}^l \mathbf{a}_j\right\|^2, l = 1, 2, \ldots T_u. \tag{D.84}$$

The computational complexity of computing $\hat{\gamma}_{m,l}, l = 1, 2, \ldots T_u$ is the same as the computation of m linear inverse problems for T_u iterations. The computation has low complexity and can be carried out distributedly in m workers before the main algorithm starts. Additionally, the computation results can be used repeatedly when we implement the coded linear inverse algorithm multiple times. In the data experiments, we use $m = 10$, which has the same complexity as solving $m = 10$ extra linear inverse problems.

The following Lemma shows that $\hat{\gamma}_{m,l}, l = 1, 2, \ldots T_u$ is an unbiased and asymptotically consistent estimate of trace$(\mathbf{C}(l))$ for all l.

Lemma D.3.4. *The statistic $\hat{\gamma}_{m,l}$ is an unbiased and asymptotically consistent estimator of trace$(\mathbf{C}(l))$. More specifically, the mean and variance of the estimator $\hat{\gamma}_{m,l}$ satisfies*

$$\mathbb{E}[\hat{\gamma}_{m,l}|\mathbf{l}] = \textit{trace}(\mathbf{C}(l)), \tag{D.85}$$

$$var_t[\hat{\gamma}_{m,l}] \leq \frac{1}{m}\left\|\mathbf{B}^l\right\|_F^4 \mathbb{E}\left[\|\mathbf{a}_j\|^4\right]. \tag{D.86}$$

Proof. The expectation of $\hat{\gamma}_{m,l}$ satisfies

$$\begin{aligned}\mathbb{E}[\hat{\gamma}_{m,l}] =& \frac{1}{m}\sum_{j=1}^{m}\mathbb{E}\left[\left\|\mathbf{B}^l\mathbf{a}_j\right\|^2\right] = \mathbb{E}\left[\left\|\mathbf{B}^l\mathbf{a}_1\right\|^2\right] = \mathbb{E}\left[\text{trace}(\mathbf{B}^l\mathbf{a}_1\mathbf{a}_1^\top(\mathbf{B}^l)^\top)\right] \\ &\overset{(a)}{=}\text{trace}(\mathbf{B}^l\mathbb{E}[\mathbf{a}_1\mathbf{a}_1^\top](\mathbf{B}^l)^\top) = \text{trace}(\mathbf{B}^l\mathbf{C}_E(\mathbf{B}^l)^\top) = \text{trace}(\mathbf{C}(l)),\end{aligned} \tag{D.87}$$

where (a) is from the fact that $\mathbf{a}_1$ has covariance $\mathbf{C}_E$. To bound the variance of $\hat{\gamma}_{m,l}$, note that for all j,

$$\left\|\mathbf{B}^l\mathbf{a}_j\right\|^2 \leq \left\|\mathbf{B}^l\right\|_F^2 \|\mathbf{a}_j\|^2. \tag{D.88}$$

Therefore,

$$\text{var}[\hat{\gamma}_{m,l}] = \text{var}[\frac{1}{m}\sum_{j=1}^{m}\|\mathbf{B}^l\mathbf{a}_j\|^2] \overset{(a)}{=} \frac{1}{m}\text{var}\left[\left\|\mathbf{B}^l\mathbf{a}_j\right\|^2\right] \overset{(b)}{\leq} \frac{1}{m}\mathbb{E}\left[\left\|\mathbf{B}^l\mathbf{a}_j\right\|^4\right] \overset{(c)}{\leq} \frac{1}{m}\left\|\mathbf{B}^l\right\|_F^4\mathbb{E}\left[\|\mathbf{a}_j\|^4\right], \tag{D.89}$$

□

where (a) holds because all $\|\mathbf{a}_j\|$ are independent of each other, and (b) holds because var$[X] \leq \mathbb{E}[X^2]$, and (c) is from the Cauchy-Schwartz inequality.

Appendix E

Theoretical proofs for Chapter 7

E.1 Proof of Theorem 7.3.1

Lemma E.1.1. *If the sparsity pattern in Definition 7.3.2 is used and Assumption 9 holds, the projection matrix $\mathbf{V}_t\mathbf{V}_t^\top$ satisfies*

$$\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top] = (1-\delta_t)\mathbf{I}_k, \tag{E.1}$$

where the expectation is taken respect to the randomness of non-zero entries' values (the sparsity pattern $\mathbf{G}$ is fixed) and the randomness of the workers' failure events.

Proof. See Appendix E.2 for the full proof. The proof relies on proving some symmetric properties of $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]$. We prove that the symmetry of standard Gaussian pdf on the real line ensures that all of the off-diagonal entries in $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]$ are zero. Further, we prove that the "combined cyclic" structure of $\mathbf{G}$ in Definition 7.3.2 ensures that all diagonal entries on $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]$ are identical. The two facts above show that $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top] = x\mathbf{I}_k$ for some constant x. Then, we can use a property of trace to compute x and obtain (E.1). □

We denote the projection $\mathbf{V}_t\mathbf{V}_t^\top$ by $\mathbf{P}_V$. Then, from (7.6), $\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top = \mathbf{I}_k - \mathbf{P}_V$. The first term $\text{vec}\left(\mathbf{V}_t\mathbf{V}_t^\top\text{mat}\left(\mathbf{B}\mathbf{e}_t\right)\right) = \text{vec}\left(\mathbf{P}_V\text{mat}\left(\mathbf{B}\mathbf{e}_t\right)\right)$ in (7.11) can be bounded by

$$\begin{aligned}
&\mathbb{E}\left[\|\text{vec}\left(\mathbf{P}_V\text{mat}\left(\mathbf{B}\mathbf{e}_t\right)\right)\|^2\right] \overset{(a)}{=} \mathbb{E}\left[\|(\mathbf{P}_V\otimes\mathbf{I}_b)\mathbf{B}\mathbf{e}_t\|^2\right]\\
\overset{(b)}{=}&\mathbb{E}\left[\text{trace}\left((\mathbf{B}\mathbf{e}_t)^\top(\mathbf{P}_V\otimes\mathbf{I}_b)^\top(\mathbf{P}_V\otimes\mathbf{I}_b)\mathbf{B}\mathbf{e}_t\right)\right]\\
\overset{(c)}{=}&\mathbb{E}\left[\text{trace}\left(\mathbf{B}\mathbf{e}_t(\mathbf{B}\mathbf{e}_t)^\top(\mathbf{P}_V\otimes\mathbf{I}_b)^\top(\mathbf{P}_V\otimes\mathbf{I}_b)\right)\right]\\
\overset{(d)}{=}&\text{trace}\left(\mathbb{E}\left[\mathbf{B}\mathbf{e}_t(\mathbf{B}\mathbf{e}_t)^\top\right]\mathbb{E}\left[(\mathbf{P}_V\otimes\mathbf{I}_b)^\top(\mathbf{P}_V\otimes\mathbf{I}_b)\right]\right)\\
=&\text{trace}\left(\mathbb{E}\left[\mathbf{B}\mathbf{e}_t(\mathbf{B}\mathbf{e}_t)^\top\right]\mathbb{E}\left[(\mathbf{P}_V^\top\mathbf{P}_V)\otimes\mathbf{I}_b\right]\right)\\
\overset{(e)}{=}&\text{trace}\left(\mathbb{E}\left[\mathbf{B}\mathbf{e}_t(\mathbf{B}\mathbf{e}_t)^\top\right]\mathbb{E}\left[\mathbf{P}_V\otimes\mathbf{I}_b\right]\right)\\
\overset{(f)}{=}&\text{trace}\left(\mathbb{E}\left[\mathbf{B}\mathbf{e}_t(\mathbf{B}\mathbf{e}_t)^\top\right]((1-\delta_t)\mathbf{I}_k)\otimes\mathbf{I}_b\right)\\
=&(1-\delta_t)\text{trace}\left(\mathbb{E}\left[\mathbf{B}\mathbf{e}_t(\mathbf{B}\mathbf{e}_t)^\top\right]\right) = (1-\delta_t)\mathbb{E}[\|\mathbf{B}\mathbf{e}_t\|^2],
\end{aligned} \tag{E.2}$$

where (a) is from the property of mat-vec operations, (b) is because $(\mathbf{P}_V \otimes \mathbf{I}_b)\mathbf{B}\mathbf{e}_t$ is a vector, (c) is because trace(AB) = trace(BA), (d) is because trace and $\mathbb{E}$ commutes and the projection $\mathbf{P}_V$ only depends on the random partial generator matrix $\mathbf{G}_s$ and is independent of $\mathbf{e}_t$, (e) is because $\mathbf{P}_V$ is a projection matrix and (f) is from Lemma E.1.1 and $\mathbf{P}_V = \mathbf{V}_t\mathbf{V}_t^\top$. Similarly, we can prove

$$\mathbb{E}\left[\|\text{vec}\left((\mathbf{I}_k - \mathbf{P}_V)\text{mat}\left(\mathbf{e}_t\right)\right)\|^2\right] = \delta_t\mathbb{E}[\|\mathbf{e}_t\|^2]. \tag{E.3}$$

Therefore

$$\begin{aligned} &\mathbb{E}[\|\mathbf{e}_{t+1}\|^2] \\ \overset{(a)}{=}&\mathbb{E}[\left\|\text{vec}\left(\mathbf{V}_t\mathbf{V}_t^\top\text{mat}\left(\mathbf{B}\mathbf{e}_t\right)\right)\right\|^2] + \mathbb{E}[\left\|\text{vec}\left(\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top\text{mat}\left(\mathbf{e}_t\right)\right)\right\|^2] \\ \overset{(b)}{=}&(1-\delta_t)\mathbb{E}[\|\mathbf{B}\mathbf{e}_t\|^2] + \delta_t\mathbb{E}[\|\mathbf{e}_t\|^2], \end{aligned} \tag{E.4}$$

where (a) is from (7.11) and the Pythagorean theorem, and (b) is from (E.2) and (E.3). Thus, we have completed the proof.

