5-2018

Signal Processing for Caching Networks and Non-volatile Memories

Tianqiong Luo
Purdue University

SIGNAL PROCESSING FOR CACHING NETWORKS AND NON-VOLATILE MEMORIES

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Tianqiong Luo

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2018

Purdue University

West Lafayette,Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Borja M. Peleato-Inarrea, Chair
Department of Electrical and Computer Engineering

Dr. Chih-Chun Wang
Department of Electrical and Computer Engineering

Dr. David J. Love
Department of Electrical and Computer Engineering

Dr. Vijay Raghunathan
Department of Electrical and Computer Engineering

Approved by:

Dr. Venkataramanan Balakrishnan
Head of the Graduate Program

ACKNOWLEDGMENTS

I would like to first thank my advisor, Professor Borja Peleato, for his support and guidance during my Ph.D. studies. I will never forget the encouragement and help from him, especially every time he patiently gave me advices on how to continue my research work, improve my writing skills and proceed with my future career. These suggestions are really helpful and beneficial to my future work.

I had two wonderful internships in 2017, one with Google and one with Facebook. I determined my mind to continue my future career as a software engineer after these two internships. I want to extend my deep gratitude to my supervisors, Kai Shen and Xiao Jing. They have provided me with guidance on how to do system research work in industry.

I want to thank my parents who have always encouraged me in my Ph.D. studies. Finally, I want to thank my boyfriend Yimajian Yan, who supported me when I went through hard times.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Luo, Tianqiong Ph.D., Purdue University, May 2018. Signal Processing for Caching Networks and Non-volatile Memories. Major Professor: Borja Peleato.

The recent information explosion has created a pressing need for faster and more reliable data storage and transmission schemes. This thesis focuses on two systems: caching networks and non-volatile storage systems. It proposes network protocols to improve the efficiency of information delivery and signal processing schemes to reduce errors at the physical layer as well.

This thesis first investigates caching and delivery strategies for content delivery networks. Caching has been investigated as a useful technique to reduce the network burden by prefetching some contents during off-peak hours. Coded caching [1] proposed by Maddah-Ali and Niesen is the foundation of our algorithms and it has been shown to be a useful technique which can reduce peak traffic rates by encoding transmissions so that different users can extract different information from the same packet. Content delivery networks store information distributed across multiple servers, so as to balance the load and avoid unrecoverable losses in case of node or disk failures. On one hand, distributed storage limits the capability of combining content from different servers into a single message, causing performance losses in coded caching schemes. But, on the other hand, the inherent redundancy existing in distributed storage systems can be used to improve the performance of those schemes through parallelism. This thesis proposes a scheme combining distributed storage of the content in multiple servers and an efficient coded caching algorithm for delivery to the users. This scheme is shown to reduce the peak transmission rate below that of state-of-the-art algorithms.

Then we study the trade-off between the network traffic load and disk I/O for caching networks. Coded caching can reduce traffic load by broadcasting coded messages that can benefit multiple users but, in the case with redundant requests, it requires reading some data segments multiple times to compose different coded messages. Hence, coded caching requires more disk I/Os than uncoded transmission. This thesis proposes caching and delivery algorithms which combine coded and uncoded transmission to strike a trade-off between traffic load and disk I/Os. Our algorithms can improve both the average and worst case performance in terms of the user requests.

Finally, we broaden our perspective to look at the storage hardware. Two methods are proposed which are suitable for NAND flash technology: multi-page read and spreading modulation. The first one reads multiple wordlines simultaneously and returns a combination of their stored information. This multi-page read method is shown to be useful for equalizing the inter-cell interference, reduce the damage caused by erase operations, and speed up the decoding of some codes, such as WOM codes [2]. Then a new data representation scheme is proposed which increases endurance and significantly reduces the probability of error caused by inter-cell-interference. This data representation scheme is based on using an orthogonal code to spread each bit across multiple cells, resulting in lower variance for the voltages being programmed. We also study an up-and-coming memory technology, ReRAM, with a different set of challenges. Specifically, we build a simple analytic model for the voltage drop and sneak currents in MLC-ReRAM arrays as a form of inter-cell-interference and proposes two techniques to minimize the resulting BER: distribution shaping and spreading modulation, which is extended from that of NAND flash.

1. INTRODUCTION

For several decades, CPUs have doubled their speed every two years in what is commonly known as Moore's law, but the storage technology has not been able to keep up with this trend: magnetic hard drives have steadily increased their capacity, but not their speed. Current computers and communication networks are not limited by the speed at which information can be processed, but rather by the speed at which it can be read, moved, and written. Furthermore, the recent information explosion is driving an exponential increase in the demand for data, which is not expected to slow down any time soon. Users and applications require storing large amounts of data and transmitting data at higher speeds, straining the devices and networks to their maximum capabilities. This thesis focuses on two modern storage systems: caching networks and non-volatile memories, as explained below.

The first part of the thesis focuses on caching networks. In the context of networking, popular content can be pre-cached in multiple nodes to balance the load and alleviate the stress of the network during peak times. So the cache problem focuses on what data to store and how to deliver it so that the system efficiency is improved. Besides the heavy network transfer load, large number of disk I/Os is another performance bottleneck for storage systems, putting a burden on the system resources.

The second part of the thesis focuses on storage hardware, specifically non-volatile memories. Non-volatile memories store data using persistent physical properties, which do not change even if the power is turned off. Specifically, Flash memories use the voltage threshold of floating gate transistors and ReRAM memories use memristors (a contraction of "memory resistor"). Unfortunately, these parameters are subject to noise during reads and writes, making it a signal processing challenge to store data reliably.

Section 1.1 and Section 1.2 will introduce the challenges that caching networks and non-volatile memory technologies are facing and describe the contributions of this thesis.

1.1 Challenges of Caching Networks

Caching has been investigated as a useful technique to reduce the network burden by prefetching some contents during off-peak hours. A caching scheme has two phases: placement and delivery. In the placement phase, the users have access to all files to fill their caches. In the delivery phase, every user requests one file and only the server has database access. The server delivers messages to the users to fulfill their requests. Coded caching has recently become quite popular among the coding community, starting with the work by Maddah-Ali and Niesen in [1]. It has been shown that coded caching can reduce peak traffic rates by encoding transmissions so that different users can extract different information from the same packet. Our study is based on Maddah-Ali and Niesen's work in [1] and we extend their work to solve two interesting problems: extending coded caching to distributed storage system (as explained in Subsection 1.1.1) AND achieving the traffic load-I/O trade-off for coded caching (as explained in Subsection 1.1.2).

1.1.1 Coded Caching with Distributed Storage

Maddah-Ali and Niesen's work [1] focuses on a system with multiple users connecting to a single server through a shared broadcast link. However, with the higher demand of data, networks usually distribute popular files across multiple independent servers. This thesis proposes and analyzes multiple caching mechanisms for multi-server systems with different system parameters.

Distributed storage deals with how the information is stored at the servers. Disk failures are very common in large storage systems, so they need to have some amount of redundancy. Erasure codes have recently sparked a renewed interest from the

research community for this task. Files are encoded and distributed among a set of nodes (disks, servers, etc.) in such a way that the system can recover from the failure of a certain number of nodes [3]. Most large scale systems use some form of erasure codes (such as RAID [4]) with striping across multiple storage drives, but some others store or replicate whole files as a single unit in the network nodes (*e.g.* data centers). This increases the peak rate, but it also simplifies book-keeping and deduplication, improves security, and makes the network more flexible.

In this thesis, we aim to design a joint storage and transmission protocol for the multi-server multi-user system. We combine distributed storage with coded caching utilizing parallelism and redundancy to reduce the peak traffic rate. The main contributions are: (1) a flexible model for multi-server systems where each file can be divided among multiple servers or kept as a single block in one server; (2) an extension of the coded caching algorithms in [1] and [5] to striping multi-server systems; (3) new caching and delivery schemes with significantly lower peak rates for the case when files are stored as a single unit in a data server. The detailed algorithms are elaborated in Chapter 2 and the related publication is [6].

1.1.2 Traffic Load-I/O Trade-off for Coded Caching

Although the traffic load is the dominant factor in slow or congested networks, disk I/Os are also valuable and can become the bottleneck in some systems, such as Haystack [7] and Colossus [8]. A significant amount of research has gone into coding techniques to minimize disk I/Os in storage systems [9, 10].

Maddah-Ali and Nisen's coded caching scheme in [1] has the lowest peak traffic load in the literature and its extension by Yu et. al. [11] is proved to achieve the best average traffic load with uncoded prefetching. However, their I/O performance is suboptimal when there are redundant user demands. The same segment could be read multiple times if it is used to construct different messages, which dramatically increases I/O reads. On the contrast, if all messages are transmitted uncoded, each

data segment requested is read once and broadcast to all users. Inspired by this fact, we study the trade-off between traffic load and I/O by designing algorithms which combine coded and uncoded transmission. The algorithms are shown to improve both the average and worst case performance in terms of the user requests as elaborated in Chapter 3 and the related publication is [12].

1.2 Challenges of Modern Non-volatile Memory Technologies

Recently, some new non-volatile memory technologies have emerged as a faster and more efficient alternative to hard drives. We will investigate two promising non-volatile memory technologies in this thesis: NAND flash memories and Resistive RAM. These two new non-volatile memory technologies offer significantly higher speeds and power efficiency than hard drives, but their higher cost is still an obstacle for its widespread use. The cost is dominated by the area of silicon that they require per Gigabyte of stored information. Manufacturers have tried to increase the capacity of the memories by shrinking the cells and storing more bits in each of them but this has introduced some problems, mainly related to reliability and endurance. The challenges for NAND flash and ReRAM are explained in Subsection 1.2.1 and Subsection 1.2.2 respectively.

1.2.1 Challenges of NAND Flash

A NAND flash memory is fundamentally an array of floating gate transistors, known as flash cells, whose threshold voltages can be programmed to represent different information symbols. Single-Level Cell (SLC) memories can only store one bit in each cell, but most commercial products now use Multi-Level Cell (MLC) memories that can store two bits in each cell by taking 4 possible voltages. In order to reduce the cost, manufacturers have aggressively scaled the technology to pack more cells in the same silicon real estate, while they also increased the number of bits stored in each cell.

As the cells shrink, the noise observed in the programmed voltages increases, specially the inter-cell interference (ICI). ICI is a phenomenon by which programming a cell increases the voltage of its neighbors. It has been shown that the ICI noise created by a cell is proportional to the voltage to which it is programmed [13]. Another challenge that the flash memory industry is facing is the limited lifetime of flash cells. Basic operations like programming and erasing[1] the cells require tunneling charges through a dielectric barrier. This results in stresses that degrade the properties of the barrier making the cells less efficient in the retention of data, more vulnerable to noise, and consequently more prone to errors. Once again, the degradation is proportional to the voltage variations [14].

In this thesis, two signal processing approaches are proposed for NAND flash to improve the reliability and endurance: (1) We propose a new method which reads multiple wordlines simultaneously and returns a combination of their stored information. This multi-page read method is shown to be useful for equalizing the ICI, reduce the damage caused by erase operations, and speed up the decoding of certain WOM codes [2]. (2) A spreading modulation is proposed. It is a new data representation scheme which reduces ICI and extends the lifetime of the memory by reducing the frequency with which the largest voltage levels are programmed. The proposed modulation is based on using an orthogonal code to spread each information symbol across multiple cells, similar to how DS-CDMA is used in wireless communications [15, 16]. The detailed algorithms are elaborated in Chapter 4 and the related publications are [17–19].

1.2.2 Challenges of Resistive RAM

Besides NAND flash memory technology which is already in commercial use, some other promising non-volatile memory technologies are also under research. Resistive RAM (ReRAM) is rising as a promising non-volatile alternative because of its high

[1]Flash cells need to be erased before they can be overwritten.

density, fast access time and low power consumption. ReRAM uses memristors (a contraction of "memory resistor") to store information. A memristor is a nonlinear resistor whose value can be adjusted by pushing current across its terminals.

To increase the density of the memristor array, crosspoint architectures are used, which shows excellent scalability but suffers from other problems, mainly related to sneak currents flowing through supposedly deactivated cells. Typically, writes and reads are done by fully biasing a selected wordline and bitline (row and column, respectively) while others are only partially biased or not at all. This creates a large voltage drop across the cell at the intersection, a smaller one across other cells in the same wordline or bitline (half-selected cells), and a negligible one across the rest of the cells. Ideally, all the current would be flowing through the selected cell, but in practice there are some additional currents flowing through the other half-selected cells. These are called sneak currents [20]. The magnitude of the sneak currents increases dramatically with the size of the memristor array, to the point that they have become the limiting factor in the scalability of ReRAM memories. This effect is even more severe in MLC memories.

In this thesis, we propose signal processing approaches to compensate for sneak currents in MLC ReRAM. Our contributions include: (1) we build a simple analytic model for the voltage drop and sneak currents in MLC-ReRAM arrays as a form of ICI. (2) we propose two techniques to minimize the resulting BER: spreading modulation and distribution shaping. The detailed algorithms are shown in Chapter 5 and the related publication is [21].

2. CODED CACHING AND DISTRIBUTED STORAGE

2.1 Introduction

The recent information explosion has created a pressing need for faster and more reliable data transmission and recovery schemes. The IT industry has addressed this challenge through parallelism and caching: instead of using a single high capacity storage drive to serve all the requests, networks usually distribute popular files across multiple independent servers that can operate in parallel and cache part of the information at intermediate or final nodes. This chapter proposes multiple caching mechanisms for multi-server systems with different system parameters. Previous literature has addressed coded caching for single server systems and distributed storage without caching but, to the extent of our knowledge, this is the first work that considers both coded caching at the users and distributed storage at the servers. Furthermore, it provides solutions for systems with and without file striping (*i.e.* with files split among multiple servers and with whole files stored in each server).

Erasure codes are adopted to solve the disk failures in distributed storage system. Files are encoded and distributed among servers in such a way that the system can recover from the failure of a certain number of servers [3]. One widely used distributed storage technique based on erasure codes is RAID (redundant array of independent disks). It combines multiple storage nodes (disks, servers, etc.) into a single logical unit with data redundancy. Two of the most common are RAID-4 and RAID-6, consisting of block-level striping with one and two dedicated parity nodes, respectively [4, 22]. Most large scale systems use some form of RAID with striping across multiple storage drives, but some others store or replicate whole files as a single unit in the network nodes (*e.g.* data centers). This increases the peak rate, but it also

simplifies book-keeping and deduplication, improves security, and makes the network more flexible.

Coded caching deals with the high temporal variability of network traffic: the peak traffic in the network is reduced by pre-fetching popular content in each receiver's local cache memory during off-peak hours, when resources are abundant. Coded caching has also recently become a research hit, starting with the work by Maddah-Ali and Niesen in [1], which focused on how a set of users with local memories can efficiently receive data from a single server through a common link. Their seminal paper proposed a caching and delivery scheme offering a worst case performance within a constant factor of the information-theoretic optimum, as well as upper and lower bounds on that optimum. The lower bounds were later refined in [23] and new schemes were designed to consider non-uniform file sizes and popularity [24, 25]; multiple requests per user [26, 27]; variable number of users [28]; and multiple servers with access to the whole library of files [29].