E.2 Proof of Lemma E.1.1

Proof. First, notice that although the SVD decomposition of $\mathbf{G}_s^{(t)}$ in (7.4) is not unique, the projection matrix $\mathbf{V}_t\mathbf{V}_t^\top$ is unique for a certain $\mathbf{G}_s^{(t)}$, because it is the projection onto the row space of $\mathbf{G}_s^{(t)}$. Then, before we prove the lemma, we show another lemma that will be useful.

Lemma E.2.1. *Suppose $\mathbf{P}$ is a $k \times k$ orthonormal matrix. Then, if the corresponding projection matrix of $\mathbf{G}_s^{(t)}$ is $\mathbf{V}_t\mathbf{V}_t^\top$, the corresponding projection matrix of $\mathbf{G}_s^{(t)}\mathbf{P}$ is $\mathbf{P}^\top\mathbf{V}_t\mathbf{V}_t^\top\mathbf{P}$.*

Proof. If $\mathbf{G}_s^{(t)}$ has the SVD decomposition $\mathbf{G}_s^{(t)} = \mathbf{U}_t\mathbf{D}_t\mathbf{V}_t^\top$, then $\mathbf{G}_s^{(t)}\mathbf{P} = \mathbf{U}_t\mathbf{D}_t(\mathbf{V}_t^\top\mathbf{P})$ is a valid SVD of $\mathbf{G}_s^{(t)}\mathbf{P}$, which means the projection matrix now should be $(\mathbf{V}_t^\top\mathbf{P})^\top\mathbf{V}_t^\top\mathbf{P} = \mathbf{P}^\top\mathbf{V}_t\mathbf{V}_t^\top\mathbf{P}$. □

Now, we prove two properties of the expected projection matrix.

E.2.1 Property 1: all off-diagonal entries in $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]$ are zero

Suppose $\mathbf{P}_i$ of size $k \times k$ is the diagonal matrix where all diagonal entries are 1 except the i-th diagonal entry is -1. When we multiply $\mathbf{G}_s^{(t)}\mathbf{P}_i$, we effectively flips the i-th column of $\mathbf{G}_s^{(t)}$. Now, we observe the fact that the distribution of $\mathbf{G}_s^{(t)}\mathbf{P}_i$ is exactly the same as the distribution of $\mathbf{G}_s^{(t)}$, because all non-zeros in $\mathbf{G}$ have unit Gaussian distribution and the positive and negative part of this distribution is symmetric. Therefore, the projection $\mathbf{V}_t\mathbf{V}_t^\top$, which is a deterministic function of $\mathbf{G}_s^{(t)}$, has the same distribution as the projection $\mathbf{P}_i^\top\mathbf{V}_t\mathbf{V}_t^\top\mathbf{P}_i$, which means

$$\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top] = \mathbb{E}[\mathbf{P}_i^\top\mathbf{V}_t\mathbf{V}_t^\top\mathbf{P}_i] = \mathbf{P}_i^\top\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]\mathbf{P}_i. \tag{E.5}$$

By the structure of $\mathbf{P}_i$, this means that after flipping the i-th column and the i-th row of $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]$, the matrix stays the same. The only way that this can be true is that all entries on the i-th row and the i-th column are zero, except the diagonal entry.

E.2.2 Property 2: all diagonal entries in $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]$ are equal

Suppose $\mathbf{P}_\pi$ is the $k \times k$ permutation matrix

$$\mathbf{P}_\pi = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{E.6}$$

It is an orthonormal matrix. When we multiply $\mathbf{G}_s^{(t)}\mathbf{P}_\pi$, we effectively do a cyclic shift on the columns of $\mathbf{G}_s^{(t)}$ by pushing each column to its left (except for the left most column which is pushed to the right-most). Then, we show that the distribution of $\mathbf{G}_s^{(t)}$ is again exactly the same as the distribution of $\mathbf{G}_s^{(t)}\mathbf{P}_\pi$. To prove this, we look at a specific random initialization of $\mathbf{G}_s^{(t)}\mathbf{P}_\pi$. Suppose $\mathbf{G}_s^{(t)}$ occupies a rows in $\mathbf{S}_1$ and b rows in $\mathbf{S}_2$, where recall that $\mathbf{S}_1$ and $\mathbf{S}_2$ are square cyclic matrices (see Definition 7.3.2). Suppose the a rows are the $i_1, i_2, \ldots, i_a$-th rows in $\mathbf{S}_1$ and the b rows are the $j_1, j_2, \ldots, j_b$-th rows in $\mathbf{S}_2$. Then, after permuting the columns of $\mathbf{G}_s^{(t)}$ by $\mathbf{P}_\pi$, the sparsity pattern of $\mathbf{G}_s^{(t)}\mathbf{P}_\pi$ is the same as if the $(i_1-1, i_2-1, \ldots, i_a-1)(\text{mod } k)$-th rows in $\mathbf{S}_1$ and the $(j_1-1, j_2-1, \ldots, j_b-1)(\text{mod } k)$-th rows in $\mathbf{S}_2$ are chosen for a realization of $\mathbf{G}_s^{(t)}$. This ensures that there exists a realization of $\mathbf{G}_s^{(t)}$ which is exactly the same as the examined realization of $\mathbf{G}_s^{(t)}\mathbf{P}_\pi$. These two realizations have a one-to-one mapping because no other realizations of $\mathbf{G}_s^{(t)}\mathbf{P}_\pi$ will lead to the same realization[1] of $\mathbf{G}_s^{(t)}$ and the pdfs of these two realizations are also the same, because all rows are selected uniformly and the all entries have the same unit Gaussian distribution. Based on this one-to-one mapping, we know that the distribution of $\mathbf{G}_s^{(t)}$ and $\mathbf{G}_s^{(t)}\mathbf{P}_\pi$ are the same. Therefore, similar to (E.5), we obtain

$$\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top] = \mathbf{P}_\pi^\top \mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]\mathbf{P}_\pi. \tag{E.7}$$

From Property 1 we have proved that all the off-diagonal entries in $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]$ are 0. From (E.7), we have that the cyclic shift on the diagonal also remains the same. This means all diagonal entries in $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]$ are identical.

[1]There is a subtle point here that when the cyclic rows of $\mathbf{S}_1$ and $\mathbf{S}_2$ are the same, the mapping is not one-to-one anymore. Therefore, $\mathbf{S}_1$ and $\mathbf{S}_2$ can be chosen to have different cyclic rows.

E.2.3 Prove $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top] = (1-\delta_t)\mathbf{I}_k$

From Property 1 and Property 2, we can assume that $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top] = x\mathbf{I}_k$ for some constant x. Notice that on one hand,

$$\begin{aligned} &\text{trace}[\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]] = \mathbb{E}[\text{trace}[\mathbf{V}_t\mathbf{V}_t^\top]] \\ =&\mathbb{E}[\text{trace}[\mathbf{V}_t^\top\mathbf{V}_t]] = \mathbb{E}[\text{rank}(\mathbf{G}_s^{(t)})]. \end{aligned} \tag{E.8}$$

One the other hand,

$$\text{trace}[\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]] = \text{trace}[x\mathbf{I}_k] = xk, \tag{E.9}$$

so we have $x = \frac{\mathbb{E}[\text{rank}(\mathbf{G}_s^{(t)})]}{k} = 1-\delta_t$. □

E.3 Equivalence of several norms

E.3.1 For $G(N,p)$ the induced two-norm $\|\mathbf{A}\|$ is close to $\rho(\mathbf{A})$

We prove a lemma stating that for $G(N,p)$, the induced two-norm $\|\mathbf{A}\|$ is close to $\rho(\mathbf{A})$.

Lemma E.3.1. *For random graph from the Erdös-Rényi model $G(N,p)$, the column-normalized adjacency matrix $\mathbf{A}$ satisfies*

$$Pr\left(\|\mathbf{A}\| > \sqrt{\frac{1+\epsilon}{1-\epsilon}}\rho(\mathbf{A})\right) < 3Ne^{-\epsilon^2Np/8}. \tag{E.10}$$

Proof. It is well known that the node degrees of Erdös-Rényi graphs concentrate at Np. For example, Theorem 4.1 in the online book chapter here `https://www.cs.cmu.edu/~avrim/598/chap4only.pdf` has the following theorem.

Theorem E.3.2. *Let v be a vertex of the random graph $G(N,p)$. For $0 < \alpha < \sqrt{Np}$*

$$Pr(|Np - deg(v)| \geq \alpha\sqrt{Np}) \leq 3e^{-\alpha^2/8}. \tag{E.11}$$

Therefore, by the union bound, with probability at least $1-3Ne^{-\epsilon^2Np/8}$, all nodes in the graph have degree within the range $(Np(1-\epsilon), Np(1+\epsilon))$. The matrix $\mathbf{A}$ is the column-normalized adjacency matrix, so its spectral radius $\rho(\mathbf{A}) = 1$. By Hölder's inequality

$$\|\mathbf{A}\|_2 \leq \sqrt{\|\mathbf{A}\|_1\|\mathbf{A}\|_\infty}. \tag{E.12}$$

The induced 1-norm $\|\mathbf{A}\|_1$ is also the maximum absolute column sum and the induced infinity norm $\|\mathbf{A}\|_\infty$ is also the maximum absolute row sum. Since $\mathbf{A}$ is the column-normalized, the induced 1-norm satisfies

$$\|\mathbf{A}\|_1 = 1. \tag{E.13}$$

Since the degree of the graph is within range $(Np(1-\epsilon), Np(1+\epsilon))$ with high probability, the minimum normalization factor for a node v when doing column normalization, i.e., the minimum column sum of the un-normalized adjacency matrix, is greater than $Np(1-\epsilon)$ with high probability. This means that with high probability, all non-zeros in $\mathbf{A}$ are smaller than $\frac{1}{Np(1-\epsilon)}$. Moreover, the maximum number of non-zeros of $\mathbf{A}$ in each row is also smaller than $Np(1+\epsilon)$ with high probability. This means that the maximum row sum $\|\mathbf{A}\|_\infty$ satisfies the following with high probability

$$\|\mathbf{A}\|_\infty \leq \frac{Np(1+\epsilon)}{Np(1-\epsilon)} = \frac{1+\epsilon}{1-\epsilon}. \tag{E.14}$$

From (E.12) to (E.14), with high probability (at least $1-3Ne^{-\epsilon^2 Np/8}$),

$$\|\mathbf{A}\|_2 \leq \sqrt{\frac{1+\epsilon}{1-\epsilon}} = \sqrt{\frac{1+\epsilon}{1-\epsilon}}\rho(\mathbf{A}). \tag{E.15}$$

This completes the proof. □

Appendix F

Theoretical proofs for Chapter 8

F.1 Proof of Corollary 8.3.3

Considering (8.2) and (8.6), to prove (8.12), it suffices to show $\mathscr{C}^{(\mathrm{N})} = \Omega(N \log\log N)$. In fact, it is stated in Theorem 1 in [5] that if the number of noisy broadcasts is

$$\mathscr{C}^{(\mathrm{N})} = \beta(N)N,$$

the error probability $P_e^{(N)}$ that the receiver does not output all self-information bits satisfies

$$1 - P_e^{(N)} < \sqrt{\frac{1}{N}} + \frac{48\beta^2 \log(1/\epsilon)}{\epsilon^{4\beta} \log N}. \tag{F.1}$$

Then, we have

$$\text{Inequality (F.1)} \iff \left(1 - P_e^{(N)} - \sqrt{\frac{1}{N}}\right) \frac{\log N}{48 \log(1/\epsilon)} < \frac{\beta^2}{\epsilon^{4\beta}}$$

$$\iff \log\log N + \log\left(1 - P_e^{(N)} - \sqrt{\frac{1}{N}}\right) - \log\left(48 \log\left(\frac{1}{\epsilon}\right)\right) < 2\log\beta + 4\beta \log\left(\frac{1}{\epsilon}\right).$$

Dividing both the LHS and the RHS with $4\log\left(\frac{1}{\epsilon}\right)$, we have

$$\beta + \frac{\log\beta}{2\log\frac{1}{\epsilon}} > \frac{\log\log N}{4\log\frac{1}{\epsilon}} + \frac{\log(1 - P_e^{(N)} - \sqrt{\frac{1}{N}}) - \log(48\log(\frac{1}{\epsilon}))}{4\log\frac{1}{\epsilon}} = \Omega(\log\log N). \tag{F.2}$$

From (F.2), we immediately have $\beta(N) = \Omega(\log\log N)$.

F.2 Proof of (8.7) in Theorem 8.3.2

Since the code length at each node $v \in \mathcal{B}_\mathcal{T}$ is $\gamma \log N$, according to Lemma 2.2.3, the decoding error probability is

$$P_{e,v} < \exp[-(\gamma \log N + 1)E_r(\epsilon, R)/R] = \exp[-E_r(\epsilon, R)/R] N^{-\gamma E_r(\epsilon,R)/R}. \tag{F.3}$$

Similarly, the decoding error probability at a node $v \in \mathcal{A}_\mathcal{T}$ is

$$P_{e,v} < \exp[-(\mathcal{D}_v + 1)E_r(\epsilon, R)/R] < \exp[-\frac{\gamma}{R}\log N E_r(\epsilon, R)] < N^{-\gamma E_r(\epsilon,R)/R}, \tag{F.4}$$

where we used the fact that the message size $\mathcal{D}_v$ in v is greater than or equal to $\gamma \log N$, and hence we can find a code with length $\lceil(\mathcal{D}_v + 1)/R\rceil > \frac{\gamma}{R}\log N$.

Combining (F.3) and (F.4) and using the union bound, the error probability is bounded as follows

$$\begin{aligned} P_e^{(N)} <& \sum_{v\in\mathcal{A}_\mathcal{T}} P_{e,v} + \sum_{v\in\mathcal{B}_\mathcal{T}} P_{e,v} \\ <& N \cdot N^{-\gamma E_r(\epsilon,R)/R} + N \cdot \exp[-E_r(\epsilon, R)/R] N^{-\gamma E_r(\epsilon,R)/R} \\ =& N^{-(\frac{\gamma E_r(\epsilon,R)}{R}-1)} \cdot (1 + \exp[-E_r(\epsilon, R)/R]) . \end{aligned} \tag{F.5}$$

When the condition $R < \gamma E_r(\epsilon, R)$ is satisfied, the error probability in (F.5) satisfies the property that $\lim_{N\to\infty} P_e^{(N)} = 0$ and the convergence rate is polynomial. This concludes the proof.

F.3 Proof of Lemma F.5.1

We know from the union bound that

$$P_e^{\mathcal{G}}(\mathbf{x}) \le \sum_{\mathbf{x}_1^\top \in\{0,1\}^N\setminus\{\mathbf{x}\}} P_e^{\mathcal{G}}(\mathbf{x} \to \mathbf{x}_1). \tag{F.6}$$

Lemma F.3.1. *The probability that $\mathbf{x}_1$ is confused with $\mathbf{x}_2$ equals the probability that $\mathbf{x}_1 - \mathbf{x}_2$ is confused with the N-dimensional zero vector $\mathbf{0}_N$, i.e.,*

$$P_e^{\mathcal{G}}(\mathbf{x}_1 \to \mathbf{x}_2) = P_e^{\mathcal{G}}(\mathbf{x}_1 - \mathbf{x}_2 \to \mathbf{0}_N). \tag{F.7}$$

Proof. We define an *erasure matrix* $\mathbf{E}$ as a $2N$-by-$2N$ diagonal matrix in which each diagonal entry is either an 'e' or a 1. Define an extended binary multiplication operation with 'e', which has the rule that $ae = e, a \in \{0, 1\}$. The intuition is that both 0 and 1 become an erasure after being erased. Under this definition, the event that $\mathbf{x}_1$ is confused with $\mathbf{x}_2$ can be written as

$$\mathbf{x}_1^\top \cdot [\mathbf{I}, \mathbf{A}] \cdot \mathbf{E} = \mathbf{x}_2^\top \cdot [\mathbf{I}, \mathbf{A}] \cdot \mathbf{E}, \tag{F.8}$$

where a diagonal entry in $\mathbf{E}$ being 'e' corresponds to erasure/removal of the corresponding linear equation. We know that if the erasure matrix $\mathbf{E}$ remains the same, we can arrange the two terms and write

$$(\mathbf{x}_1^\top - \mathbf{x}_2^\top) \cdot [\mathbf{I}, \mathbf{A}] \cdot \mathbf{E} = 0_N^\top \cdot [\mathbf{I}, \mathbf{A}] \cdot \mathbf{E}. \tag{F.9}$$

That is to say, if $\mathbf{x}_1$ is confused with $\mathbf{x}_2$, then, if all the erasure events are the same and the self-information bits are changed to $\mathbf{x}_1 - \mathbf{x}_2$, they will be confused with the all zero vector $\mathbf{0}_N$ and vice-versa. Thus, in order to prove (F.7), we only need to show that the probability of having particular erasure events remains the same with different self-information bits. This claim is satisfied, because by the BEC assumption the erasure events are independent of the channel inputs and identically distributed. □

Thus, using the result from Lemma F.3.1, we obtain

$$P_e^{\mathcal{G}}(\mathbf{x}) \leq \sum_{\mathbf{x}_1^\top \in \{0,1\}^N \setminus \{\mathbf{x}\}} P_e^{\mathcal{G}}(\mathbf{x} - \mathbf{x}_1 \to \mathbf{0}_N), \tag{F.10}$$

and hence, (F.14) holds.