Maddah-Ali and Niesen's work in [1] caches the information uncoded and encodes the transmitted packets. This scheme performs well when the cache size is relatively large, but a close inspection shows that there are other cases in which its performance is far from optimal. Tian and Chen's recent work in [5] designs a new algorithm which encodes both the cached and transmitted segments to achieve a better performance than [1] when the cache size is small or the number of users is greater than the number of files. However, this scheme also focuses on a single server system. In this chapter, we aim to design a joint storage and transmission protocol for the multi-server multi-user system.

Summarizing, prior work on distributed storage has studied how a single user can efficiently recover data distributed across a set of nodes and prior work on coded caching has studied how a set of users with local memories can efficiently receive data from a single node. However, to the extent of our knowledge, it has not been studied how the cache placement and content delivery should be performed when multiple nodes send data to multiple users through independent channels. We combine dis-

tributed storage with coded caching utilizing parallelism and redundancy to reduce the peak traffic rate in this thesis.

The rest of the chapter is structured as follows: Section 2.2 introduces the system model and two existing coded caching algorithms for single server systems, namely the one proposed by Maddah-Ali and Niesen in [1] and the interference elimination scheme in [5]. Section 2.3 extends both algorithms to a multi-server system with file striping, while Sections 2.4 and 2.5 consider the case where servers store whole files. Specifically, Section 2.4 extends Maddah-Ali and Niesen's scheme, suitable for systems with large cache capacity, and Section 2.5 extends the interference elimination scheme, which provides better performance when the cache size is small. Finally, Section 2.6 provides simulations to support and illustrate our algorithms and Section 2.7 concludes the chapter.

2.2 Background

This section describes the multi-node multi-server model in 2.2.1 and then reviews two existing coded caching schemes that constitute the basis for our algorithms. Subsection 2.2.2 summarizes Maddah-Ali and Niesen's coded caching scheme from [1] and subsection 2.2.3 summarizes Tian and Chen's interference elimination scheme from [5].

2.2.1 System Model

We consider a network with K users[1] and N files stored in L data servers. Some parts of the chapter will also include additional parity servers, denoted parity server P when storing the bitwise XOR of the information in the data servers (RAID-4) and parity server Q when storing a different linear combination of the data (RAID-6). The network is assumed to be flexible, in the sense that there is a path from every

[1]Servers and users can be anything from a single disk to a computer cluster, depending on the application.

server to every user [29]. Each server stores the same number of files with the same size and each user has a cache with capacity for M files. For the sake of simplicity, this chapter assumes that all files have identical length and popularity.

The servers are assumed to operate on independent error-free channels, so that two or more servers can transmit messages simultaneously and without interference to the same or different users. A server can broadcast the same message to multiple users without additional cost in terms of bandwidth, but users cannot share the content of their caches with each other. This assumption makes sense in a practical setting since peer-to-peer content sharing is generally illegal. Also, users typically have an asymmetric channel, with large download capacity but limited upload speed.

Similarly, each server can only access the files that it is storing, not those stored on other servers. A server can read multiple segments from its own files and combine them into a single message, but two files stored on different servers cannot be combined into a single message. However, it will be assumed that servers are aware of the content cached by each user and of the content stored in other servers, so that they can coordinate their messages. This can be achieved by exchanging segment IDs through a separate low-capacity control channel or by maintaining a centralized log.

The problem consists of two phases: placement and delivery. During the placement phase, the content is stored in the user's caches. The decisions on where to locate each file, how to compute the parity, and what data to store in each cache are made based on the statistics for each file's popularity, without knowledge of the actual user requests. In our chapter, we assume all the files have the same popularity. The delivery phase starts with each user requesting one of the files. All servers are made aware of these requests and proceed to send the necessary messages.

Throughout the chapter, we use subindices to represent file indices and superindices to represent segment indices, so F_i^j will represent the j-th segment from file F_i. Some parts of the chapter will also use different letters to represent files from different servers. For example, A_i to represent the i-th file from server A and A_i^j to represent the j-th segment from file A_i. The chapter focuses on minimizing the

peak rate (or delay), implicitly assuming that different users request different files. Therefore, we will indistinctly refer to users or their requests.

2.2.2 Maddah-Ali and Niesen's scheme

The coded caching scheme proposed by Maddah-Ali and Niesen in [1] has a single server storing all the files $\{F_1, F_2 \ldots, F_N\}$, and users are connected to this server through a shared broadcast link. Their goal is to design caching and delivery schemes so as to minimize the peak load on the link, *i.e.* the total amount of information transferred from the server to the users. This scheme splits each file F_i into $\binom{K}{t}$ nonoverlapping segments F_i^j of equal size, $j = 1, \ldots \binom{K}{t}$, with $t = \frac{KM}{N}$, and caches each segment in a distinct group of t users. In other words, each subset of t users is assigned one segment from each file for all the users to cache[2]. In the delivery phase the server sends one message to each subset of $t+1$ users, for a total of $\binom{K}{t+1}$ messages. This caching scheme ensures that, regardless of which files have been requested, each user in a given subset of $t+1$ nodes is missing a segment that all the others have in their cache. The message sent to that subset of nodes consists of the bitwise XOR of all $t+1$ missing segments: a set of users $\mathbf{S}$ requesting files $F_{i_1}, F_{i_2}, \ldots, F_{i_{t+1}}$ would receive the message

$$m^{\mathbf{S}} = F_{i_1}^{j_1} \oplus F_{i_2}^{j_2} \oplus \cdots \oplus F_{i_{t+1}}^{j_{t+1}}, \tag{2.1}$$

where j_k is the index for the segment cached by all the users in the set except the one requesting F_{i_k}. Each user can then cancel out the segments that it already has in its cache to recover the desired segment. In the worst case, *i.e.* when all users request different files, this scheme yields a (normalized by file size) peak rate of

$$\begin{aligned} R_C(K,t) &= \binom{K}{t+1} \Big/ \binom{K}{t} \\ &= K(1 - M/N) \frac{1}{1 + KM/N}. \end{aligned} \tag{2.2}$$

[2]Parameter t is assumed to be an integer for the sake of symmetry. Otherwise some segments would be cached more often than others, requiring special treatment during the delivery phase and complicating the analysis unnecessarily.

Under some parameter combinations, broadcasting all the missing segments uncoded could require lower rate than $R_C(K,t)$, so the generalized peak rate is

$$\min\{R_C(K,t), N-M\}$$

but this chapter will ignore those pathological cases, assuming that N, M, and K are such that $R_C(K,t) \leq N-M$. It has been shown that this peak rate is the minimum achievable for some parameter combinations and falls within a constant factor of the information-theoretic optimum for all others [1, 23].

This scheme, henceforth refered to as “Maddah’s scheme" will be the basis for multiple others throughout the chapter. It is therefore recommended that the reader has a clear understanding of Maddah’s scheme before proceeding.

2.2.3 Interference Elimination

A close examination of Maddah’s algorithm reveals that it has poor performance when the cache is small and $N \leq K$. Thus, a new coded caching scheme based on interference elimination was proposed by Tian and Chen in [5] for the case where the number of users is greater than the number of files. Instead of caching file segments in plain form, they propose that the users cache linear combinations of multiple segments. After formulating the requests, undesired terms are treated as interference that needs to be eliminated to recover the requested segment. The transmitted messages are designed to achieve this using maximum distance separable (MDS) codes [30, 31].

In the placement phase, this scheme also splits each file into $\binom{K}{t}$ non-overlapping segments of equal size and each segment is cached by t users, albeit combined with other segments. Let $F_i^{\mathbf{S}}$, where $\mathbf{S} \subseteq \{1,2,\ldots,K\}$ and $|\mathbf{S}| = t$, denote the file segment from file F_i chosen to be cached by the users in $\mathbf{S}$. In the placement phase user k collects the file segments

$$\{F_i^{\mathbf{S}} | i \in \{1,2,\ldots,N\}, k \in \mathbf{S}\}, \tag{2.3}$$

($P = \binom{K-1}{t-1}N$ in total), encodes them with a MDS code $\mathcal{C}(P_0, P)$ of length $P_0 = 2\binom{K-1}{t-1}N - \binom{K-2}{t-1}(N-1)$, and stores the $P_0 - P$ parity symbols in its cache.

The delivery phase proceeds as if all the files are requested. When only some files are requested, the scheme replaces some users' requests to the "unrequested files" and proceeds as if all files were requested. A total of $\binom{K-1}{t}$ messages are transmitted (either uncoded or coded) for each file F_i, regardless of the requests. Uncoded messages provide the segments that were not cached by the users requesting F_i, while coded messages combining multiple segments from F_i are used to eliminate the interference in their cached segments. Each user gathers $\binom{K-2}{t-1}(N-1)$ useful messages which, together with the $P - P_0$ components stored in its cache, are enough to recover all P components in the $\mathcal{C}(P_0, P)$ MDS code. A more detailed description of the messages can be found in [5].

Therefore, the total number of messages transmitted from the server is $N\binom{K-1}{t}$. In this interference elimination scheme, the following normalized (M, R) pairs are achievable:

$$\left(\frac{t\left[(N-1)t + K - N\right]}{K(K-1)}, \frac{N(K-t)}{K}\right), \quad t = 0, 1, \ldots, K. \tag{2.4}$$

This scheme is shown to improve the inner bound given in [1] for the case $N \leq K$ and has a better performance than the algorithm in subsection 2.2.2 when the cache capacity is small.

2.2.4 Extension to multiple servers

Both of the previous schemes assume that a single server stores all the files and can combine any two segments into a message. Then, they design a list of messages to be broadcast by the server, based on the users' requests. In practice, however, it is often the case that content delivery networks have multiple servers and throughput is limited by the highest load on any one server rather than by the total traffic in the link between servers and users. Shariatpanahi et al. addressed this case in [29], but still assumed that all servers had access to all the files and could therefore compose

any message. They proposed a load balancing scheme distributing the same list of messages among all the servers, scaling the peak rate by the number of servers.

If each server only has access to some of the files, the problem is significantly more complicated. The general case, where each segment can be stored by multiple servers and users, is known as the index coding problem. This is one of the core problems of network information theory but it remains open despite significant efforts from the research community [32–34]. Instead of addressing the index coding problem in its general form, we focus on the case where each data segment is stored in a single server, all caches have the same capacity, and users request a single file.

A simple way to generalize the previous schemes to our scenario is to follow the same list of messages, combining transmissions from multiple servers to compose each of them. Instead of receiving a single message with all the segments as shown in Eq. (2.1), each node would receive multiple messages from different servers. The peak rate for any one server would then be the same as in a single server system.

With parity servers storing linear combinations of the data, the peak rate can be reduced. In general, distributed storage systems use MDS codes for the parity[3], so any subset of L servers can be used to generate any message. Therefore, a simple balancing of the load by rotating among all subsets of L servers would scale the peak rate by $\frac{L}{L+L'}$, where L' is the number of parity servers. However, we intend to design caching and delivery algorithms capable of further reducing the peak rate of any one server.

2.3 File striping

The simplest way to extend single-server coded caching algorithms to a multi-server system is to spread each file across all servers. That way, each user will request an equal amount of information from each server, balancing the load. This is called data striping [38] and it is common practice in data centers and solid state drives

[3]Some systems use repetition or pyramid codes [35–37] to reduce the recovery bandwidth, but this chapter will focus on MDS codes.

(SSD), where multiple drives or memory blocks can be written or read in parallel. The users then allocate an equal portion of their cache to each server and the delivery is structured as L independent single-server demands. We now proceed to give a detailed description of how striping can reduce the peak rate of Maddah's scheme, but the same idea can be applied to any other scheme.

Each of the N files $\{F_1, F_2 \ldots, F_N\}$ is split into L blocks to be stored in different servers and each block is divided into $\binom{K}{t}$ segments. These segments are denoted by $F_i^{(j,m)}$, where $i = 1, 2, \ldots, N$ represents the file number; $j = 1, 2, \ldots, \binom{K}{t}$ the segment number; and $m = 1, 2, \ldots, L$ the block number. The m-th server is designed to store the m-th segment of each file, that is $F_i^{(j,m)}$ for every i and j.

The placement is the same as in Maddah's scheme. Each segment is cached by t users, with $\{F_i^{(j,1)}, F_i^{(j,2)}, \ldots, F_i^{(j,L)}\}$ being cached by the same user. We notice that each message transmitted by Maddah's scheme in Eq. (2.1) can be split into L components

$$F_{i_1}^{(j_1,m)} \oplus F_{i_2}^{(j_2,m)} \oplus \cdots \oplus F_{i_{t+1}}^{(j_{t+1},m)}, \tag{2.5}$$

$m = 1, 2, \ldots, L$ to be sent by different servers. Then the problem can be decomposed into L independent single-server subproblems with reduced file sizes of $\frac{F}{L}$ bits. The subproblems have the same number of users, files, and cache capacity (relative to the file size) as the global problem. Since all servers can transmit simultaneously, the peak load is reduced to $\frac{1}{L}$ of that in Eq. (2.2) (Maddah's single server scheme).

If one additional parity server P is available (RAID-4), it will store the bitwise XOR of the blocks for each file, *i.e.* $F_i^{(j,1)} \oplus F_i^{(j,2)} \oplus \cdots \oplus F_i^{(j,L)}$ for all i and j. Then, server P can take over some of the transmissions, reducing the peak load to $\frac{1}{L+1}$ of that with Maddah's scheme[4]. Specifically, instead of having all data servers transmit their corresponding component in Eq. (2.5), server P can transmit the XOR of all the components, relieving one data server from transmitting. The users can combine the rest of the components with this XOR to obtain the missing one. Similarly, if two

[4]The number of segments must be a multiple of L to achieve this reduction, but it is always possible to divide each segment into multiple chunks to fulfil this condition.

additional parity servers P and Q are available (RAID-6), it is possible to choose any L out of the $L+2$ servers to take care of each set of messages in Eq. (2.5), thereby reducing the peak rate to $\frac{1}{L+2}$ of that with Maddah's scheme.

A similar process with identical file splitting can be followed for the interference cancelling scheme, achieving the same scaling of the peak rate: $\frac{1}{L}$ when there is no parity, $\frac{1}{L+1}$ with a single parity server, and $\frac{1}{L+2}$ with two parity servers.

In practice, however, it is often preferred to avoid striping and store whole files as a single unit in each server to simplify the book-keeping, ensure security, and make the network more flexible. The rest of the chapter will focus on the case where nodes store entire files, and each user requests a file stored in a specific node.

2.4 Scheme 1: Large cache

In this section, we extend Maddah-Ali and Niesen's scheme to the multiple server system. Instead of spreading each file across multiple servers as in Section 2.3, each file is stored as a single unit in a data server, as shown in Table 2.1.

Table 2.1.: Files stored in each server in distributed storage system.

Server A	Server B	$\cdots$	Server L
A_1	B_1	$\cdots$	L_1
A_2	B_2	$\cdots$	L_2
$\vdots$	$\vdots$		$\vdots$
A_r	B_r	$\cdots$	L_r

The performance of Maddah's scheme in Eq. (2.2) is highly dependent on the cache capacity M. Compared with the interference elimination in section 2.2.3, the advantage of Maddah's scheme lies in that file segments are stored in plain form instead of encoded as linear combinations. This saves some segments from being transmitted in the delivery phase, but it requires larger cache capacities to obtain

coded caching gains. Hence, Maddah's scheme is appropriate when the cache capacity is large.

The placement phase of our algorithm is identical to that in the traditional scheme. For example, in a system with $K = 6$ users with cache capacity $M = 4$ and $N = 8$ files, each file is divided into 20 segments and each segment is stored by $t = 3$ users. Table 2.3 indicates the indices of the 10 segments that each user stores, assumed to be the same for all files without loss of generality.

In order to simplify later derivations, the notation is clarified here. Since the peak rate for the storage system is considered, we assume that all users request different files, hence each user can be represented by the file that it has requested. Denote $\mathbf{S}$ to be the user set and $m_A^{\mathbf{S}}$ to represent the message sent from server A to all the users in $\mathbf{S}$. Furthermore, if $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_i\}$ represents a vector of file indices and $\boldsymbol{\gamma} = \{\gamma_1, \gamma_2, \ldots, \gamma_i\}$ represents a vector of segment indices, then $\mathbf{A}_{\boldsymbol{\alpha}}$ represents the set of requests (or users)

$$\mathbf{A}_{\boldsymbol{\alpha}} = \{A_{\alpha_1}, A_{\alpha_2}, \ldots, A_{\alpha_i}\}$$

and $\mathbf{A}_{\boldsymbol{\alpha}}^{\boldsymbol{\gamma}}$ represents the message

$$\mathbf{A}_{\boldsymbol{\alpha}}^{\boldsymbol{\gamma}} = A_{\alpha_1}^{\gamma_1} \oplus A_{\alpha_2}^{\gamma_2} \oplus \ldots \oplus A_{\alpha_i}^{\gamma_i},$$

where A_i^j represents the j-th segment from the i-th file in server A. Similarly, $\mathbf{A}_{\boldsymbol{\alpha}}^{\boldsymbol{\gamma}} \oplus \mathbf{B}_{\boldsymbol{\alpha}}^{\boldsymbol{\gamma}}$ represents the the message:

$$\mathbf{A}_{\boldsymbol{\alpha}}^{\boldsymbol{\gamma}} \oplus \mathbf{B}_{\boldsymbol{\alpha}}^{\boldsymbol{\gamma}} = (A_{\alpha_1}^{\gamma_1} \oplus B_{\alpha_1}^{\gamma_1}) \oplus \ldots \oplus (A_{\alpha_i}^{\gamma_i} \oplus B_{\alpha_i}^{\gamma_i}).$$

We first explore the multi-server system without parity servers in subsection 2.4.1. Then we study a simple system with two data and one parity server in subsection 2.4.2. Finally, we study the cases with one and two parity servers in subsections 2.4.3 and 2.4.4, respectively.

2.4.1 No parity servers

In a system without redundancy, such as the one shown in Table 2.1, the servers cannot collaborate with each other. During the delivery phase, each user is assigned to the server storing the file that it requested, and then each data server transmits enough messages to fulfil its requests. Specifically, following Maddah's scheme, a server receiving m requests would need to transmit $\binom{K}{t+1} - \binom{K-m}{t+1}$ messages, *i.e.* one for each group of t users containing at least one of its requesters. The normalized peak rate for that server would therefore be

$$\left(\binom{K}{t+1} - \binom{K-m}{t+1}\right) \Big/ \binom{K}{t}$$

The worst case occurs when all users request files from the same server, *i.e.* $m = K$. Then the peak transmission rate is the same as in the single server system.

2.4.2 One parity and two data servers

This section focuses on a very simple storage system with two data servers and a third server storing their bitwise XOR, as shown in Table 2.2. Despite each server can only access its own files, the configuration in Table 2.2 allows composing any message by combining messages from any two servers. Intuitively, if server A (or B) finish its transmission task before the other one, it can work with the parity server to help server B (or A). This collaborative scheme allows serving two requests for files stored in the same server in parallel, balancing the load and reducing the worst case peak rate below that achieved without the parity server (see Section 2.4.1).

However, there is a better transmission scheme where messages from all three servers are combined to get more information across to the users. The basic idea is to include some unrequested segments, as well as the requested ones, in each message from a data server. If the additional segments are well chosen, they can be combined with messages from the parity server to obtain desired file segments. The algorithm developed in this section is based on this idea.

Table 2.2.: Files stored in each server in a system with two data and one parity server.

Server A	Server B	Server P
A_1	B_1	$A_1 \oplus B_1$
A_2	B_2	$A_2 \oplus B_2$
$\vdots$	$\vdots$	$\vdots$
A_r	B_r	$A_r \oplus B_r$

Just like in Maddah's scheme, data servers will send each message to a set of $t+1$ users and the message will contain the XOR of $t+1$ segments (one for each user). These segments are chosen so that all users except the intended receiver can cancel them out. If the user had requested a file stored by the sender, the message will contain the corresponding segment; otherwise the message will include its complement in terms of the parity in server P, *i.e.* A_i^j instead of B_i^j and vice versa. Therefore, the contents of each message from server A or B are uniquely determined by the sender and the set of receivers, denoted by S_1 or S_2 respectively. In the example shown in Table 2.3, the message from server A to $S_1 = \{A_1, A_2, A_3, B_4\}$, corresponding to users 1 through 4, will be $m_A^{\mathbf{S_1}} = A_1^{11} \oplus A_2^5 \oplus A_3^2 \oplus A_4^1$.

Lemma 2.4.1 *Let the receivers for servers A and B be*

$$S_1 = \{\mathbf{A}_{\boldsymbol{\alpha}}, \mathbf{B}_{\boldsymbol{\beta}}, \mathbf{A}_*\} \qquad S_2 = \{\mathbf{A}_{\boldsymbol{\alpha}}, \mathbf{B}_{\boldsymbol{\beta}}, \mathbf{B}_*\},$$

respectively, where α and β denote (possibly empty) sets of indices, the $*$ denote arbitrary sets, and $S_1 \neq S_2$. The corresponding messages are

$$m_A^{\mathbf{S_1}} = \mathbf{A}_{\boldsymbol{\alpha}}^* \oplus \mathbf{A}_{\boldsymbol{\beta}}^{\boldsymbol{\gamma}} \oplus \mathbf{A}_*^* \qquad m_B^{\mathbf{S_2}} = \mathbf{B}_{\boldsymbol{\alpha}}^{\boldsymbol{\eta}} \oplus \mathbf{B}_{\boldsymbol{\beta}}^* \oplus \mathbf{B}_*^*,$$

with segment indices chosen so that each user can cancel all but one of the components. This provides users $\mathbf{B}_{\boldsymbol{\beta}}$ and $\mathbf{A}_{\boldsymbol{\alpha}}$ with some unrequested segments $\mathbf{A}_{\boldsymbol{\beta}}^{\boldsymbol{\gamma}}$ and $\mathbf{B}_{\boldsymbol{\alpha}}^{\boldsymbol{\eta}}$, respectively. Then server P can send the message

$$m_P^{\mathbf{S_1} \cap \mathbf{S_2}} = (\mathbf{A}_{\boldsymbol{\alpha}}^{\boldsymbol{\eta}} \oplus \mathbf{B}_{\boldsymbol{\alpha}}^{\boldsymbol{\eta}}) \oplus (\mathbf{A}_{\boldsymbol{\beta}}^{\boldsymbol{\gamma}} \oplus \mathbf{B}_{\boldsymbol{\beta}}^{\boldsymbol{\gamma}}),$$

to $S_1 \cap S_2$, so that each user in S_1 and S_2 obtains a missing segment and those in the intersection obtain two. These three transmissions are equivalent to messages $m^{\mathbf{S_1}}$ and $m^{\mathbf{S_2}}$ as defined in Eq. (2.1) for Maddah's single server scheme. They both provide the same requested segments to their destinations.

Proof All the users in S_1 and S_2 get at least one desired segment, from the server storing their requested file. Those in $S_1 \cap S_2$ also receive an unrequested segment from server A or B. It only remains to prove that users in $S_1 \cap S_2$ can use this unrequested segment to obtain its complement from $m_P^{\mathbf{S_1} \cap \mathbf{S_2}}$.

Without loss of generality, consider user $B_{\beta_i} \in S_1 \cap S_2$. The set of segment indices γ in $m_A^{\mathbf{S_1}}$ were chosen so that user B_{β_i} is caching all the segments except the γ_i-th. Similarly, the set of indices η in $m_B^{\mathbf{S_2}}$ was chosen so that B_{β_i} is caching all of them (for all files). Therefore, B_{β_i} can obtain $A_{\beta_i}^{\gamma_i}$ from $m_A^{\mathbf{S_1}}$ and should be able to cancel all terms from $m_P^{\mathbf{S_1} \cap \mathbf{S_2}}$ except $A_{\beta_i}^{\gamma_i} \oplus B_{\beta_i}^{\gamma_i}$. Combining both of these yields the desired segment $B_{\beta_i}^{\gamma_i}$. As long as $S_1 \neq S_2$, this segment will be different from the one that B_{β_i} obtains from $m_B^{\mathbf{S_2}}$ because there is a one-to-one relationship between segment indices and user subsets. ■

Take the case in Table 2.3 as an example. Lemma. 2.4.1 states that if $S_1 = \{A_1, A_2, A_3, B_4\}$ and $S_2 = \{A_1, A_2, B_1, B_4\}$, we construct $m_A^{\mathbf{S_1}}$, $m_B^{\mathbf{S_2}}$, $m_P^{\mathbf{S_1} \cap \mathbf{S_2}}$ as:

$$
\begin{aligned}
m_A^{\mathbf{S_1}} &= A_1^{11} \oplus A_2^5 \oplus A_3^2 \oplus A_4^1, \\
m_B^{\mathbf{S_2}} &= B_1^{14} \oplus B_2^8 \oplus B_1^2 \oplus B_4^3, \\
m_P^{\mathbf{S_1} \cap \mathbf{S_2}} &= (A_1^{14} \oplus B_1^{14}) \oplus (A_2^8 \oplus B_2^8) \oplus (A_4^1 \oplus B_4^1).
\end{aligned}
$$

It is easy to verify that these messages are equivalent to two transmissions in Maddah's scheme, specifically those intended for users $\{A_1, A_2, A_3, B_4\}$ and $\{A_1, A_2, B_1, B_4\}$.

Corollary 2.4.1.1 *Assume $S_1 = \{\mathbf{A}_*, \mathbf{B}_{\boldsymbol{\beta}}\}$ and $S_2 = \{\mathbf{B}_*\}$, i.e. it only contains requests for server B. Then server P sends $m_P^{\mathbf{B}_{\boldsymbol{\beta}}} = \mathbf{A}_{\boldsymbol{\beta}}^{\boldsymbol{\gamma}} \oplus \mathbf{B}_{\boldsymbol{\beta}}^{\boldsymbol{\gamma}}$ to all the users in $\mathbf{B}_{\boldsymbol{\beta}}$ in Lemma 2.4.1, so that all the users in S_1 and S_2 get the same segments as in Maddah's scheme. The same holds switching the roles of A and B.*

Table 2.3.: Mapping of file segments to user caches. Each cache stores the same 10 segments for every file, marked with X ($K = 6$, $M = 4$, $N = 8$).

Segment\ User	1	2	3	4	5	6
1	X	X	X			
2	X	X		X		
3	X	X			X	
4	X	X				X
5	X		X	X		
6	X		X		X	
7	X		X			X
8	X			X	X	
9	X			X		X
10	X				X	X
11		X	X	X		
12		X	X		X	
13		X	X			X
14		X		X	X	
15		X		X		X
16		X			X	X
17			X	X	X	
18			X	X		X
19			X		X	X
20				X	X	X
Request	A_1	A_2	A_3	B_4	B_1	B_2

Proof This is a particular case of Lemma 2.4.1 when $\boldsymbol{\alpha}$ is empty ($\boldsymbol{\beta}$ can be empty or non-empty). ■

Definition 2.4.1 *If user subsets S_1 and S_2 fulfill the conditions in Lemma 2.4.1, we call (S_1, S_2) an **effective pair**.*

Our goal is to design a scheme equivalent to Maddah's scheme while minimizing the maximum number of messages sent by any server. If two user subsets form an effective pair, the corresponding messages in Maddah's scheme (see Eq. (2.1)) can be replaced by a single transmission from each server. Hence, we wish to make as many effective pairs as possible.

Lemma 2.4.2 *The peak rate is $\left(\frac{1}{2} + \frac{1}{6}\Delta\right) R_C(K,t)$ for the server system in Table 2.2, where Δ represents the ratio of unpaired messages and $t = \frac{KM}{N}$.*

Proof For each effective pair, we can use a single transmission from each server to deliver the same information as two transmissions in Maddah's single server scheme. This contributes $\frac{1}{2}(1-\Delta)R_C(K,t)$ to the total rate. Unpaired messages are transmitted as described in section 2.2.4, that is combining messages from any two out of the three servers. Assuming that this load is balanced among all three servers, the contribution to the total rate is $\frac{2}{3}\Delta R_C(K,t)$. Adding both contributions yields the rate above. ■

The following lemma characterizes the ratio of unpaired user subsets Δ in the case with symmetric requests (both servers receive the same number of requests).

Lemma 2.4.3 *If the requests are symmetric, then $\Delta = 0$ when t is even and $\Delta \leq \frac{1}{3}$ when t is odd. That is, the following peak rate is achievable in the case with symmetric requests:*

$$R_T(K,t) = \begin{cases} \frac{1}{2} R_C(K,t) & \text{if } t \text{ is even} \\ \left(\frac{1}{2} + \frac{1}{6}\Delta\right) R_C(K,t) & \text{if } t \text{ is odd,} \end{cases} \tag{2.6}$$

where $R_C(K,t)$ is defined in Eq. (2.2).

Proof A pairing algorithm with these characteristics is presented in the Appendix. ■

Although Δ can reach $\frac{1}{3}$, in most cases the pairing algorithm in the Appendix performs much better. As an example, Table 2.3 has each segment cached by $t = \frac{KM}{N} = 3$ users and the normalized peak rate with the pairing algorithm is $\frac{2}{5}$, significantly lower than the $\frac{3}{4}$ with Maddah's single server scheme.

Finally, we are ready to derive an achievable peak rate for a general set of requests, based on the following lemma.