F.4 Proof of Lemma F.5.3

First, we notice that for $1 \leq i \leq N$, the vector $\tilde{\mathbf{x}}^\top$ received is the noisy version of $\mathbf{x}_0^\top$. Since, according to the in-network computing algorithm in Section 8.4.1, the vector $\tilde{\mathbf{x}}^\top$ is obtained in the second step, the event $A_3^{(i)}(\mathbf{x}_0^\top)$ is the only ambiguity event. Moreover, if the i-th entry of $\mathbf{x}_0^\top$ is zero, it does not matter whether an erasure happens to this entry. Thus, the error probability can be calculated by considering all the k non-zero entries, which means

$$\prod_{i=1}^{N} \Pr[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top) \cup A_3^{(i)}(\mathbf{x}_0^\top)] = \epsilon^k.$$

For $N+1 \leq i \leq 2N$, $A_3^{(i)}(\mathbf{x}_0^\top)$ is the erasure event during the second step and is independent from the previous two events $A_1^{(i)}(\mathbf{x}_0^\top)$ and $A_2^{(i)}(\mathbf{x}_0^\top)$. Therefore

$$\begin{aligned} &\Pr\left[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top) \cup A_3^{(i)}(\mathbf{x}_0^\top)\right] \\ \leq &\Pr\left[(A_3^{(i)}(\mathbf{x}_0^\top))^C\right] + \Pr\left[A_3^{(i)}(\mathbf{x}_0^\top)\right] \Pr\left[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top)\right] \\ = &1 - \epsilon + \epsilon \Pr\left[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top)\right] \\ = &1 - \epsilon + \epsilon \left(\Pr\left[A_1^{(i)}(\mathbf{x}_0^\top)\right] + \Pr\left[(A_1^{(i)}(\mathbf{x}_0^\top))^C \cap A_2^{(i)}(\mathbf{x}_0^\top)\right]\right). \end{aligned} \tag{F.11}$$

The event $A_1^{(i)}(\mathbf{x}_0^\top)$ happens when the local parity $\mathbf{x}_0^\top \mathbf{a}_i$ equals zero, i.e., in the k locations of non-zero entries in $\mathbf{x}_0^\top$, there are an even number of ones in the corresponding entries in $\mathbf{a}_i$, the i-th column of the graph adjacency matrix $\mathbf{A}$. Denote by l the number of ones in these k corresponding entries in $\mathbf{a}_i$. Since each entry of $\mathbf{a}_i$ takes value 1 independently with probability p, the probability that an even number of entries are 1 in these k locations is

$$\Pr[A_1^{(i)}(\mathbf{x}_0^\top)] = \Pr[l \text{ is even}] = \sum_{l \text{ is even}} p^l (1-p)^{k-l} = \frac{1 + (1-2p)^k}{2}. \tag{F.12}$$

The event $(A_1^{(i)}(\mathbf{x}_0^\top))^C \cap A_2^{(i)}(\mathbf{x}_0^\top)$ indicates that l is odd and at least one entry of all non-zero entries in $\mathbf{x}_0^\top$ is erased. Suppose in the remaining $N-k$ entries in $\mathbf{a}_i$, j entries take the value 1 and hence there are $(l+j)$ 1's in $\mathbf{a}_i$. Therefore, for a fixed l, we have

$$\begin{aligned}\Pr[(A_1^{(i)}(\mathbf{x}_0^\top))^C \cap A_2^{(i)}(\mathbf{x}_0^\top)|l] &= \sum_{j=0}^{N-k}\binom{N-k}{j}p^j(1-p)^{N-k-j}\cdot[1-(1-p_e)^{l+j}]\\ &\le \sum_{j=0}^{N-k}\binom{N-k}{j}p^j(1-p)^{N-k-j}(l+j)p_e,\end{aligned}$$

where p is the edge connection probability and p_e is the probability that a certain bit in $\mathbf{x}_0$ is erased for $t=\frac{\log(\frac{c\log N}{p_{\text{ch}}})}{\log(1/\epsilon)}$ times when transmitted to v_i from one of its neighbors during the first step of the algorithm. Combining the above inequality with Lemma 8.4.1, we get

$$\begin{aligned}&\Pr[(A_1^{(i)})^C \cap A_2^{(i)}(l)]\\ \le&\sum_{j=0}^{N-k}\binom{N-k}{j}p^j(1-p)^{N-k-j}(l+j)\frac{p_{\text{ch}}}{c\log N}\\ =&l\frac{p_{\text{ch}}}{c\log N}\sum_{j=0}^{N-k}\binom{N-k}{j}p^j(1-p)^{N-k-j}+\frac{p_{\text{ch}}}{c\log N}\sum_{j=1}^{N-k}j\binom{N-k}{j}p^j(1-p)^{N-k-j}\\ \overset{(a)}{=}&l\frac{p_{\text{ch}}}{c\log N}+\frac{p_{\text{ch}}p}{c\log N}\sum_{j=1}^{N-k}(N-k)\binom{N-k-1}{j-1}p^{j-1}(1-p)^{N-k-j}\\ =&l\frac{p_{\text{ch}}}{c\log N}+\frac{p_{\text{ch}}(N-k)}{N}\sum_{j=1}^{N-k}\binom{N-k-1}{j-1}p^{j-1}(1-p)^{N-k-j}=l\frac{p_{\text{ch}}}{c\log N}+p_{\text{ch}}\cdot\frac{N-k}{N},\end{aligned}$$

where step (a) follows from $j\binom{N-k}{j}=(N-k)\binom{N-k-1}{j-1}$. Therefore

$$\begin{aligned}&\Pr[(A_1^{(i)})^C \cap A_2^{(i)}]\\ =&\sum_{l\text{ is odd}}\binom{k}{l}p^l(1-p)^{k-l}\Pr[(A_1^{(i)})^C \cap A_2^{(i)}(l)]\\ \le&\sum_{l\text{ is odd}}\binom{k}{l}p^l(1-p)^{k-l}(l\frac{p_{\text{ch}}}{c\log N}+p_{\text{ch}}\cdot\frac{N-k}{N})\\ =&\sum_{l\text{ is odd}}\binom{k}{l}p^l(1-p)^{k-l}p_{\text{ch}}\cdot\frac{N-k}{N}+\sum_{l\text{ is odd}}l\binom{k}{l}p^l(1-p)^{k-l}\frac{p_{\text{ch}}}{c\log N}\\ =&p_{\text{ch}}\cdot\frac{N-k}{N}\sum_{l\text{ is odd}}\binom{k}{l}p^l(1-p)^{k-l}+\frac{kpp_{\text{ch}}}{c\log N}\sum_{l\text{ is odd}}\binom{k-1}{l-1}p^{l-1}(1-p)^{k-l}\\ =&p_{\text{ch}}\cdot\frac{N-k}{N}\frac{1-(1-2p)^k}{2}+p_{\text{ch}}\cdot\frac{k}{N}\frac{1+(1-2p)^{k-1}}{2}\\ \overset{(a)}{\le}&Lp_{\text{ch}}\frac{1-(1-2p)^k}{2},\end{aligned}$$

where the constant L in step (a) is to be determined. Now we show that $L = \frac{2}{1-1/e} + 1$ suffices to ensure that (a) holds. In fact, we only need to prove

$$\frac{N-k}{N}\frac{1-(1-2p)^k}{2} + \frac{k}{N}\frac{1+(1-2p)^{k-1}}{2} \leq L\frac{1-(1-2p)^k}{2}.$$

Since $\frac{N-k}{N} < 1$, it suffices to show that

$$\frac{k}{N}\frac{1+(1-2p)^{k-1}}{2} \leq (L-1)\,\frac{1-(1-2p)^k}{2}.$$

Since $(1-2p)^{k-1} < 1$, it suffices to show that

$$\frac{k}{N} \leq (L-1)\,\frac{1-(1-2p)^k}{2},$$

or equivalently,

$$\frac{2k}{1-(1-2p)^k} \leq N\,(L-1)\,. \tag{F.13}$$

We know that

$$\begin{aligned} 1-(1-2p)^k &\geq 2kp - C_k^2(2p)^2 \\ &= 2kp - 2k(k-1)p^2 \\ &= 2kp\,[1-p(k-1)] \geq 2kp(1-kp). \end{aligned}$$

Thus, when $kp \leq \frac{1}{2}$, $1-(1-2p)^k \geq 2kp(1-kp) \geq kp$ and

$$\frac{2k}{1-(1-2p)^k} \leq \frac{2k}{kp} = \frac{2N}{c\log N} \leq 2N,$$

when $c\log N > 1$. When $kp > \frac{1}{2}$, $(1-2p)^k \leq (1-2p)^{\frac{1}{2p}} \leq \frac{1}{e}$ and

$$\frac{2k}{1-(1-2p)^k} \leq \frac{2k}{1-1/e} \leq \frac{2N}{1-1/e}.$$

Thus, as long as $L \geq 1 + \frac{2}{1-1/e}$, (F.13) holds. Jointly considering (F.12), we get

$$\Pr[A_1^{(i)} \cup A_2^{(i)}] \leq \frac{1+(1-2p)^k}{2} + Lp_{\text{ch}}\frac{1-(1-2p)^k}{2}.$$

Combining (F.11), we finally arrive at

$$\begin{aligned}
\Pr[A_1^{(i)} \cup A_2^{(i)} \cup A_3^{(i)}] &\le \epsilon + (1-\epsilon)\left[\frac{1+(1-2p)^k}{2} + Lp_{\text{ch}}\frac{1-(1-2p)^k}{2}\right] \\
&= \epsilon + (1-\epsilon)\left[1 - (1-Lp_{\text{ch}})\,\frac{1-(1-2p)^k}{2}\right] \\
&= 1 - (1-\epsilon)\,(1-Lp_{\text{ch}})\,\frac{1-(1-2p)^k}{2} \\
&< 1 - (1-\epsilon-Lp_{\text{ch}})\frac{1-(1-2p)^k}{2} \\
&= 1 - (1-\epsilon-Lp_{\text{ch}})\left[1 - \frac{1+(1-2p)^k}{2}\right] \\
&= \epsilon + Lp_{\text{ch}} + (1-\epsilon-Lp_{\text{ch}})\frac{1+(1-2p)^k}{2} \\
&= \varepsilon_0 + (1-\varepsilon_0)\frac{1+(1-2p)^k}{2},
\end{aligned}$$

where $\varepsilon_0 = Lp_{\text{ch}} + \epsilon$.

F.5 Proof of Theorem 8.4.2

From Section 8.4.1, we know that an error occurs when there exist more than one feasible solutions that satisfy the version with possible erasures of (8.18). That is to say, when all positions with erasures are eliminated from the received vector, there are at least two solutions to the remaining linear equations. Denote by $\mathbf{x}_1$ and $\mathbf{x}_2$ two different vectors of self-information bits. We say that $\mathbf{x}_1$ *is confused with* $\mathbf{x}_2$ if the true vector of self-information bits is $\mathbf{x}_1$ but $\mathbf{x}_2$ also satisfies the possibly erased version of (8.18), in which case x_1 is indistinguishable from $\mathbf{x}_2$. Denote by $P_e^{\mathcal{G}}(\mathbf{x}_1 \to \mathbf{x}_2)$ the probability that $\mathbf{x}_1$ is confused with $\mathbf{x}_2$.