Lemma 2.4.4 *If (S_1, S_2) form an effective pair, then $S_1' = \{S_1, \mathbf{A}_{\boldsymbol{\alpha}}\}$ and $S_2' = \{S_2, \mathbf{A}_{\boldsymbol{\alpha}}\}$ also form an effective pair of a larger dimension. The same holds when an all-B file set is appended instead of the all-A file set $\mathbf{A}_{\boldsymbol{\alpha}}$.*

Proof The proof is straightforward by observing that (S_1', S_2') still fulfills the conditions in Lemma 2.4.1. ■

The extension to the asymmetric case is as follows. Let K_A and K_B respectively denote the number of requests for servers A and B, and assume $K_A > K_B$ without loss of generality. Divide the $K = K_A + K_B$ requests (or users) into two groups: the first with K_B requests for each server (symmetric demands) and the second with the remaining $K_A - K_B$ requests for server A. We construct effective pairs of length $t+1$ by appending requests from the second group to effective pairs from the first.

Theorem 2.4.5 *If the requests are asymmetric, the ratio of unpaired messages is also bounded by $\Delta \leq \frac{1}{3}$. Specifically, if K_A and K_B respectively denote the number of requests for servers A and B, assuming $K_A > K_B$ without loss of generality, the following normalized peak rate is achievable:*

$$R(K_A, K_B, t) = \sum_{l=0}^{t+1} \left(\binom{K_A - K_B}{l} R_T(2K_B, t-l), \right. \tag{2.7}$$

where R_T is defined in Eq. (2.6) and $K = K_A + K_B$.

Proof From Lemma 2.4.3, $R_T(2K_B, t-l)$ represents the peak rate after pairing all subsets of $t+1-l$ requests from the symmetric group. For each $l = 0, 1, \ldots, t+1$, we multiply $R_T(2K_B, t-l)$ by the number of possible completions with l requests from the second group, to obtain the peak rate corresponding to subsets with $t+1-l$ requests from the first group and l from the second. Adding them for all l gives Eq. (2.7).

Since $R_T(i,j) \leq \left(\frac{1}{2} + \frac{1}{6}\Delta\right) R_C(i,j)$ with $\Delta \leq \frac{1}{3}$ by Lemma 2.4.3, and $\sum_{l=0}^{t+1} \binom{K_A - K_B}{l} R_C(2K_B, t-l) = R_C(K,t)$ by combinatorial equations, Eq. (2.7) implies that $R(K_A, K_B, t) \leq \left(\frac{1}{2} + \frac{1}{6}\Delta\right) R_C(K,t)$ with $\Delta \leq \frac{1}{3}$ as defined in Lemma 2.4.2. ■

Corollary 2.4.5.1 *A peak rate of $\frac{5}{9} R_C(K,t)$ is achievable for a system with two data servers and a parity server.*

2.4.3 One parity and L data servers

The previous subsection has discussed the case with two data servers and one parity server, but the same algorithm can be extended to systems with more than two data servers. Intuitively, if there are L data servers and one parity server, any message can be built by combining messages from any L servers. A first approach could be distributing the $\binom{K}{t+1}$ messages in Maddah's scheme across the $L+1$ possible groups of L servers, as proposed in subsection 2.2.4. Each server would then need to send a maximum of $\binom{K}{t+1} \frac{L}{L+1}$ messages. However, there is a more efficient way of fulfilling the requests based on the algorithms in subsections 2.2.4, 2.4.1 and 2.4.2.

Lemma 2.4.6 *Let $S_1 = \{\mathbf{A}_{\boldsymbol{\alpha}}, \mathbf{B}_{\boldsymbol{\beta}}, \mathbf{A}_*, \mathbf{Y}\}$ and $S_2 = \{\mathbf{A}_{\boldsymbol{\alpha}}, \mathbf{B}_{\boldsymbol{\beta}}, \mathbf{B}_*, \mathbf{Y}'\}$ be two user subsets, where $\mathbf{Y}$ and $\mathbf{Y}'$ are arbitrary lists of requests for servers C through L and the $*$ represent arbitrary (possibly empty) index sets. Then, S_1 and S_2 can be paired so that servers A, B and P require a single transmission to provide the same information as messages $m^{\mathbf{S_1}}$ and $m^{\mathbf{S_2}}$ in Maddah's single server scheme. The other data servers, C*

through L, require a maximum of two transmissions, as shown in paired transmissions in Fig. 2.1.

Proof The transmissions would proceed as follows:

1. Servers C through L each send two messages, to S_1 and S_2. For example, server C would send $m_C^{S_1}$ and $m_C^{S_2}$, providing a desired segment to users requesting files from C and the corresponding C-segments to those requesting other files.

2. Server A sends[5] $m_A^{S_1}$, providing a desired segment to users requesting $\{\mathbf{A}_*, \mathbf{A}_{\boldsymbol{\alpha}}\}$ and the corresponding undesired A-segments to those requesting $\mathbf{B}_{\boldsymbol{\beta}}$.

3. Server B sends $m_B^{S_2}$, providing a desired segment to users requesting $\{\mathbf{B}_{\boldsymbol{\beta}}, \mathbf{B}_*\}$ and the corresponding undesired B-segments to those requesting $\mathbf{A}_{\boldsymbol{\alpha}}$.

4. Server P sends $m_P^{\{\mathbf{A}_{\boldsymbol{\alpha}}, \mathbf{B}_{\boldsymbol{\beta}}\}}$ to users requesting $\{\mathbf{A}_{\boldsymbol{\alpha}}, \mathbf{B}_{\boldsymbol{\beta}}\}$. Using the undesired segments previously received, the users in $\{\mathbf{A}_{\boldsymbol{\alpha}}, \mathbf{B}_{\boldsymbol{\beta}}\}$ can solve for the desired A and B segments.

A simple comparison of the requested and received segments shows that these transmissions deliver the same information as messages $m^{\mathbf{S_1}}$ and $m^{\mathbf{S_2}}$ in Maddah's single server scheme. ■

As an example, Table 2.4 shows the segments that each user gets in transmissions (1)-(4) when $S_1 = \{A_1, A_2, B_1, C_1\}$ and $S_2 = \{A_1, B_1, B_2, C_2\}$, respectively corresponding to segments $\{A_1^1, A_2^2, B_1^3, C_1^4\}$ and $\{A_1^5, B_1^6, B_2^7, C_2^8\}$.

Theorem 2.4.7 *The following normalized peak rate is achievable for a system with $L \geq 3$ data servers and one parity server:*

$$R_P(K,t) = \frac{L-1}{L} R_C(K,t), \tag{2.8}$$

where R_C is defined in Eq. (2.2).

[5]It would be enough for A to send $m_A^{\{\mathbf{A}_*, \mathbf{A}_{\boldsymbol{\alpha}}, \mathbf{B}_{\boldsymbol{\beta}}\}}$ instead of $m_A^{S_1}$, but we use the latter for the sake of simplicity. The same applies to the message from server B.

Table 2.4.: Segments received by each users in transmissions (1)-(4) from Lemma 2.4.6, where $P_i^j = A_i^j \oplus B_i^j \oplus C_i^j$.

Trans.\Req.	A_1	A_2	B_1	B_2	C_1	C_2
(1)	C_1^5		C_1^3		C_1^4	C_2^8
(2)	A_1^1	A_2^2	A_1^3			
(3)	B_1^5		B_1^6	B_2^7		
(4)	P_1^5		P_1^3			
in total	A_1^1, A_1^5	A_2^2	B_1^3, B_1^6	B_2^7	C_1^4	C_2^8

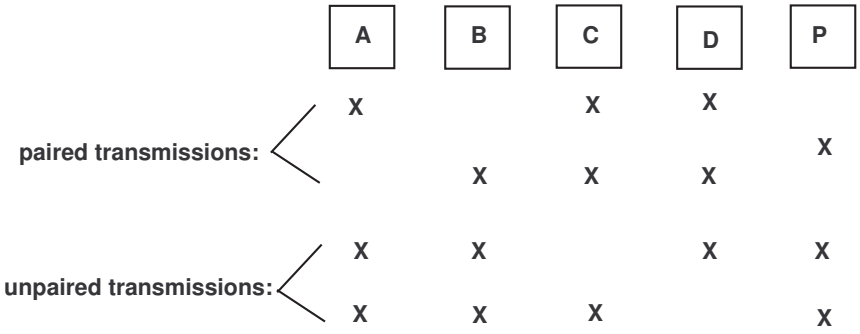


Figure 2.1.: Pairing for 4 data servers and 1 parity server system. A, B, C, D are data servers and P represents the parity server. X means there is a message transmitted from the corresponding server.

Proof First we show that we can deliver $\frac{2}{L}\binom{K}{t+1}$ of the messages in Maddah's scheme using at most $\frac{1}{L}\binom{K}{t+1}$ transmissions from servers A, B and P; and at most $\frac{2}{L}\binom{K}{t+1}$ transmissions from each of the other servers. This can be done by pairing the messages as shown in Lemma 2.4.6, if they include requests for A or B, and by using the scheme in subsection 2.4.1, if they do not.

Selecting these $\frac{2}{L}\binom{K}{t+1}$ messages can be done as follows: group messages by the number of segments that they have from servers A or B. Within each group, we pair the messages as shown in Lemma 2.4.6. This is equivalent to pairing the A and B requests into effective pairs according to Theorem 2.4.5 and considering all possible completions for each pair using requests for other servers. Theorem 2.4.5 showed that at least $\frac{2}{3} \geq \frac{2}{L}$ of the messages in each group can be paired. Messages which have no A or B segments can be transmitted as described in section 2.4.1, without requiring any transmissions from servers A, B or P.

The remaining $\frac{L-2}{L}\binom{K}{t+1}$ messages can be transmitted as described in subsection 2.2.4, distributing the savings evenly among servers C through L. This requires $\frac{L-2}{L}\binom{K}{t+1}$ transmissions from servers A, B and P; and $\frac{L-3}{L}\binom{K}{t+1}$ from each of the rest. Each server then transmits a total of $\frac{L-1}{L}\binom{K}{t+1}$, hence the peak rate in Eq. (2.8). ■

Theorem 2.4.7 provides a very loose bound for the peak rate in a system with one parity and L data servers. In practice, there often exist alternative delivery schemes with significantly lower rates. For example, if all the users request files from the same server, that server should send half of the messages while all the other servers collaborate to deliver the other half. The rate would then be reduced to half of that in Maddah's scheme. Similarly, if $L > t+1$ and all the servers receive similar numbers of requests, the scheme in subsection 2.4.1 can provide significantly lower rates than Eq. (2.8).

Table 2.5.: Files stored in parity servers in RAID-6.

Server P	Server Q
$A_1 + B_1 + \ldots + L_1$	$A_1 + \kappa_B B_1 + \ldots + \kappa_L L_1$
$A_2 + B_2 + \ldots + L_2$	$A_2 + \kappa_B B_2 + \ldots + \kappa_L L_2$
$\vdots$	$\vdots$
$A_r + B_r + \ldots + L_r$	$A_r + \kappa_B B_r + \ldots + \kappa_L L_r$

2.4.4 Two parity and L data servers

In this section, we will extend our algorithm to a system with L data and two linear parity servers operating in a higher order field instead of GF(2). The parity server P stores the horizontal sum of all the files while the parity server Q stores a different linear combination of the files BY ROW, as shown in Table 2.5. It will be assumed that the servers form an MDS code. We will show that with a careful design of the delivery strategy, the peak rate can be reduced to almost half of that with Maddah's single server scheme.

Lemma 2.4.8 *Let* $S_1 = \{\mathbf{A}_*, \mathbf{Y}\}$ *and* $S_2 = \{\mathbf{B}_*, \mathbf{Y}\}$*, where* $\mathbf{Y}$ *represents a common set of requests from any server. Then* S_1 *and* S_2 *can be paired so that a single transmission from each server fills the same requests as messages* $m^{\mathbf{S_1}}$ *and* $m^{\mathbf{S_2}}$ *in Eq. (2.1).*

Proof The transmission scheme shares the same pairing idea as the algorithm in subsection 2.4.2. The transmissions are as follows:

1. Server A sends $m_A^{\mathbf{S_1}}$, providing a desired segment to users requesting its files and the corresponding undesired A-segments to others.

2. Server B sends $m_B^{\mathbf{S_2}}$, providing a desired segment to users requesting its files and the corresponding undesired B-segments to others.

3. Servers $C, D, \ldots, L$ each send a single message to $S_1 \bigcap \mathcal{S}_2 = \{\mathbf{Y}\}$ with the following content for each user:

   - Users requesting files from server B received some undesired segments from server A. Servers $C, D, \ldots, L$ send them the matching ones so that the desired segments can be decoded using the parity in server P later.
   - The remaining users in $\mathbf{Y}$ will get the desired segment corresponding to S_1 when possible, otherwise they will get the undesired segment corresponding to S_2.

   In other words, each server $C, \ldots, L$ will send segments corresponding to S_1 to users requesting its files or those from server B, and segments corresponding to S_2 to the rest. At this point, all the users have satisfied their requests related to S_1, except those requesting files from server B, who satisfied their requests related to S_2 instead. Each user has also received $L-2$ undesired "matched" segments[6], corresponding to S_1 for those requesting files from server B and corresponding to S_2 for the rest.

4. Finally, parity servers P and Q each transmit a message to $S_1 \bigcap \mathcal{S}_2 = \{\mathbf{Y}\}$ with a combination of segments for each user (see Table 2.5). Those requesting files from server B will get two combinations of the segments corresponding to S_1, while the rest will get two combinations of the segments corresponding to S_2. Since each user now has $L-2$ individual segments and two independent linear combinations of all L segments, it can isolate the requested segment (as well all the "matching" segments in other servers).

A simple comparison of the requested and received segments shows that these transmissions deliver the same information as messages $m^{\mathbf{S_1}}$ and $m^{\mathbf{S_2}}$ in Maddah's single server scheme. ■

[6]Users in $\mathbf{Y}$ requesting files from servers A or B received $L-1$ "matched" segments instead of $L-2$, but we can ignore the extra one.

Table 2.6.: Segments users get in (1)-(4) transmissions (In order to simplify notation, denote $P_i^j = A_i^j + B_i^j + C_i^j$ and $Q_i^j = A_i^j + \kappa_B B_i^j + \kappa_C C_i^j$).

Trans.\Req.	A_1	A_2	B_1	B_2	C_1	C_2
(1)	A_1^1	A_2^2	A_1^3		A_1^4	A_2^5
(2)	B_1^6		B_1^7	B_2^8		
(3)	C_1^6		C_1^3		C_1^9	C_2^{10}
(4)	P_1^6		P_1^3		P_1^4, Q_1^4	P_2^5, Q_2^5
in total	A_1^1, A_1^6	A_2^2	B_1^3, B_1^7	B_2^8	C_1^4, C_1^9	C_2^5, C_2^{10}

As an example, Table 2.6 shows the delivered segments in transmissions (1)-(4) if $m^{\mathbf{S_1}} = \{A_1^1, A_2^2, B_1^3, C_1^4, C_2^5\}$ and $m^{\mathbf{S_2}} = \{A_1^6, B_1^7, B_2^8, C_1^9, C_2^{10}\}$.

Theorem 2.4.9 *For the L data server and two parity server system, the following normalized peak rate is achievable:*

$$R_Q(K,t) = \left(\frac{1}{2} + \frac{L-2}{2L+4}\Delta\right) R_C(K,t), \tag{2.9}$$

where $\Delta \leq \frac{1}{3}$ is the pairing loss and R_C is the rate of the single server Maddah's scheme in Eq. (2.2).

Proof Group messages by the number of segments that they have from servers A or B. Within each group, we pair the messages as shown in Lemma 2.4.8. If the number of requests from A or B is not zero, this is equivalent to pairing the A and B requests into effective pairs according to Theorem 2.4.5 and considering all possible completions for each pair using requests for other servers. Theorem 2.4.5 showed that at most $\frac{1}{3}$ of the messages in each group remains unpaired. For the messages which do not contain segments from A or B we repeat the same process with two other servers, with identical results: at most $\frac{1}{3}$ of them remain unpaired.

Each pair of messages can be delivered using a single transmission from each server, as shown in Lemma 2.4.8, hence paired messages contribute $\frac{1}{2}(1-\Delta)R_C(K,t)$ to the total rate, where Δ denotes the ratio of unpaired messages. Unpaired messages

are transmitted as described in section 2.2.4, that is using L out of the $L+2$ servers. Balancing this load among all the servers, they contribute $\frac{L}{L+2}\Delta R_C(K,t)$ to the total rate. Adding both contributions yields the rate above. ■

2.5 Scheme 2: Small cache

This section extends the interference elimination scheme in section 2.2.3 to a multi-server system. The interference elimination scheme is specially designed to reduce the peak rate when the cache size is small [5]. Unlike Maddah's scheme, which caches plain segments, the interference elimination scheme proposes caching linear combinations of them. That way each segment can be cached by more users, albeit with interference. This section will start with the system without parity in Table 2.1, showing that the transmission rate decreases as $\frac{1}{L}$ with the number of servers. Then it performs a similar analysis for the case with parity servers, which can be interpreted as an extension of the user's caches.

Theorem 2.5.1 *In a system with L data servers and parallel channels, the peak rate of the interference cancelling scheme can be reduced to $\frac{1}{L}$ of that in a single server system, i.e. the following (M,R) pair is achievable:*

$$\left(t, \frac{[(N-1)t+K-N]}{K(K-1)}, \frac{N(K-t)}{LK}\right) \quad t=0,1,\ldots,K. \tag{2.10}$$

This holds regardless of whether each file is spread across servers (striping) or stored as a single block in one server.

Proof Section 2.3 showed that striping the files across L servers reduces the peak rate of the interference cancelling scheme by $\frac{1}{L}$ compared with a single server system.

In contrast to Maddah's scheme, the interference cancelling scheme sends the same number of segments from each file, regardless of the users' requests. Moreover, each message consists of a combination of segments from a single file [5]. Therefore, the same messages can be transmitted even if different files are stored in different servers. Each server will need to transmit a fraction $\frac{1}{L}$ of the messages, since it will be storing

that same fraction of the files. The peak load can then be reduced to $\frac{1}{L}$ of that in Eq. (2.4). ■

If there are parity servers, we can further reduce the transmission rate by regarding them as an extension of the users' cache. Section 2.2.3 explained that in the interference elimination algorithm [5], each user caches the parity symbols resulting from encoding a set of segments with a systematic MDS code $\mathcal{C}(P_0, P)$. It is possible to pick the code in such a way that some of these parity symbols can be found as combinations of the information stored in servers P and Q. Then, instead of storing them in the user's cache, they are discarded. Those that are needed in the delivery phase will be transmitted by the parity servers.

For example, parity server P stores the horizonal sum of the files, so it can transmit messages of the form:

$$\sum_{i=1}^{N/L} \sum_{j=1}^{\binom{K-1}{t-1}} \lambda_{ij} \left(A_i^{\mathbf{s}_j} + B_i^{\mathbf{s}_j} \ldots + L_i^{\mathbf{s}_j} \right) \Bigg($$

with arbitrary coefficients λ_{ij} for any user set $\mathbf{s}_j$. This corresponds to a linear combination of all the segments in Eq. (2.3). Similarly, parity server Q can transmit some other linear combinations of the segments which can also work as components of an MDS code. This effectively increases the size of the cache memories by M' file units, corresponding to the amount of information that the parity servers can afford to send each user during the delivery phase.

Theorem 2.5.2 *If there are η parity servers and $K \geq N$, the following (M, R) pairs are achievable for $t = 0, 1, \ldots, K$*

$$\left(\frac{t\left[(N-1)t + K - N\right]}{K(K-1)} - \eta \frac{N(K-t)}{LK^2}, \frac{N(K-t)}{LK} \right) \Bigg($$

Proof The information sent by the parity server is bounded by the peak rate of the data servers, *i.e.* $\frac{N(K-t)}{LK}$ according to Eq. (2.10). Assuming a worst case scenario, each transmission from a parity server will benefit a single user. Therefore, each parity server can effectively increase the cache of each user by $M' = \frac{N(K-t)}{LK^2}$. ■

This memory sharing strategy provides significant improvement when the cache capacity is small. Fig. 2.2 shows the performance for $K = 15$ users and $N = 12$ files stored in $L = 4$ data servers. When the cache size is small, the peak rate of the system with two parity servers is much lower than that without parity servers. As the cache grows the advantage of the system with parity servers becomes less clear.

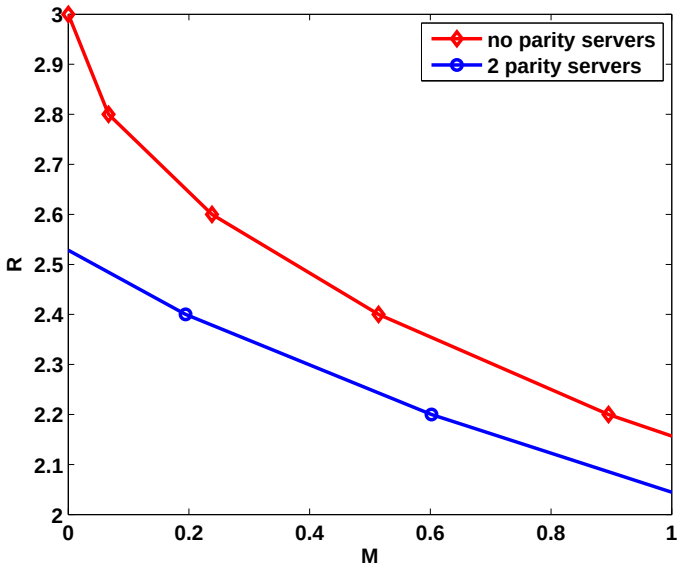


Figure 2.2.: Comparison of the performance between multi-server system without parity servers and the system with two parity servers.

The interference elimination scheme is specially designed for the case with less files than users ($N \leq K$) in the single server system. However, since the peak load is reduced by $\frac{1}{L}$ in a multi-server system, the interference elimination scheme might also have good performance when $N > K$ if L is large. In order to apply the algorithm, we can just add $N - K$ dummy users with arbitrary requests. Then, we have the following corollary from Theorem 2.5.2:

Table 2.7.: Normalized peak rate of Scheme 1.

server system	Normalized peak rate
single server	$R_C(K,t) = \binom{K}{t+1} / \binom{K}{t}$
L data 1 parity	$\frac{L-1}{L} R_C(K,t)$
L data 2 parity	$(\frac{1}{2} + \frac{L-2}{2L+4}\Delta) R_C(K,t)$ $(\Delta \leq \frac{1}{3})$

Table 2.8.: Normalized (M,R) pair of Scheme 2. (η is the number of parity servers.)

server system	Normalized (M,R)
single server	$\left(\frac{t[(N-1)t+K-N]}{K(K-1)}, \frac{N(K-t)}{K}\right)$
L data η parity $(K \geq N)$	$\left(\frac{t[(N-1)t+K-N]}{K(K-1)} - \eta\frac{N(K-t)}{LK^2}, \frac{N(K-t)}{LK}\right)$
L data η parity $(K \leq N)$	$\left(\frac{t^2}{N} - \eta\frac{(N-t)}{LN}, \frac{(N-t)}{L}\right)$

Corollary 2.5.2.1 *If there are η parity servers and $K \leq N$, the following (M,R) pairs are achievable:*

$$\left(\frac{t^2}{N} - \eta\frac{(N-t)}{LN}, \frac{(N-t)}{L}\right) \quad t = 0, 1, \ldots, N.$$

2.6 Simulations

This section compares all the schemes studied in this chapter, for a system with $N = 20$ files stored in $L = 4$ data servers with 5 files each. We show that striping has better performance than the schemes in sections 2.4 and 2.5 (Scheme 1 and Scheme 2, respectively) at the cost of network flexibility. If each file is stored as a single block in one server, Scheme 2 has better performance when the cache capacity is small while Scheme 1 is more suitable for the case where the cache capacity is large. The performances of Scheme 1 and Scheme 2 are summarized in Table 2.7 and Table 2.8, respectively.

Fig. 2.3 and Fig. 2.4 focus on the case with one and two parity servers, respectively. We assume that there are $K = 15$ users, thus there are more files than users,

with varying cache capacity. We observe that striping provides lower peak rates than storing whole files, as expected. Additionally, since $N > K$, the interference elimination scheme always has worse performance than Maddah's scheme when striping is used. Without striping, Scheme 2 provides lower peak rate than Scheme 1 when the cache capacity is small, and it is the other way around when the capacity is large.

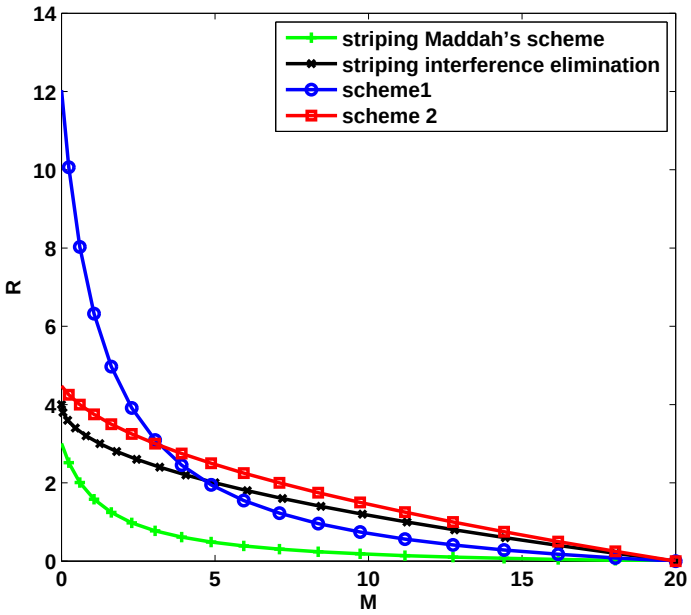


Figure 2.3.: Comparison between the performance between Scheme 1 and Scheme 2 in one parity server system when $N = 20$ and $K = 15$.

Then Fig. 2.5 and Fig. 2.6 compare the performance between Scheme 1 and Scheme 2 when there are more users ($K = 60$) than files for the one or two parity case, respectively. As shown in Fig. 2.5 and Fig. 2.6, the striping has lower rate than storing whole files and when the cache capacity is very small, the striping interference elimination has better performance than striping Maddah's scheme. For Scheme 1 and Scheme 2, when the cache capacity is small, Scheme 2 provides lower peak rate, while when the cache capacity increases, Scheme 1 has better performance.

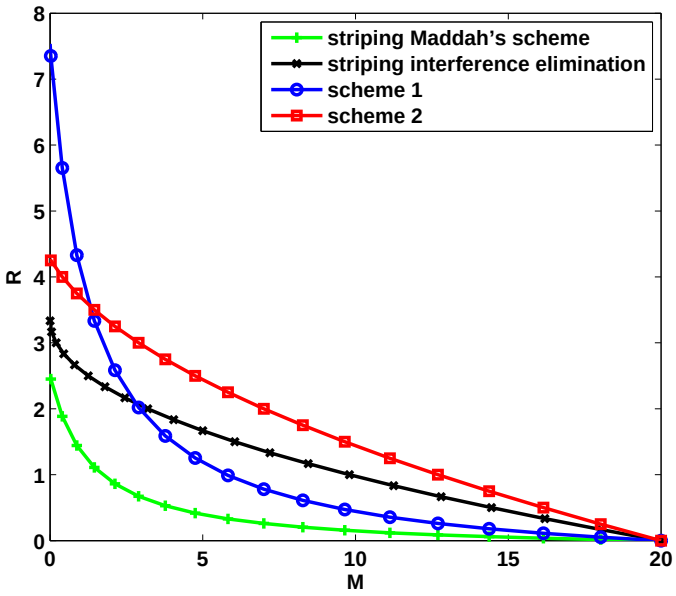


Figure 2.4.: Comparison between the performance between Scheme 1 and Scheme 2 in two parity server system when $N = 20$ and $K = 15$.

Moreover, we notice that the curves intersect at a point with larger M than they did in Fig. 2.3 and Fig. 2.4, which means that we are more prone to utilize Scheme 2 when there are more users than files.

2.7 Summary

This chapter proposes coded caching algorithms for reducing the peak data rate in multi-server systems with distributed storage and different levels of redundancy. It shows that, by striping each file across multiple servers, the peak rate can be reduced proportionally to the number of servers. Then it addresses the case where each file is stored as a single block in one server and proposes different caching and delivery schemes depending on the size of the cache memories.

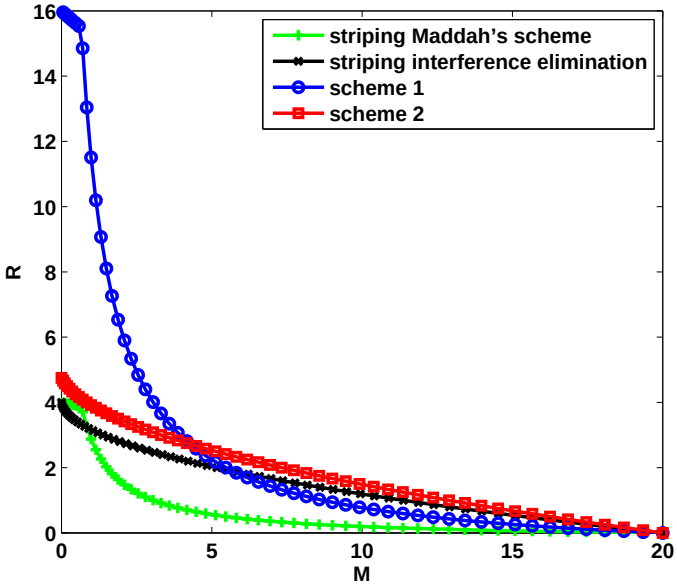


Figure 2.5.: Comparison between the performance between scheme 1 and scheme 2 in one parity server system when $N = 20$ and $K = 60$.