The Lemma F.5.1 in the following states that $P_e^{\mathcal{G}}(\mathbf{x})$ is upper bounded by an expression which is independent of the argument $\mathbf{x}$ (self-information bits).

Lemma F.5.1. *The error probability $P_e^{\mathcal{G}}$ can be upper-bounded by*

$$P_e^{\mathcal{G}}(\mathbf{x}) \le \sum_{\mathbf{x}_0^\top \in \{0,1\}^N \setminus \{\mathbf{0}_N\}} P_e^{\mathcal{G}}(\mathbf{x}_0 \to \mathbf{0}_N), \tag{F.14}$$

where $\mathbf{0}_N$ is the N-dimensional zero vector.

Proof. See Appendix F.3. □

Each term on the RHS of (F.14) can be interpreted as the probability of the existence of a non-zero vector input $\mathbf{x}_0^\top$ that is confused with the all-zero vector after all the non-zero entries of $\mathbf{x}_0^\top \cdot [\mathbf{I}, \mathbf{A}]$ are erased, in which case $\mathbf{x}_0^\top$ is indistinguishable from the all zero channel input. For example, suppose the code length is $2N = 6$ and the codeword $\mathbf{x}_0^\top \cdot [\mathbf{I}, \mathbf{A}] = [x_1, 0, 0, x_4, x_5, x_6]$ is sent and the output happens to be $\mathbf{r}^\top = [e, 0, 0, e, e, e]$. In this case, we cannot distinguish between the input vector $\mathbf{x}_0^\top$ and the all-zero vector $\mathbf{0}_N^\top$ based on the channel output.

The Lemma F.5.2 in the following states that the expected error of the error event discussed above can be upper-bounded. This upper bound is obtained by decomposing the error event into the union of three error events on each bit.

Lemma F.5.2. *Define $\varepsilon_0 = (\frac{2}{1-1/e} + 1)p_{ch} + \epsilon$, where ϵ is the erasure probability of the BECs and p_{ch} is a constant defined in* (8.13). *Then, the expected error probability $P_e^{(N)}(\mathbf{x}) = \mathbb{E}_\mathcal{G}[P_e^\mathcal{G}(\mathbf{x})]$ can be upper-bounded by*

$$P_e^{(N)}(\mathbf{x}) = \mathbb{E}_\mathcal{G}[P_e^\mathcal{G}(\mathbf{x})] \leq \sum_{k=1}^{N} \binom{N}{k} \epsilon^k \left[\varepsilon_0 + (1-\varepsilon_0) \cdot \frac{1+(1-2p)^k}{2}\right]^N. \tag{F.15}$$

Proof. We will first show how to decompose the error event mentioned in the above example to obtain an upper bound on the conditional error probability $P_e^\mathcal{G}(\mathbf{x})$. Then, we show how to obtain an upper bound on the expected error probability $P_e^{(N)}(\mathbf{x}) = \mathbb{E}_\mathcal{G}[P_e^\mathcal{G}(\mathbf{x})]$. Finally, we compute the expected error probability upper bound using random graph theory.

Decomposing the error event conditioned on $\mathcal{G}$

The ambiguity event mentioned above, i.e., a non-zero vector of self-information bits being confused with the all-zero vector $\mathbf{0}_N$, happens if and only if each entry of the received vector $\mathbf{r}^\top$ is either zero or 'e'. When $\mathbf{x}_0^\top$ and the graph $\mathcal{G}$ are both fixed, different entries in $\mathbf{r}^\top$ are independent of each other. Thus, the ambiguity probability $P_e^\mathcal{G}(\mathbf{x}_0 \to \mathbf{0}_N)$ for a fixed non-zero input $\mathbf{x}_0^\top$ and a fixed graph instance $\mathcal{G}$ is the product of the corresponding ambiguity probability of each entry in $\mathbf{r}^\top$ (being a zero or a 'e').

The ambiguity event of each entry may occur due to structural deficiencies in the graph topology as well as due to erasures. In particular, three events contribute to the error at the i-th entry of $\mathbf{r}^\top$: the product of $\mathbf{x}_0^\top$ and the i-th column of $[\mathbf{I}, \mathbf{A}]$ is zero; the i-th entry of $\mathbf{r}^\top$ is 'e' due to erasures in the first step; the i-th entry is 'e' due to an erasure in the second step. We denote these three events respectively by $A_1^{(i)}(\mathbf{x}_0^\top)$, $A_2^{(i)}(\mathbf{x}_0^\top)$ and $A_3^{(i)}(\mathbf{x}_0^\top)$, where the superscript i and the argument $\mathbf{x}_0^\top$ mean that the events are for the i-th entry and conditioned on a fixed message vector $\mathbf{x}_0^\top$. The ambiguity event on the i-th entry is the union of the above three events. Note that the first event is due to structural deficiency, while the second and the third events are due to erasures. Therefore, by applying the union bound over all possible inputs, the error probability $P_e^\mathcal{G}(\mathbf{x})$ can be upper bounded by

$$P_e^\mathcal{G}(\mathbf{x}) \leq \sum_{\mathbf{x}_0^\top \in \{0,1\}^N \setminus \{\mathbf{0}^N\}} \prod_{i=1}^{2N} \Pr[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top) \cup A_3^{(i)}(\mathbf{x}_0^\top) | \mathcal{G}], \tag{F.16}$$

In this expression, $\mathcal{G}$ is a random graph. The randomness of $\mathcal{G}$ lies in the random edge connections.

Decomposing the unconditioned error event

We will further show that

$$P_e^{(N)}(\mathbf{x}) = \mathbb{E}_{\mathcal{G}}[P_e^{\mathcal{G}}(\mathbf{x})] \leq \sum_{\mathbf{x}_0^\top \in \{0,1\}^N \backslash \{\mathbf{0}^N\}} \prod_{i=1}^{2N} \Pr[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top) \cup A_3^{(i)}(\mathbf{x}_0^\top)], \tag{F.17}$$

We use a set of random binary indicators $\{E_{mn}\}_{m,n=1}^N$ to denote these edges, i.e., $E_{mn} = 1$ if there is a directed edge from node v_m to v_n. Note that we allow self-loops, because each node can certainly broadcasts information to itself. By Assumption (A.4), all random variables in $\{E_{mn}\}_{m,n=1}^N$ are mutually independent. Since in the in-network computing algorithm, the self-information bit x_i and the local parity bit y_i is only calculated based on the in-edges of v_i, i.e., the edge set $\mathcal{E}_i^{\text{in}} = \{E_{ni} | 1 \leq n \leq N\}$, we obtain

$$\Pr[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top) \cup A_3^{(i)}(\mathbf{x}_0^\top) | \mathcal{G}] = \Pr[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top) \cup A_3^{(i)}(\mathbf{x}_0^\top) | E_{ni}, 1 \leq n \leq N]. \tag{F.18}$$

Thus

$$\begin{aligned} &\prod_{i=1}^{2N} \Pr[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top) \cup A_3^{(i)}(\mathbf{x}_0^\top) | \mathcal{G}] \\ =&\prod_{i=1}^{2N} \Pr[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top) \cup A_3^{(i)}(\mathbf{x}_0^\top) | E_{ni}, 1 \leq n \leq N]. \end{aligned} \tag{F.19}$$

Note a bidirectional edge in the current setting corresponds to two independently generated directional edges. Therefore

$$\begin{aligned} &P_e^{(N)}(\mathbf{x}) \\ =&\mathbb{E}_{\mathcal{G}}[P_e^{\mathcal{G}}(\mathbf{x})] \\ \leq& \sum_{\mathbf{x}_0^\top \in \{0,1\}^N \backslash \{\mathbf{0}^N\}} \mathbb{E}_{\mathcal{G}} \left[\prod_{i=1}^{2N} \Pr[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top) \cup A_3^{(i)}(\mathbf{x}_0^\top) | \mathcal{G}] \right] \\ \overset{(a)}{=}& \sum_{\mathbf{x}_0^\top \in \{0,1\}^N \backslash \{\mathbf{0}^N\}} \prod_{i=1}^{2N} \mathbb{E}_{\mathcal{G}} \left[\Pr[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top) \cup A_3^{(i)}(\mathbf{x}_0^\top) | E_{ni}, 1 \leq n \leq N] \right] \\ =& \sum_{\mathbf{x}_0^\top \in \{0,1\}^N \backslash \{\mathbf{0}^N\}} \prod_{i=1}^{2N} \Pr[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top) \cup A_3^{(i)}(\mathbf{x}_0^\top)], \end{aligned} \tag{F.20}$$

where the equality (a) follows from the fact that the sets $\{E_{ni}\}_{1 \leq n \leq N}$ and $\{E_{nj}\}_{1 \leq n \leq N}$ are independent (by the link generation hypothesis) for any pair (i, j) with $i \neq j$.

Computing the expected error upper bound using random graph theory

Lemma F.5.3. *Define k as the number of ones in $\mathbf{x}_0^\top$ and $\varepsilon_0 = (\frac{2}{1-1/e}+1)p_{ch}+\epsilon$, where ϵ is the erasure probability of the BECs and p_{ch} is a constant defined in* (8.13). *Further suppose $c\log N > 1$. Then, for $1 \leq i \leq N$, it holds that*

$$\prod_{i=1}^{N} \Pr[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top) \cup A_3^{(i)}(\mathbf{x}_0^\top)] = \epsilon^k. \tag{F.21}$$

For $N+1 \leq i \leq 2N$, it holds that

$$\Pr[A_1^{(i)}(\mathbf{x}_0^\top) \cup A_2^{(i)}(\mathbf{x}_0^\top) \cup A_3^{(i)}(\mathbf{x}_0^\top)] \leq \varepsilon_0 + (1-\varepsilon_0)\cdot\frac{1+(1-2p)^k}{2}, \tag{F.22}$$

where p is the connection probability defined in Assumption (A.4).