Distributed storage systems generally use MDS codes across the servers to protect the information against node failures. The coded caching schemes proposed in this chapter are able to leverage that redundancy in creative ways to reduce the achievable traffic peak rate. The results for Scheme 1 and Scheme 2 are shown in Table 2.7 and Table 2.8 respectively.

This chapter proposed methods to reduce the load on the links between servers and users, which is the most common bottleneck for system performance. However, there are cases in which the server I/Os, not the overall traffic on the links, are the limiting parameter. The next chapter will study the trade-off between network traffic load and disk I/Os.

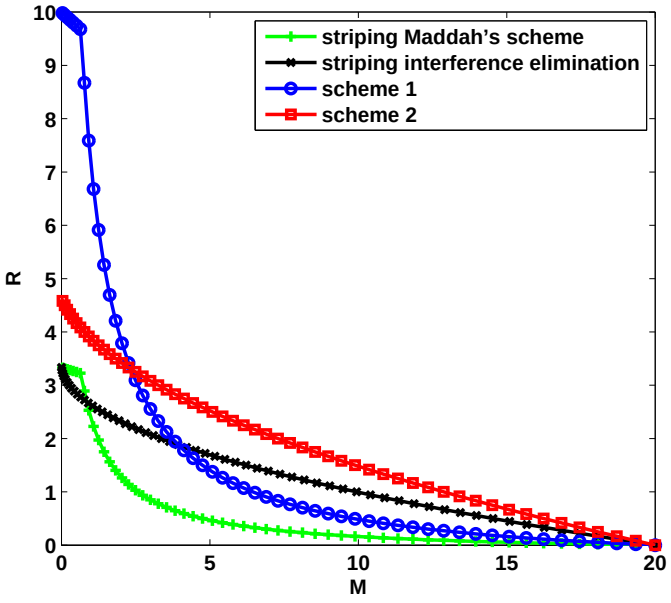


Figure 2.6.: Comparison between the performance between scheme 1 and scheme 2 in two parity server system when $N = 20$ and $K = 60$.

3. TRAFFIC LOAD-I/O TRADE-OFF FOR CACHING

3.1 Introduction

Users and applications demand accessing data at a higher speed and lower latency nowadays, which poses challenges to both networks and devices. This chapter addresses two performance bottlenecks of storage systems: the number of read and write operations (disk I/Os) and the amount of data transferred (transfer load).

Disk I/Os are a valuable resource. Many applications are I/O bounded and serve a huge number of user requests and perform intensive computations. A significant amount of research has gone into coding techniques to minimize disk I/Os in storage systems [9, 10].

Transfer load (or traffic) is another dominant factor in slow or congested networks. Caching has been investigated as a useful technique to relieve peak traffic by prefetching contents during off-peak hours. A caching scheme has two phases: placement and delivery. In the placement phase, the users have access to all files to fill their caches. In the delivery phase, only the server has database access and it delivers messages to the users to fulfill their requests. In [1], Maddah-Ali and Niesen proposed a caching and delivery scheme offering a worst case performance within a constant factor of the information-theoretic optimum, for a system with a single server broadcasting to multiple users and uniform file popularity. Inspired by their work, [40, 41] studied its average performance and the case with random demands. Further works improved the delivery scheme by exploiting commonality among users' demands [6, 11, 42] and introduced a decentralized version [43].

This chapter focuses on the same system, illustrated in Fig. 3.1. Maddah-Ali and Nisen's coded caching scheme in [1] (henceforth denoted "M-N scheme") has the lowest peak traffic load in the literature and its extension by Yu et. al. [11]

(henceforth denoted “Yu’s scheme") is proved to achieve the best average traffic load with uncoded prefetching. However, their I/O performance is suboptimal when there are redundant user demands. The same segment could be read multiple times if it is used to construct different messages, which dramatically increases I/O reads. In contrast, if all messages are transmitted uncoded, each data segment requested is read once and broadcast to all users. Inspired by this fact, we study the trade-off between traffic load and I/O by designing algorithms which combine coded and uncoded transmission. To the extent of our knowledge, this is the first work which studies the I/O performance for coded caching.

The rest of this chapter is organized as follows. Section 3.2 introduces the system model, the traditional uncoded and coded caching schemes. Section 3.3 proposes two algorithms which study the trade-off between traffic load and I/O access. Section 3.4 provides simulations to support and illustrate our algorithms and Section 3.5 concludes the chapter.

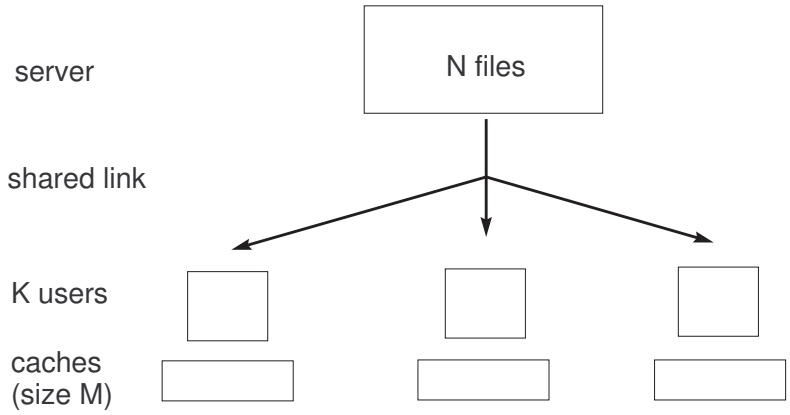


Figure 3.1.: Caching system considered in this chapter.

3.2 Background

3.2.1 System Model

Our system model is identical to the one in [1], shown in Fig. 3.1: a single server is connected to K users through a shared broadcast link, and N files of size F bits with uniform popularity. Each user has a cache of size MF bits.

Users fill their caches during the placement phase and then independently request a file in the delivery phase. We denote these requests by $\mathbf{d} = \{d_1, \ldots, d_k\}$, where d_k is the index of the file requested by user k. $\mathbf{d}$ is uniformly distributed over $\mathcal{D} = \{1, \ldots, N\}^K$ and the number of distinct files requested is denoted by $N_e(\mathbf{d})$.

The server must then fulfil those requests. We wish to study the trade-off between the resulting load on the shared link and the disk I/O. Disks are read one page at a time, all of the same size [44]. Therefore, the disk I/O is approximately proportional to the total data read. Moreover, if the same file segment is used to construct k messages, we assume that it needs to be read k times. Denoting the load on the shared link by R_t and the total data read by R_{IO} (both normalized by the file size), the objective is to design an algorithm to minimize the cost

$$R_{\text{cost}}(\alpha, \mathbf{d}) = \alpha R_t(\alpha, \mathbf{d}) + (1 - \alpha) R_{IO}(\alpha, \mathbf{d}), \tag{3.1}$$

where $\alpha \in [0, 1]$ is the trade-off coefficient. Although this chapter will focus on $R_{\text{cost}}(\alpha, \mathbf{d})$, the proposed algorithms could be easily applied to other cost functions. Also, we denote the expected cost over all users' requests as

$$\overline{R_{\text{cost}}(\alpha)} = \alpha E_{\mathbf{d}}[R_t(\alpha, \mathbf{d})] + (1 - \alpha) E_{\mathbf{d}}[R_{IO}(\alpha, \mathbf{d})]. \tag{3.2}$$

3.2.2 Uncoded scheme

In the uncoded scheme, every user caches the same M/N fraction of each of the N files. In the delivery phase, the server sends plain missing segments to all users. Since

each data segment is read once and all of those segments need to be transmitted, the normalized I/O R_{IO}^u is identical to the normalized traffic load of the shared link R_t^u:

$$R_{IO}^u(\mathbf{d}) = R_t^u(\mathbf{d}) = N_e(\mathbf{d})g, \tag{3.3}$$

where $g = 1 - \frac{M}{N}$ is the local caching gain. Each file is requested with probability $p_r = 1 - (1 - \frac{1}{N})^K$, so

$$\overline{R_{cost}^u} = \overline{R_t^u} = \overline{R_{IO}^u} = Np_rg. \tag{3.4}$$

Lemma 3.2.1 *The uncoded scheme is optimal in terms of expected data read among all schemes with uncoded pre-fetching, i.e. , for any other scheme with uncoded prefetching* $\overline{R_{IO}}$ *will be greater than that in Eq. (3.4).*

Proof Denote m_{ij} the fraction of file j being cached by user i, for $j = 1, \ldots, N$ and $i = 1, \ldots, K$. Given a list of demands $\mathbf{d}$, let $\mathcal{U} = \{u_1, \ldots, u_{N_e(\mathbf{d})}\}$ be an arbitrary subset of users requesting distinct files $\{f_1, \ldots, f_{N_e(\mathbf{d})}\}$. Then

$$R_{IO}(\mathbf{d}) \geq \sum_{i=1}^{N_e(\mathbf{d})} (1 - m_{u_i f_i}), \tag{3.5}$$

since the total data read cannot be lower than that delivered.

Each user $u_i \in \mathcal{U}$ has probability $\frac{1}{N}$ of requesting file j, so the average I/O for user u_i is at least $\frac{1}{N}\sum_{j=1}^{N}(1 - m_{u_i j})$. Since $\sum_{j=1}^{N} m_{u_i j} = M$, the average I/O for u_i is at least $\frac{1}{N}(N - M) = g$. Combined with Eq. (3.5), $\overline{R_{IO}}$ is bounded by:

$$\overline{R_{IO}} \geq E_{\mathbf{d}}[N_e(\mathbf{d})g] = Np_rg. \tag{3.6}$$

■

According to Lemma 3.2.1, the uncoded scheme achieves the best average I/O performance with uncoded prefetching! However, the I/O could be even lower with coded prefetching [45]. For example, consider a system with 2 files (A, B), 2 users, and cache size $M = \frac{1}{2}$. The uncoded scheme yields $\overline{R_{IO}^u} = 1.125$, but dividing each file into 2 segments of the same size (A_1, A_2, B_1, B_2) and caching $A_i \oplus B_i$ at user i $(i = 1, 2)$

would only require $R_{IO} = 1$, regardless of the requests (*e.g.* , if user 1 requests A and user 2 requests B, then the server only needs to transmit A_2, B_1). However, the I/O for coded prefetching is a complex problem beyond the scope of this thesis. Instead, we focus our discussion on uncoded prefetching.

3.2.3 Coded scheme

The centralized coded caching scheme proposed by Maddah-Ali and Niesen [1] splits each file into $\binom{K}{t}$ nonoverlapping segments of equal size, with $t = \frac{KM}{N}$, and caches each segment in a distinct group of t users. In the delivery phase, the server sends one message to each subset of $t+1$ users, for a total of $\binom{K}{t+1}$ messages. Each message is composed as the XOR of the $t+1$ segments requested by one user and cached by the others. Each user can then cancel out the segments that it already has in its cache to recover the desired segment. This algorithm has the best normalized traffic load in the worst case, *i.e.* when all users request distinct files. When $N \geq K$, the normalized rate R_t^m is

$$R_t^m = \binom{K}{t+1} / \binom{K}{t} = Kg/(1+KM/N), \tag{3.7}$$

where $g = 1 - \frac{M}{N}$. Each message is the XOR of $t+1$ segments, thus the normalized I/O is $R_{IO}^m = Kg$.

Yu's scheme and our own research [6, 11] extended this work to the case with redundant requests and more general values of N and K. It uses the same placement as [1]. As for the delivery, the server picks $N_e(\mathbf{d})$ "leader" users requesting distinct files and only sends messages to subsets of $t+1$ users containing at least 1 leader. The corresponding rate R_t^c is:

$$R_t^c(\mathbf{d}, t) = \frac{\binom{K}{t+1} - \binom{K-N_e(\mathbf{d})}{t+1}}{\binom{K}{t}}. \tag{3.8}$$

This extension is shown to achieve the best average traffic load with uncoded prefetching. Since each message is the XOR of $t+1$ segments, the average cost R_{cost}^c is:

$$R_{\text{cost}}^c(\alpha, t) = E_{\mathbf{d}}[\alpha R_t^c(\mathbf{d}, t) + (1-\alpha)(1+t)R_{IO}^c(\mathbf{d}, t)]. \tag{3.9}$$

Further extensions proposed a decentralized version of the algorithm [11,40] without coordination in the content placement: users randomly cache a subset of $\frac{MF}{N}$ bits from every file. The delivery phase takes a K-step greedy approach: for bits which are stored in exactly i users ($i = 0, \ldots, K-1$), it constructs messages by XORing $i+1$ segments, similarly to the centralized scheme.

Both M-N and Yu's schemes have optimal I/O in the worst case (*i.e.* , no repeated requests), given by Lemma 3.2.1 ($N_e(\mathbf{d}) = K$ in Eq.(3.6)). Moreover, both schemes have the best peak traffic load (hence the best worst case $R_{\text{cost}}(\alpha)$) in the literature. They are the basis for the algorithms proposed in this chapter. It is therefore recommended that readers have a clear understanding of both M-N and Yu's schemes before proceeding.

3.3 General Algorithms

In this section, we propose algorithms aiming at minimizing the cost functions in Eq.(3.1) and Eq.(3.2). Subsection 3.3.1 introduces an algorithm with the same placement as M-N and Yu's scheme, to maintain optimal performance in the worst case, and an adaptive delivery algorithm to further reduce $R_{\text{cost}}(\mathbf{d})$ in the case with redundant requests. Subsection 3.3.2 sacrifices worst case performance to improve it in the average case. It introduces a new placement algorithm that yields a lower average $\overline{R_{\text{cost}}}$ than both the coded and uncoded schemes.

3.3.1 Adaptive delivery

As mentioned in section 3.2.3, the coded caching scheme has the best R_{cost} in the worst case. However, it could be further reduced when some requests are redundant by sending some segments uncoded. Take the following case as an example.

Example 3.3.1 *Consider a server with* 4 *files (denoted A, B, C and D),* 4 *users with a normalized cache size* $M = 2$*, and a trade-off parameter* $\alpha = 0.2$*. In the placement phase, file A is split into* 6 *segments (denoted* $A_1, A_2, \ldots, A_6$*) and each segment is*

Table 3.1.: Mapping of file segments to user caches. Each cache stores the same three segments for every file, marked with X ($K = 4$, $N = 4$, $M = 2$).

user\ segment	1	2	3	4	5	6	request
1	X	X	X				C
2	X			X	X		B
3		X		X		X	A
4			X		X	X	A

cached by 2 *users. The same goes for files* B, C, D*. Table 3.1 indicates the indices of the 3 segments that each user stores, assumed to be the same for all files without loss of generality. Let the requests be* C, B, A, A*.*

In the delivery phase, we can easily derive that $R^c_{cost} = 1.73$ *according to Eq.(3.9). In this example, we notice that* A_1 *is needed by users* 3 *and* 4*. The messages containing* A_1 *are* $A_1 \oplus B_2 \oplus C_4$ *and* $A_1 \oplus B_3 \oplus C_5$*. If we transmit* A_1 *uncoded along with* $B_2 \oplus C_4$ *and* $B_3 \oplus C_5$ *instead, the users are still able to recover the requested files. The traffic load is higher but the I/O is reduced. The resulting* $R_{\text{cost}} = 1.63$ *is better than both M-N and Yu's schemes.*

The general algorithm is shown in the following lemma.