Proof. See Appendix F.4. □

Based on Lemma F.5.3 and simple counting arguments, note that (F.17) may be bounded as

$$P_e^{(N)}(\mathbf{x}) \leq \sum_{k=1}^{N}\binom{N}{k}\epsilon^k\left[\varepsilon_0 + (1-\varepsilon_0)\cdot\frac{1+(1-2p)^k}{2}\right]^N, \tag{F.23}$$

where the binomial expression $\binom{N}{k}$ is from the fact that there are $\binom{N}{k}$ codewords x_0 with k ones. Thus, we conclude the proof. □

By respectively analyzing the upper bound in Lemma F.5.2 for $k = o\left(\frac{N}{\log N}\right)$ and $k = \Omega\left(\frac{N}{\log N}\right)$, we can obtain the final result.

Combining two bounds together

We will prove that for any $\delta > 0$, it holds that

$$P_e^{(N)} \leq (1-b_\delta)^N + \delta e\epsilon\frac{N^{2-c(1-\varepsilon_0)(1-c\delta)}}{\log N}. \tag{F.24}$$

As shown in what follows, we bound the right hand side of (F.15) with two different methods for different k's. First, when k satisfies

$$1 \leq k < \delta\frac{N}{\log N}, \tag{F.25}$$

define

$$u = N(1-\varepsilon_0)\frac{1-(1-2p)^k}{2} \tag{F.26}$$

Then, based on the inequality

$$(1-\frac{1}{x})^x \leq e^{-1}, \forall x \in (0,1], \tag{F.27}$$

we have

$$[\varepsilon_0 + (1-\varepsilon_0)\frac{1+(1-2p)^k}{2}]^N = (1-\frac{u}{N})^N = [(1-\frac{u}{N})^{\frac{N}{u}}]^u \leq e^{-u}. \tag{F.28}$$

From the Taylor's expansion, we get

$$(1-2p)^k = 1 - 2pk + \frac{k(k-1)}{2}\theta^2, \theta \in [0, 2p].$$

By applying the equation above to (F.26), we get

$$u = N(1-\varepsilon_0)[kp - \frac{k(k-1)}{4}\theta^2].$$

Therefore, we have

$$\begin{aligned} e^{-u} =& e^{-k(1-\varepsilon_0)\cdot c\log N}\exp\{N(1-\varepsilon_0)\frac{k(k-1)}{4}\theta^2\} \\ \leq& \left(\frac{1}{N}\right)^{ck(1-\varepsilon_0)}\exp\{N(1-\varepsilon_0)\frac{k(k-1)}{4}\frac{4c^2\log^2 N}{N^2}\} \\ =& \left(\frac{1}{N}\right)^{ck(1-\varepsilon_0)} N^{(1-\varepsilon_0)\cdot\frac{c^2k(k-1)\log N}{N}}. \end{aligned}$$

Plugging the above inequality into (F.28), we get

$$\begin{aligned} &\binom{N}{k}\epsilon^k[\varepsilon_0 + (1-\varepsilon_0)\frac{1+(1-2p)^k}{2}]^N \\ \leq& \left(\frac{Ne}{k}\right)^k \epsilon^k \left(\frac{1}{N}\right)^{ck(1-\varepsilon_0)} N^{(1-\varepsilon_0)\cdot\frac{c^2k(k-1)\log N}{N}} \\ =& \left(\frac{e}{k}\epsilon N^{1-c(1-\varepsilon_0)[1-\frac{c(k-1)\log N}{N}]}\right)^k \\ <& \left(\frac{e}{k}\epsilon N^{1-c(1-\varepsilon_0)(1-c\delta)}\right)^k, \end{aligned} \tag{F.29}$$

where the last inequality follows from (F.25).

Second, when k satisfies

$$k > \delta\frac{N}{\log N}, \tag{F.30}$$

we can directly write

$$(1-2p)^k = [(1-2p)^{\frac{1}{2p}}]^{2pk} \leq e^{-2pk} < e^{-2c\delta}.$$

Therefore, it holds that

$$\begin{aligned}
&\sum_{k>\delta\frac{N}{\log N}} \binom{N}{k}\epsilon^k[\varepsilon_0+(1-\varepsilon_0)\frac{1+(1-2p)^k}{2}]^N \\
\leq &\sum_{k>\delta\frac{N}{\log N}} \binom{N}{k}\epsilon^k[\varepsilon_0+(1-\varepsilon_0)\frac{1+e^{-2c\delta}}{2}]^N \\
\leq &[\varepsilon_0+(1-\varepsilon_0)\frac{1+e^{-2c\delta}}{2}]^N \sum_{k=0}^{N}\binom{N}{k}\epsilon^k \\
= &[\varepsilon_0+(1-\varepsilon_0)\frac{1+e^{-2c\delta}}{2}]^N(1+\epsilon)^N \\
= &[(1-(1-\varepsilon_0)\frac{1-e^{-2c\delta}}{2})(1+\epsilon)]^N \\
\leq &\{1-[(1-\varepsilon_0)(1-\frac{1-e^{-2c\delta}}{2})-\epsilon]\}^N=\{1-(2b_\delta-\epsilon)\}^N.
\end{aligned}$$

When (8.21) holds, we have

$$\sum_{k>\delta\frac{N}{\log N}} \binom{N}{k}(\frac{p_{\text{ch}}}{c\log N})^k[\varepsilon_0+(1-\varepsilon_0)\frac{1+(1-2p)^k}{2}]^N<(1-b_\delta)^N. \tag{F.31}$$

Combining (F.15) and (F.29), we get

$$\begin{aligned}
P_e^{(N)} &\leq (1-b_\delta)^N + \\
&\qquad \sum_{k<\delta\frac{N}{\log N}} \binom{N}{k}\epsilon^k[\varepsilon_0+(1-\varepsilon_0)\frac{1+(1-2p)^k}{2}]^N \\
&\leq (1-b_\delta)^N + \sum_{k<\delta\frac{N}{\log N}} \left(\frac{e}{k}\epsilon N^{1-c(1-\varepsilon_0)(1-c\delta)}\right)^k \\
&\leq (1-b_\delta)^N + \delta\frac{N}{\log N}\frac{e}{k}\epsilon N^{1-c(1-\varepsilon_0)(1-c\delta)} \\
&\leq (1-b_\delta)^N + \delta e\epsilon\frac{N^{2-c(1-\varepsilon_0)(1-c\delta)}}{\log N}.
\end{aligned}$$

When $2<c(1-\varepsilon_0)(1-c\delta)$, the right hand side decreases polynomially with N.

F.6 Proof of Theorem 8.4.4

During the first step of the algorithm in Section 8.4.1, each self-information bit is broadcasted for t times. Therefore, for a node v_n, the total number of possibly erased versions

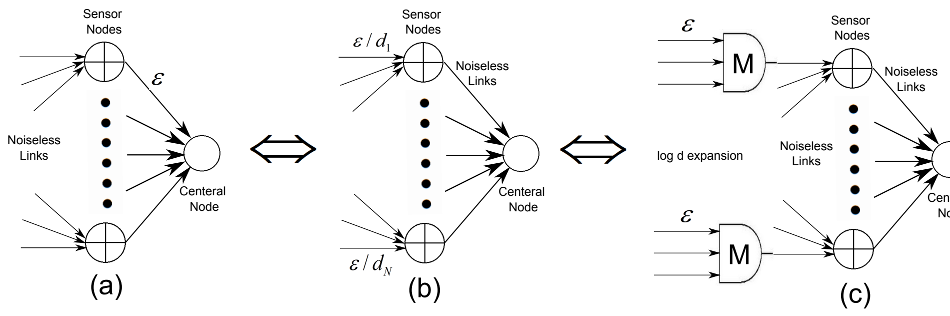


Figure F.1: Network transformations that relate coding theory to noisy broadcast networks.

of x_n is $d_n t$ where $d_n = \sum\limits_{m=1}^{N} \mathbf{1}_{\{v_n \in \mathcal{N}^-(v_m)\}}$. Each directed edge is counted once, so we have

$$\sum_{n=1}^{N} d_n = |\mathcal{E}|. \tag{F.32}$$

During the second step of the algorithm, each self-information bit x_n is transmitted to the sink once. For any x_n, the probability that all $d_n t + 1$ copies of x_n are erased is

$$p_{\mathrm{n}} = \epsilon^{d_n t + 1}. \tag{F.33}$$

If this event happens for any x_n, the identity function cannot be computed reliably, because at least all possible information about x_n has been erased. Thus, we have

$$P_e^{(N)} > 1 - \prod_{n=1}^{N} (1 - p_{\mathrm{n}}). \tag{F.34}$$

Based on $1 - x \leq \exp(-x)$ and the fact that arithmetic mean is no less than geometric mean, we have

$$\begin{aligned} 1 - P_e^{(N)} &< \prod_{n=1}^{N} (1 - p_{\mathrm{n}}) \leq \left[\frac{1}{N} \sum_{n=1}^{N} (1 - p_n) \right]^N \\ &= \left(1 - \frac{1}{N} \sum_{n=1}^{N} p_n \right)^N \leq \left(1 - \epsilon^{\frac{1}{N} \sum\limits_{n=1}^{N} d_n t + 1} \right)^N \leq \exp\left(-N \cdot \epsilon^{\frac{1}{N} \sum\limits_{n=1}^{N} d_n t + 1} \right), \end{aligned} \tag{F.35}$$

which can be translated into

$$\sum_{n=1}^{N} (t d_n + 1) \geq N \cdot \frac{\log N - \log\log(1/(1 - P_e^{(N)}))}{\log(1/\epsilon)}. \tag{F.36}$$

When $\lim\limits_{N \to \infty} P_e^{(N)} = 0$, it holds that $-\log\log(1/(1 - P_e^{(N)})) = \Theta(\log \frac{1}{P_e^{(N)}})$. Therefore, jointly considering (8.13), we get

$$|\mathcal{E}| = \sum_{n=1}^{N} d_n = \Omega\left(\frac{N \log(N / P_e^{(N)})}{\log(c \log N / p_{\mathrm{ch}})}\right). \tag{F.37}$$

Appendix G

Theoretical proofs for Chapter 9

G.1 Proof of Theorem 9.3.2

From Lemma 9.3.1, a check agent can determine the position and value of an incorrect message if the check agent is a single-ton. The main idea in the proof is to show that most of the check agents are single-tons. Suppose $\mathcal{V}_t$ satisfies (9.6). Then, we prove that $\mathcal{V}_{t+1}$ also satisfies (9.6). First, we examine the t-th decoding iteration under the assumption that all computations at the check agents and variable agents are fault-free.

Since $\mathcal{V}_t$ satisfies (9.6), from the expansion property, we have that

$$|\mathcal{N}(\mathcal{V}_t)| \geq \delta d_v |\mathcal{V}_t|. \tag{G.1}$$

Denote by $\mathcal{F}_t \subset \mathcal{V}_t$ the set of variable agents that fail to correct data after the t-th decoding iteration. First we upper-bound $|\mathcal{F}_t|$ under the assumption that all computations at the t-th decoding iteration are fault-free. Since all computations are fault-free, the correct variable agents will not become faulty after the decoding iteration.

Then, we upper-bound the size of $\mathcal{N}(\mathcal{V}_t)$ by respectively examining the neighborhood of $\mathcal{V}_t \setminus \mathcal{F}_t$ and $\mathcal{F}_t$. The neighborhood of $\mathcal{V}_t \setminus \mathcal{F}_t$ satisfies the trivial upper bound

$$|\mathcal{N}(\mathcal{V}_t \setminus \mathcal{F}_t)| \leq d_v |\mathcal{V}_t \setminus \mathcal{F}_t| = d_v(|\mathcal{V}_t| - |\mathcal{F}_t|). \tag{G.2}$$

Denote by E the number of edges connecting $\mathcal{F}_t$ and $\mathcal{N}(\mathcal{F}_t)$. We have that $E \leq d_v|\mathcal{F}_t|$. Since variable agents in $\mathcal{F}_t$ fail to correct the wrong data, the check agents in $\mathcal{N}(\mathcal{F}_t)$ must be multi-tons, and they cannot be connected to $\mathcal{V}_t \setminus \mathcal{F}_t$. Therefore, $E \geq 2|\mathcal{N}(\mathcal{F}_t)|$. Thus, we have

$$|\mathcal{N}(\mathcal{F}_t)| \leq E/2 \leq \frac{d_v}{2}|\mathcal{F}_t|. \tag{G.3}$$

Combining (G.2) and (G.3), we have

$$|\mathcal{N}(\mathcal{V}_t)| = |\mathcal{N}(\mathcal{V}_t \setminus \mathcal{F}_t)| + |\mathcal{N}(\mathcal{F}_t)| \leq d_v(|\mathcal{V}_t| - |\mathcal{F}_t|) + \frac{d_v}{2}|\mathcal{F}_t| = d_v|\mathcal{V}_t| - \frac{d_v}{2}|\mathcal{F}_t|. \tag{G.4}$$

From (G.1) and (G.4), we have

$$\delta d_v |\mathcal{V}_t| \leq d_v|\mathcal{V}_t| - \frac{d_v}{2}|\mathcal{F}_t|, \tag{G.5}$$

which is equivalent to

$$|\mathcal{F}_t| \leq 2(1-\delta)|\mathcal{V}_t|. \tag{G.6}$$

Now, we consider the case when computations can also be faulty. A single fault at a check agent can at most corrupt the data at one variable agent. Similarly, a single fault at a variable agent can only corrupt the data at the same variable agent. Therefore, when computations are also faulty,

$$|\mathcal{F}_t| \leq 2(1-\delta)|\mathcal{V}_t| + e_t^{\text{var}} + e_t^{\text{chk}}. \tag{G.7}$$

The number of variable agents that store incorrect data before the $(t+1)$-th decoding iteration satisfies

$$|\mathcal{V}_{t+1}| \leq |\mathcal{F}_t| + e_t^{\text{sto}}. \tag{G.8}$$

Therefore,

$$|\mathcal{V}_{t+1}| \leq 2(1-\delta)|\mathcal{V}_t| + e_t^{\text{var}} + e_t^{\text{chk}} + e_t^{\text{sto}} \leq 2(1-\delta)\alpha N + (2\delta-1)\alpha N = \alpha N. \tag{G.9}$$

G.2 Proof of Theorem 9.4.1

The proof for the first part (i.e., the proof of (9.14)) can be derived similarly as in the proof of Theorem 9.3.2. In fact, we can obtain the same bound (G.6) under the assumption that all computations in the t-th iteration are fault-free. When computations are fault-prone, each of the e_t^{loc} computation faults in computing $\mathbf{w}_t^\top \mathbf{X}_i$ may result in the failure of detecting at most d_v storage faults, because all connected parity check agents may change from single-tons to multi-tons due to this computation failure; a computation fault in the parity check computation may result in the failure of correcting at most one incorrect variable agent. Therefore,

$$|\mathcal{V}_{t+1}| \leq 2(1-\delta)|\mathcal{V}_t| + d_v e_t^{\text{loc}} + e_t^{\text{chk}} + e_t^{\text{sto}} \leq 2(1-\delta)\alpha N + (2\delta-1)\alpha N = \alpha N. \tag{G.10}$$

Then, we look at the effect of different types of faults on the computation of the local gradient update $\nabla \mathbf{w}_t^{(i)} = \mathbf{X}_i \left[\mathbf{y}_i - \sigma(\mathbf{X}_i^\top \mathbf{w}_t)\right]$. When a first-type fault (storage fault) happens, the stored data changes from $\mathbf{X}_i$ to $\mathbf{X}_i + \mathbf{N}_i$; the second-type fault (computation of $\mathbf{X}_i^\top \mathbf{w}_t$), the third-type fault (solving (9.3)) and the fourth-type fault (computation of the majority) all change $\mathbf{X}_i^\top \mathbf{w}_t$ to $\mathbf{X}_i^\top \mathbf{w}_t + \mathbf{n}_i$, and the effect of these three types of faults do not accumulate, because the third-type fault masks the second-type fault and the fourth-type fault masks the third-type fault; the fifth-type fault (final computation of the local gradient) changes the final local gradient from $\mathbf{X}_i \left[\mathbf{y}_i - \sigma(\mathbf{X}_i^\top \mathbf{w}_t)\right]$ to $\mathbf{X}_i \left[\mathbf{y}_i - \sigma(\mathbf{X}_i^\top \mathbf{w}_t)\right] + \widetilde{\mathbf{n}}_i$. The overall effect of all five types of errors is to change the local gradient from $\mathbf{X}_i \left[\mathbf{y}_i - \sigma(\mathbf{X}_i^\top \mathbf{w}_t)\right]$ to

$$\nabla \mathbf{w}_{t,\text{f}}^{(i)} = (\mathbf{X}_i + \mathbf{N}_i) \left[\mathbf{y}_i - \sigma\left((\mathbf{X}_i + \mathbf{N}_i)^\top \mathbf{w}_t + \mathbf{n}_i\right)\right] + \widetilde{\mathbf{n}}_i. \tag{G.11}$$

The change in the local gradient computed at the i-th variable agent is

$$\begin{aligned}\mathbf{e}_t^{(i)} &= \nabla \mathbf{w}_{t,\mathrm{f}}^{(i)} - \nabla \mathbf{w}_t^{(i)} \\ &= (\mathbf{X}_i + \mathbf{N}_i)\left[\mathbf{y}_i - \sigma\left((\mathbf{X}_i + \mathbf{N}_i)^\top \mathbf{w}_t + \mathbf{n}_i\right)\right] + \widetilde{\mathbf{n}}_i - \mathbf{X}_i\left[\mathbf{y}_i - \sigma(\mathbf{X}_i^\top \mathbf{w}_t)\right] \\ &= -\mathbf{X}_i\left(\sigma\left((\mathbf{X}_i + \mathbf{N}_i)^\top \mathbf{w}_t + \mathbf{n}_i\right) - \sigma(\mathbf{X}_i^\top \mathbf{w}_t)\right) + \mathbf{N}_i\left[\mathbf{y}_i - \sigma\left((\mathbf{X}_i + \mathbf{N}_i)^\top \mathbf{w}_t + \mathbf{n}_i\right)\right] + \widetilde{\mathbf{n}}_i.\end{aligned} \tag{G.12}$$