Lemma 3.3.1 *If a segment is requested by more than* $\frac{1}{1-\alpha}$ *users, then* $R_{\text{cost}}(\alpha, \mathbf{d})$ *can be reduced by transmitting it uncoded.*

Proof If a segment is requested by j users ($j = 1, \ldots, K$), transmitting it uncoded increases R_t to $R'_t = R_t + 1/\binom{K}{t}$ and decreases R_{IO} to $R'_{IO} = R_{IO} - (j-1)/\binom{K}{t}$. Therefore, $R'_{\text{cost}} = \alpha R'_t + (1-\alpha)R'_{IO}$ as defined in Eq.(3.1) is lower than R_{cost} when $j > \frac{1}{1-\alpha}$. ■

In the worst case, each segment is requested by only one user. Our adaptive delivery scheme is then identical to M-N and Yu's scheme, ensuring an optimal R_{cost} in the worst case. This adaptive delivery algorithm can also be easily extended to

the decentralized scheme in [11, 43], following the same principle: the bits which are requested by more than $\frac{1}{1-\alpha}$ users are sent uncoded and all the others coded. Fig. 3.2 compares the average $\overline{R_{\text{cost}}}$ for M-N scheme, Yu's scheme and the adaptive algorithm proposed in this chapter both for the centralized and decentralized settings. It shows that, when α is small, the adaptive algorithm has a much better performance than M-N scheme and Yu's scheme both in the centralized and decentralized cases.

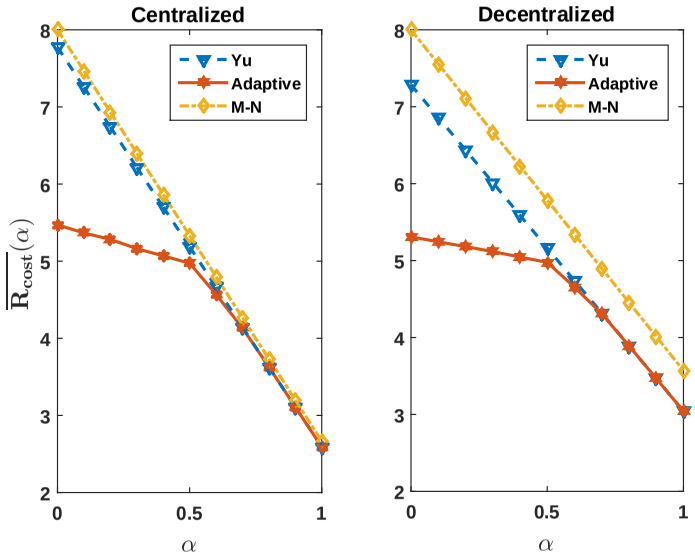


Figure 3.2.: Comparison of adaptive and coded schemes for varying trade-off coefficient α ($K = 10, N = 10, M = 2$).

3.3.2 Partial Caching

The adaptive delivery scheme used the same placement as [1] to ensure optimal R_{cost} in the worst case. This subsection proposes an algorithm seeking a lower average $\overline{R_{\text{cost}}(\alpha)}$ as defined in Eq.(3.2), at the cost of suboptimal worst case performance. Intuitively, when α is very small, $\overline{R_{\text{cost}}(\alpha)}$ mainly depends on I/O, so we should transmit all the requested segments uncoded; vice versa, when α is close to 1, we should transmit all the requested segments coded. Inspired by this fact, if only a

portion $p \in [0,1]$ of every file is cached at the users and transmitted coded, while the rest is always transmitted uncoded, we expect a better average performance for intermediate values of α.

The general algorithm is as follows. In the placement phase, we choose a fraction p of each file to be cached at the users' end. This portion is divided into $t' = \frac{KM}{Np}$ segments and the rest (of size $(1-p)F$) is not cached. In the delivery phase, the cached part is transmitted coded using Yu's scheme and the uncached portion is transmitted uncoded. The fraction p is optimized to minimize $\overline{R_{\text{cost}}}$:

$$p = \arg\min_{p} \left((1-p)R^{u}_{\text{cost}}(\alpha) + pR^{c}_{\text{cost}}(\alpha, t')\right),$$

where $t' = \frac{KM}{Np}$, R^{u}_{cost} is defined in Eq.(3.4) and R^{c}_{cost} is defined in Eq.(3.9). This algorithm can be easily extended to the decentralized case by caching a random portion p of each file at each user, employing the decentralized transmission strategy mentioned in section 3.2.3 for these portions, and transmitting the uncached portions uncoded.

The coded, uncoded and partial caching schemes are compared in Fig. 3.3 for both the centralized and decentralized cases. When α is small, the partial caching scheme takes $p = 0$ and transmits all the segments uncoded; vice versa, when α is close to 1, the algorithm takes $p = 1$ and it is equivalent to Yu's scheme. For intermediate values of α, partial caching offers lower $\overline{R_{\text{cost}}}$ than both the coded and uncoded schemes.

3.4 Simulations

This section compares the proposed algorithms with traditional schemes through simulations. It fixes the number of files as $N = 8$ and studies the trade-off between traffic load and I/O by varying the cache size M and the number of users K.

Fig. 3.4 compares the performance of the adaptive scheme and Yu's scheme with trade-off parameter $\alpha = 0.3$ when the cache size M changes. It shows that the adaptive scheme has an advantage over Yu's scheme in terms of $\overline{R_{\text{cost}}}$ when M is small, but the gap closes as M increases. Moreover, the figure presents results for

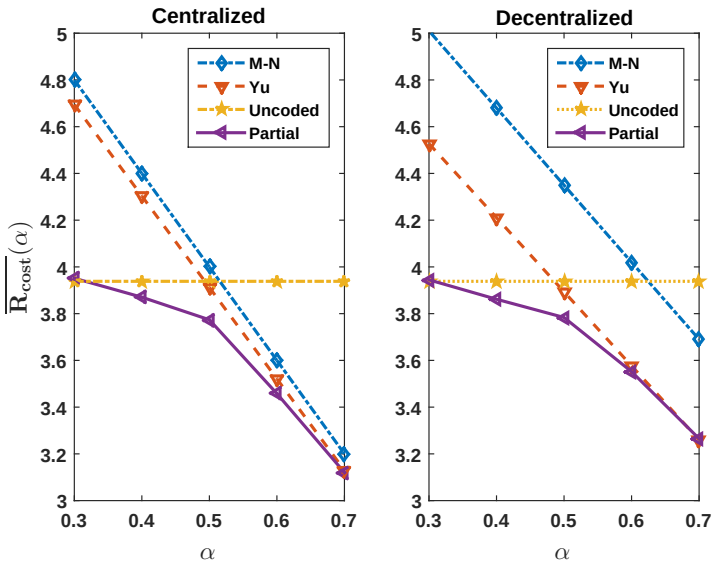


Figure 3.3.: Comparison of partial caching, coded and uncoded schemes for varying α ($K = 8, N = 8, M = 2$).

4 and 8 users, showing that the gains are more prominent as the number of users increases. This is because both small cache size and more users increase the chance that a segment is requested by multiple users.

Fig. 3.5 compares the performance of the coded, uncoded and partial caching schemes for different number of users. As mentioned in section 3.3.2, when the trade-off parameter α is small, we prefer to use the uncoded scheme to minimize I/O. As α increases, the portion p of each file that is transmitted coded also increases. The threshold α for which p is no longer 0 is bigger for the system with 8 users than for the system with 4 users. This is because when there are more users, the probability that some users request the same file increases, which benefits the uncoded transmission.

3.5 Summary

This chapter proposes algorithms to study the trade-off between traffic load and I/O for coded caching in both the centralized and decentralized settings. Reading a

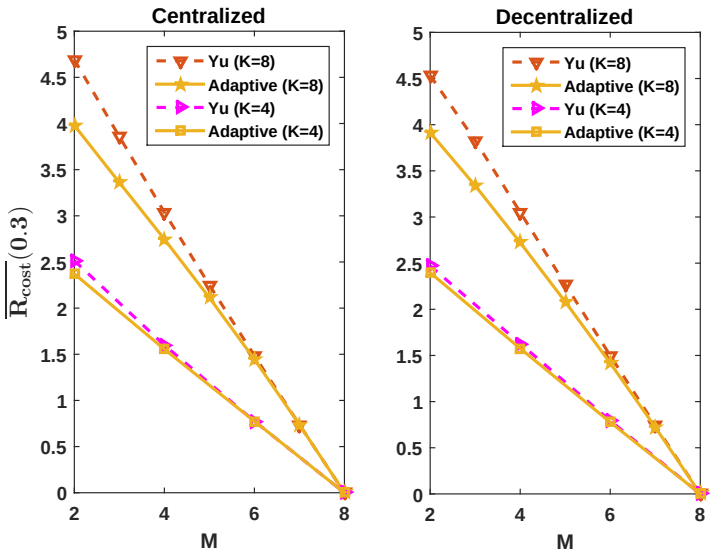


Figure 3.4.: Comparison of adaptive and coded schemes for varying cache capacity and users ($N = 8, \alpha = 0.3$).

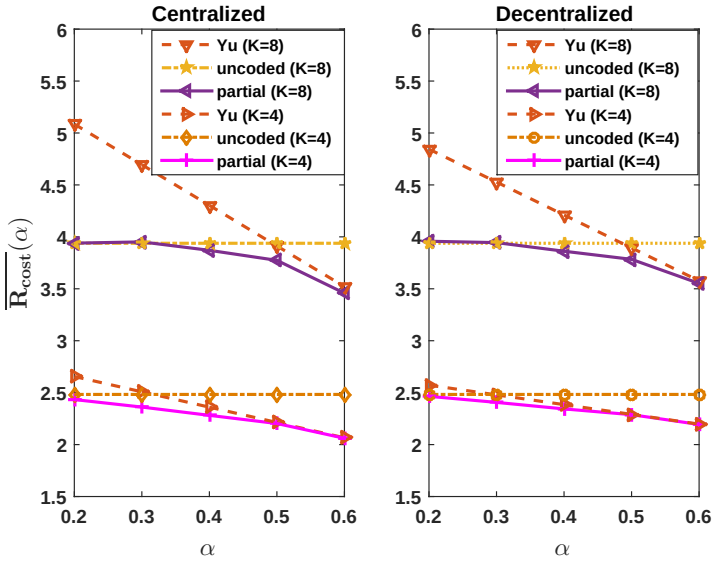


Figure 3.5.: Comparison of partial caching, coded and uncoded schemes for different number of users ($N = 8, M = 2$).

file segment multiple times to compose different messages can be suboptimal when I/O is considered. The proposed algorithms strike a balance between coded and

uncoded transmissions, showing better performance than traditional schemes both in the worst case and the average case.

Besides network protocols, as we studied in Chapter 2 and Chapter 3, storage hardware is another constraint which limits the system performance. In next two chapters, we will focus on another storage system, non-volatile memories, and study how to utilize signal processing approaches to improve the reliability.

4. SIGNAL PROCESSING FOR NAND FLASH MEMORIES

4.1 Introduction

NAND Flash is a non-volatile memory technology which offers significantly higher speeds and power efficiency than hard drives, but its higher cost is still an obstacle for its widespread use. In order to reduce the cost, manufacturers are scaling the technology and trying to pack more bits in each cell. One of the main problems that NAND flash memories are facing today is the reliability of the stored information [46].

A flash cell is a floating gate transistor whose threshold voltage can be adjusted by injecting charges into its floating gate. Information is stored by setting this voltage threshold to specific values. In its simplest form, one bit is stored in each cell, depending on whether it is charged or discharged. Memories of this type are known as SLC. In order to increase the capacity (and reduce their cost accordingly) most applications now use MLC memories, which can be programmed to four different voltage levels and store two bits in each cell. Some manufacturers have gone even further, producing memories which store three (TLC) or even four bits in each cell [47, 48].

As Flash memory technology scales and more bits are stored in each cell, the signal to noise ratio observed in the programmed voltages decreases. One of the main sources of noise, which is becoming increasingly important as the technology scales and is expected to get even worse for the forthcoming 3D flash structures, is inter-cell interference (ICI) [13, 49]. The shift in threshold voltage of one cell can change the threshold voltage of its neighbors due to the parasitic capacitance-coupling effect [50]. Extensive measurements have shown that the ICI noise created by a cell is proportional to the voltage to which it is being programmed [51]. Other sources of noise include Gaussian noise, caused by overprogramming and charge leakage, and impulse noise, caused by defective or broken cells [52].

Additionally, flash cells have a limited lifetime. Before data can be written to a page[1], the block must have been erased (i.e., all the cells need to be discharged). The tunneling of charges into and out of the floating gate causes damage to the dielectric barrier that holds the charges, limiting the range of programmed voltages and the number of times that each cell can be written. The amount of damage that a cell suffers in a single write operation increases super-linearly with the programmed voltage [14]. Hence, writing data patterns that are represented by a lower threshold voltage could prolong the lifetime of the flash [53–56].

This chapter proposes two new signal processing methods. The definitions for page, block, and read threshold will be given in Section 4.2, together with some necessary background on NAND flash memories. Then Section 4.3 studies a new read method, which we call multi-page read, that can help alleviate some of the challenges that the flash memory industry is facing. A multi-page read operation selects multiple pages in a block, biases them with different read thresholds, and returns a combination of their stored information. Section 4.4 proposes a new data representation scheme which increases endurance and significantly reduces the probability of error caused by inter-cell-interference. The method is based on using an orthogonal code to spread each bit across multiple cells, resulting in lower variance for the voltages being programmed in the cells. This new data representation method is also shown to present many of the advantages that spreading sequences bring to wireless communications. For example, multiple information sequences can be written on the same cells at different times without interfering with each other. It also allows storing additional information on an already programmed memory in such a way that the new information is hidden by the noise. At last, Section 4.5 summaries the whole chapter. The results of this chapter are published in [17–19].

[1]Cells in a NAND flash are grouped into pages, which is the smallest unit for write and read operations. Pages are grouped into blocks, which is the elementary unit for erase operations

4.2 Background

A flash cell, illustrated in Fig. 4.1, is a floating gate transistor whose threshold voltage can be adjusted by Fowler-Nordheim (FN) tunneling [57] of charge into or out of the floating gate. If the control gate voltage is greater than this threshold, the cell opens a channel between the drain and the source and we say that the cell *conducts*. Otherwise, the cell acts as an open circuit and the cell does not conduct. NAND flash memories organize cells in array structures known as blocks, like the one

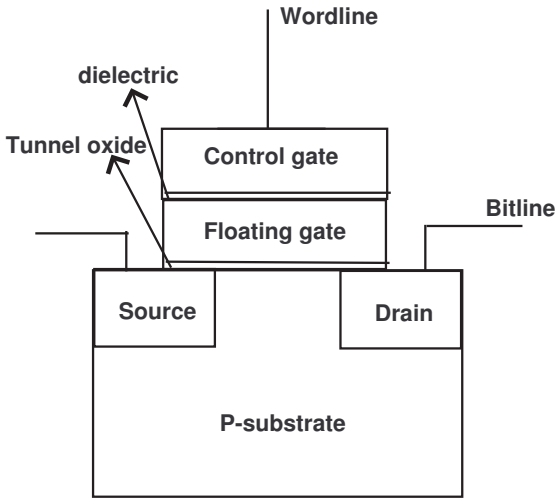


Figure 4.1.: Floating gate transistor structure.

shown in Fig. 4.2. We refer to each row of cells in a block as a wordline and to each column as a bitline. A page is a logical structure that includes one bit from each cell in a wordline. SLC memories have one page per wordline, MLC memories have two, and TLC have three. Blocks are the elementary unit for erase operations, but reads and writes can be done at a page granularity. This wordline-bitline structure allows programming or reading all the cells in a page in parallel, as described below.