Define $\widetilde{\sigma}(x) = \frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$. When $\widetilde{\sigma}(x)$ is applied to a vector, the function $\sigma(x)(1-\sigma(x))$ is applied to each entry of the vector. Then, using Cauchy's mean-value theorem [301, Section 10.4], we have

$$\sigma\left((\mathbf{X}_i + \mathbf{N}_i)^\top \mathbf{w}_t + \mathbf{n}_i\right) - \sigma(\mathbf{X}_i^\top \mathbf{w}_t) = \mathbf{D} \cdot (\mathbf{N}_i^\top \mathbf{w}_t + \mathbf{n}_i), \tag{G.13}$$

where

$$\mathbf{D} = \mathrm{diag}\left[\widetilde{\sigma}\left(\theta\left((\mathbf{X}_i + \mathbf{N}_i)^\top \mathbf{w}_t + \mathbf{n}_i\right) + (1-\theta)\mathbf{X}_i^\top \mathbf{w}_t\right)\right], \tag{G.14}$$

for some $\theta \in (0,1)$. Since $\widetilde{\sigma}(x) = \sigma(x)(1-\sigma(x)) \le \frac{1}{4}\ \forall x \in \mathbb{R}$, we have

$$\left\|\mathbf{D}\left(\mathbf{N}_i^\top \mathbf{w}_t + \mathbf{n}_i\right)\right\| \le \frac{1}{4}\left\|\mathbf{N}_i^\top \mathbf{w}_t + \mathbf{n}_i\right\| \le \frac{1}{4}(L_\phi \|\mathbf{N}_i\| + \|\mathbf{n}_i\|). \tag{G.15}$$

Therefore, the change in the local gradient satisfies

$$\begin{aligned}\left\|\mathbf{e}_t^{(i)}\right\| \le& \left\|\mathbf{X}_i\left(\sigma\left((\mathbf{X}_i + \mathbf{N}_i)^\top \mathbf{w}_t + \mathbf{n}_i\right) - \sigma(\mathbf{X}_i^\top \mathbf{w}_t)\right)\right\| \\ &+ \left\|\mathbf{N}_i\left[\mathbf{y}_i - \sigma\left((\mathbf{X}_i + \mathbf{N}_i)^\top \mathbf{w}_t + \mathbf{n}_i\right)\right]\right\| + \|\widetilde{\mathbf{n}}_i\| \\ \le& \|\mathbf{X}_i\| \cdot \frac{1}{4}((L_\phi + L_u)\|\mathbf{N}_i\| + \|\mathbf{n}_i\|) + \|\mathbf{N}_i\| \cdot 2k + \|\widetilde{\mathbf{n}}_i\| \\ \le& (\frac{1}{4}\sqrt{k}(L_\phi + L_u)L_x + 2k)\|\mathbf{N}_i\| + \frac{1}{4}\sqrt{k}L_x\|\mathbf{n}_i\| + \|\widetilde{\mathbf{n}}_i\|,\end{aligned} \tag{G.16}$$

where the induced matrix norm is the Frobenius norm. Note that the derivation above accounts for the accumulative effect of different types of faults on the output at a single local gradient. If a fault does not happen at the i-th variable agent, the fault equals to zero. Define $\mathbf{e}_t = \nabla L(\mathbf{w}_{t,\mathrm{f}}) - \nabla L(\mathbf{w}_t)$ as the overall computation error of the gradient. Using the triangle inequality, we have

$$\begin{aligned}\|\mathbf{e}_t\| \le& \sum_{i=1}^{L}\left\|\mathbf{e}_t^{(i)}\right\| \le \sum_{i=1}^{L}(\frac{1}{4}\sqrt{k}(L_\phi + L_u)L_x + 2k)\|\mathbf{N}_i\| + \frac{1}{4}\sqrt{k}L_x\|\mathbf{n}_i\| + \|\widetilde{\mathbf{n}}_i\| \\ \le& \left(\frac{1}{4}\sqrt{k}L_x(L_\phi + L_u) + 2k\right)\sum_{i=1}^{L}\|\mathbf{N}_i\| + \frac{1}{4}\sqrt{k}L_x\sum_{i=1}^{L}\|\mathbf{n}_i\| + \sum_{i=1}^{L}\|\widetilde{\mathbf{n}}_i\|.\end{aligned} \tag{G.17}$$

From the bound on the number of storage faults (9.14), the bound on the number of computation faults (9.15) at each iteration and the bound on the ℓ_2-norm of the faults (see Assumption 15), we have

$$\sum_{i=1}^{L}\|\mathbf{N}_i\| \le \alpha N L_e, \tag{G.18}$$

and

$$\sum_{i=1}^{L} \|\mathbf{n}_i\| + \sum_{i=1}^{L} \|\widetilde{\mathbf{n}}_i\| \leq \beta N L_e. \tag{G.19}$$

Therefore, we have

$$\|\mathbf{e}_t\| \leq \left(\frac{1}{4}\sqrt{k}L_x(L_\phi + L_u) + 2k\right) \cdot \alpha N L_e + \max\left(\frac{1}{4}\sqrt{k}L_x, 1\right) \cdot \beta N L_e = \eta N, \tag{G.20}$$

where $\eta := \left(\frac{1}{4}\sqrt{k}L_x(L_\phi + L_u) + 2k\right) \cdot \alpha L_e + \max\left(\frac{1}{4}\sqrt{k}L_x, 1\right) \cdot \beta L_e$. The update at the central controller satisfies

$$\mathbf{w}_{t+1} = (1 - 2\varepsilon\lambda)\mathbf{w}_t - \varepsilon\nabla L(\mathbf{w}_{t,\mathrm{f}}) = (1 - 2\varepsilon\lambda)\mathbf{w}_t - \varepsilon(\nabla L(\mathbf{w}_t) + \mathbf{e}_t). \tag{G.21}$$

The desired update is

$$\mathbf{u}_{t+1} = (1 - 2\varepsilon\lambda)\mathbf{u}_t - \varepsilon\nabla L(\mathbf{u}_t). \tag{G.22}$$

Suppose the initial value of these two systems are the same. Define the difference between the two systems as $\boldsymbol{\phi}_t = \mathbf{w}_t - \mathbf{u}_t$, then,

$$\begin{aligned} \boldsymbol{\phi}_{t+1} &= (1 - 2\varepsilon\lambda)\mathbf{w}_t - \varepsilon(\nabla L(\mathbf{w}_t) + \mathbf{e}_t) - (1 - 2\varepsilon\lambda)\mathbf{u}_t + \varepsilon\nabla L(\mathbf{u}_t) \\ &= (1 - 2\varepsilon\lambda)\boldsymbol{\phi}_t - \varepsilon\mathbf{e}_t - \varepsilon\mathbf{H}\left(\mathbf{w}_t^\theta\right)\boldsymbol{\phi}_t, \\ &= \left((1 - 2\varepsilon\lambda)\mathbf{I} - \varepsilon\mathbf{H}\left(\mathbf{w}_t^\theta\right)\right)\boldsymbol{\phi}_t - \varepsilon\mathbf{e}_t. \end{aligned} \tag{G.23}$$

where $\mathbf{w}_t^\theta = \theta\mathbf{w}_t + (1-\theta)\mathbf{u}_t$ and the constant $\theta \in (0, 1)$ is obtained using Cauchy's mean-value theorem

$$\begin{aligned} \nabla L(\mathbf{w}_t) - \nabla L(\mathbf{u}_t) &= \mathbf{H}\left(\theta\mathbf{w}_t + (1-\theta)\mathbf{u}_t\right)(\mathbf{w}_t - \mathbf{u}_t) \\ &= \mathbf{H}\left(\theta\mathbf{w}_t + (1-\theta)\mathbf{u}_t\right)\boldsymbol{\phi}_t, \end{aligned} \tag{G.24}$$

and the Hessian $\mathbf{H}(\mathbf{w})$ is

$$\mathbf{H}(\mathbf{w}_t^\theta) = \widetilde{\mathbf{X}}\mathbf{R}(\mathbf{w}_t^\theta)\widetilde{\mathbf{X}}^\top = \sum_{i=1}^{L}\sum_{j=1}^{K} \phi_{i,j}(1 - \phi_{i,j})\mathbf{x}_{i,j}\mathbf{x}_{i,j}^\top, \tag{G.25}$$

where

$$\mathbf{R}(\mathbf{w}_t^\theta) = \mathrm{Diag}\{\phi_{1,1}(1 - \phi_{1,1}), \phi_{1,2}(1 - \phi_{1,2}), \ldots, \phi_{L,k}(1 - \phi_{L,k})\}, \tag{G.26}$$

and

$$\phi_{i,j}(\mathbf{w}_t^\theta) = \sigma((\mathbf{w}_t^\theta)^\top\mathbf{x}_{i,j}), 1 \leq i \leq L, 1 \leq j \leq k. \tag{G.27}$$

Now we prove that the error $\boldsymbol{\phi}_t$ satisfies

$$\|\boldsymbol{\phi}_t\| \leq L_\phi. \tag{G.28}$$

First, we can see that

$$\|\boldsymbol{\phi}_0\| = 0, \tag{G.29}$$

which satisfies (G.28) for $t = 0$. Suppose at time t, (G.28) is satisfied.

Now, for the contraction linear system (G.23),

$$\begin{aligned} \|\boldsymbol{\phi}_{t+1}\| &\leq \left\|(1-2\varepsilon\lambda)\mathbf{I} - \varepsilon\mathbf{H}\left(\mathbf{w}_t^\theta\right)\right\| \cdot \|\boldsymbol{\phi}_t\| + \varepsilon\|\mathbf{e}_t\| \\ &\leq (1-2\varepsilon\lambda)\|\boldsymbol{\phi}_t\| + \varepsilon\|\mathbf{e}_t\| \\ &\leq (1-2\varepsilon\lambda)L_\phi + \varepsilon\eta N \\ &\leq (1-2\varepsilon\lambda)L_\phi + 2\epsilon\lambda L_\phi = L_\phi. \end{aligned} \tag{G.30}$$