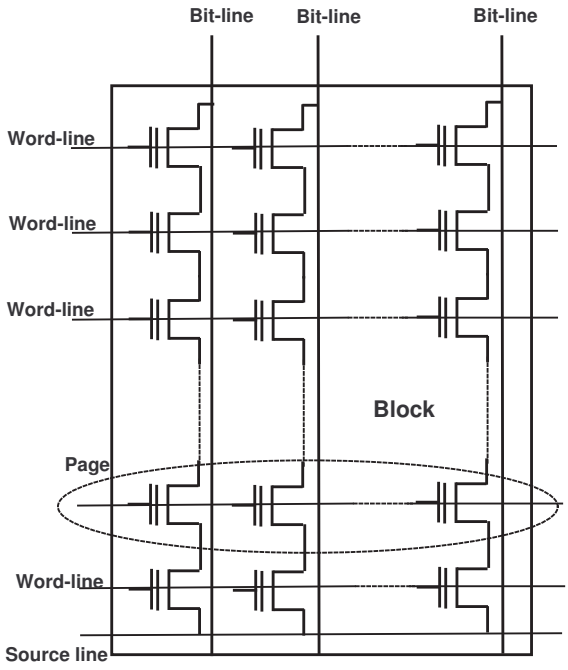


Figure 4.2.: Bitline-Wordline structure of NAND flash memory.

Program operation

The programming is done by sending high voltage pulses into one wordline and biasing all other wordlines so that their cells conduct. Cells in the selected wordline with grounded bitlines experience a high electric field across the floating gate and the substrate, triggering the FN tunneling. After each pulse, a verify read is performed and cells which have reached the desired level of charge are inhibited from further programming. This can be done by biasing their bitlines to a high voltage. This programming method is called ISPP algorithm [58]. For MLC cells, the programming includes two stages: the LSB programming leaves the cell either erased or at half its maximum charge, and the MSB programming does the fine adjustment of the final voltage. The amount of electrons injected into the floating gate is determined by both the LSB and MSB bit values [59].

Erase operation

The erasing of a block follows an similar process to the programming, but using negative pulses to remove charges from the floating gate instead. Additionally, stronger and fewer pulses are used since there is little harm in over-erasing the cells. Individual pages cannot be erased independently because a dielectric breakdown may occur due to the interference between wordlines [60].

Read operation

The voltage threshold of the cells cannot be read directly, it can only be compared with an adjustable reference (read voltage). Pages are read by biasing one wordline with this read voltage l while all others are set to a high voltage V_{pass} ($l \ll V_{\text{pass}}$) so that their cells conduct. Cells in the selected wordline whose threshold voltage is below the read voltage l also conduct, causing the discharge of a capacitor through the bitline, whereas cells with higher threshold voltage act as an open circuit, not letting the current through. By sensing which capacitors got discharged, many bitlines can be read in parallel.

4.3 Multi-page Read for NAND Flash

This section explains the multi-page read method and it is structured as follows: Subsection 4.3.1 describes the multi-page read and explains how it should be implemented. Then, Subsection 4.3.2 provides examples where the multi-page read can help improve the reliability, speed, and endurance of NAND flash memories.

4.3.1 Multi-page Read Method

The read operation described above provides one bit of information about each cell in the selected wordline. If a bitline conducts, it means that the cell's threshold voltage is below the read voltage. The corresponding bit is then read as "0". If a

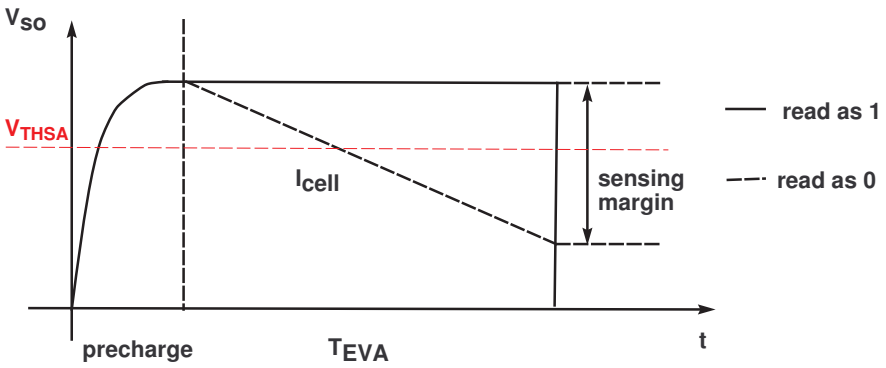


Figure 4.3.: ABL read operation timing diagram.

bitline does not conduct, it means that the cell's threshold is above the read voltage and the corresponding bit is then read as "1". However, it is important to understand that the bit values depend on the read threshold: the same page can yield different bit values for different read thresholds[2].

From the perspective of sensing circuits, there exist multiple read architectures, all of which use capacitances to integrate the bitline current. Most modern NAND Flash memories use the All Bitline (ABL) architecture [61] shown in Fig. 4.4, which includes a dedicated capacitor C_{SO} and keeps the bitline voltage constant during the evaluation phase. Fig. 4.3 shows the three phases in a read operation. First, C_{SO} is pre-charged to a high voltage V_{DD}. Then M_{PCH} is shut off and conducting bitlines experience a constant current I_{cell} that discharges C_{SO} (evaluation phase). After T_{EVA} seconds, the capacitor voltage is compared with a reference V_{THSA} and the read result is output through a latch [62]. The total read time is dominated by the evaluation time T_{EVA}, which can be represented as:

$$T_{\mathrm{EVA}} = \frac{(V_{\mathrm{DD}} - V_{\mathrm{THSA}})C_{\mathrm{SO}}}{I_{\mathrm{cell}}}. \tag{4.1}$$

[2]Some manufacturers use the reverse bit labels, "1" to denote conducting and "0" not conducting. This convention makes no difference towards our results, but OR operations should be replaced with AND.

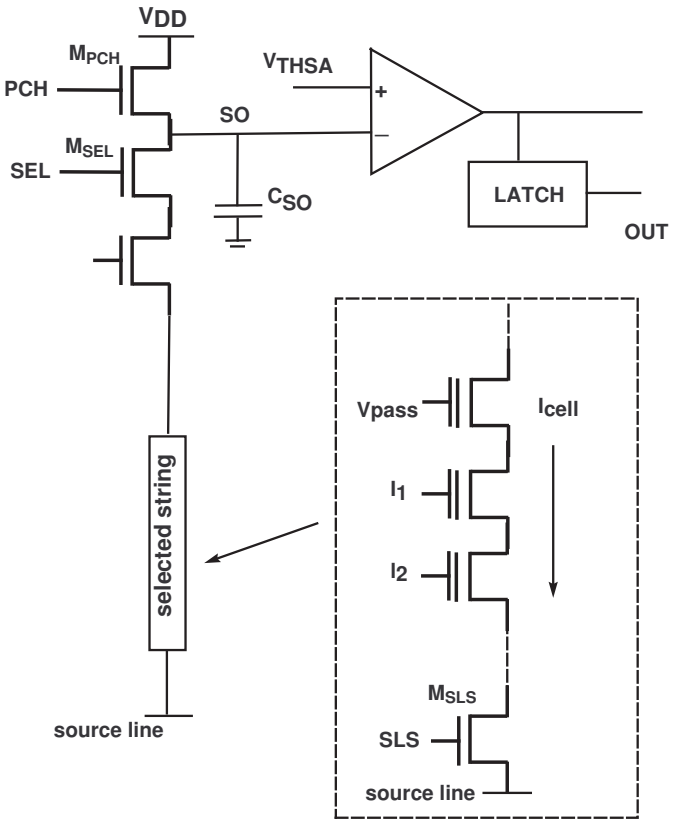


Figure 4.4.: ABL sense circuits for NAND flash.

The current through a MOS transistor operating in ohmic region I_{cell} with small drain to source voltage V_{DS} can be approximated by:

$$I_{\text{cell}} = k\left[(V_{\text{GS}} - V_{\text{TH}})V_{\text{DS}}\right], \tag{4.2}$$

where V_{GS} and V_{TH} respectively represent the gate-to-source and threshold voltages and k is a scaling parameter [62]. Hence, the equivalent resistance R_{oh} for the transistor working in the Ohmic region is:

$$R_{\text{oh}} = \frac{V_{\text{DS}}}{I_{\text{cell}}} = \frac{1}{k(V_{\text{GS}} - V_{\text{TH}})}. \tag{4.3}$$

The multi-page read proposed in this section uses the same components and read methodology as the ABL architecture, but instead of biasing a single wordline with a read threshold and all others with V_{pass}, multiple wordlines are biased with different

read thresholds $\{l_1, l_2, \ldots\}$ while the rest are kept at V_{pass} as shown in Fig. 4.4. A bitline will conduct only when all the selected cells have lower voltage than the corresponding read thresholds. Since we are using value "1" to denote "not conducting", this is equivalent to a bit-wise OR operation of all the selected wordlines.

The main problem of the ABL architecture is the static current consumption during the pre-charge phase, specially by cells with threshold voltage much smaller than the read voltage as indicated by Eq. (4.2). With multi-page read, however, fewer bitlines will conduct and those that do will only draw strong currents if **all** the read voltages are much larger than the corresponding thresholds. Hence, multi-page read helps alleviate the power consumption problem.

Bitlines that do conduct will experience a read current very similar to that in regular reads. Each bitline has hundreds of cells connected in series whose equivalent resistance is determined by the gap between the read and threshold voltages according to Eq. (4.3). This gap is smaller for cells being read than for those biased at V_{pass}, but both are usually in the same order of magnitude. The equivalent resistance of the whole string is then dominated by the hundreds of cells acting as pass transistors, not by the few being read. If the read and threshold voltages are very close for a cell, it would reach the saturation mode, thereby limiting the current independently of V_{DS}. Since I_{cell} in Eq. (4.1) does not suffer a significant decrease in either case, we can conclude that the evaluation time T_{EVA} in a multi-page read is similar to that in a regular read, offering comparable read speeds.

Additionally, the multi-page read method can be applied to improve several applications of flash memories as we will discuss in Section 4.3.2.

4.3.2 Applications for Multi-page Read

ICI Equalization

The ISPP programming algorithm can compensate for the inter-cell interference caused by previously programmed wordlines, but not for the interference of subsequent

write operations. Since wordlines are programmed in sequential order, most of the ICI suffered by a specific wordline is caused by the direct-above-neighbor. Extensive measurements have shown that the change in threshold voltage suffered by the victim cell is proportional to the threshold voltage of the aggressor cell, with a proportionally factor γ that depends on the parasitic capacitance between the aggressor cell and the victim cell.

The neighbor-cell assisted error correction (NAC) algorithm was proposed in [59] to equalize ICI. The NAC method first performs one read of the aggressor wordline and classifies the cells in the victim wordline as suffering weak or strong ICI depending on the value programmed in their direct above neighbor. Then, it reads the victim wordline with different thresholds, selectively chosing which result to keep for each cell. It effectively reads cells suffering strong ICI with a different threshold as those suffering weak ICI, thereby reducing the probability of error. However, this algorithm requires reading the aggressor wordline and thus reduces the read speed. We propose to use the multi-page read method to read the victim and aggressor wordlines simultaneously.

If we set a read threshold l_{victim} on the desired wordline and an intermediate threshold $l_{\text{aggressor}}$ on its neighbor, while the rest are set to V_{pass}, only bitlines which fulfil both conditions would conduct. This way we can use a single multi-page read to detect the cells which have voltage below l_{victim} **and** are suffering weak ICI. Combining these multi-page reads allows us to obtain similar results to the NAC algorithm.

For example: In MLC memories, each cell can be programmed to four different levels, denoted S_0, S_1, S_2, and S_3. According to the suffered ICI, we further classify them into 8 states: S_0^{weak}, S_0^{strong}, S_1^{weak}, S_1^{strong} ..., as Fig. 4.5 shows. The read thresholds for weak ICI cells are A_1, B_1, C_1 and for strong ICI cells A_2, B_2, C_2. The six proposed reads are listed in Table 4.1.

To classify the cells into S_0, S_1, S_2 and S_3, we combine the results of the six threshold comparisons. State S_i $(i = 0, 1, 2, 3)$ can be represented as (S_i^{weak} OR S_i^{strong}), where S_i^{weak} can be found from the first three reads and S_i^{strong} can be found

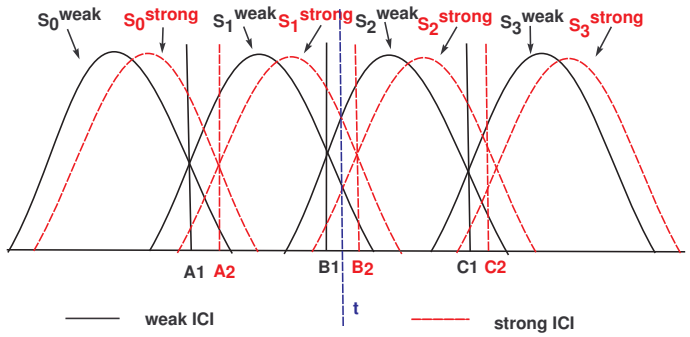
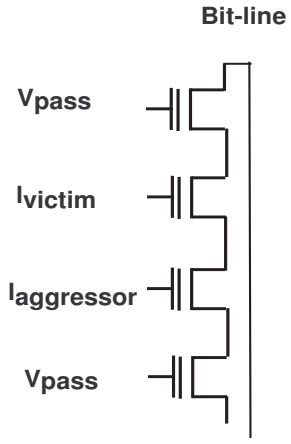


Figure 4.5.: Illustration of multi-page read method for MLC ICI equalization. The curves represent histograms of threshold voltages across a page and vertical lines represent read thresholds.



	l_{victim}	$l_{\text{aggressor}}$
Read_1	A_1	t
Read_2	B_1	t
Read_3	C_1	t
Read_4	A_2	V_{pass}
Read_5	B_2	V_{pass}
Read_6	C_2	V_{pass}

Figure 4.6 & Table 4.1: Bitline illustration & Multi-page reads for MLC ICI equalization.

from the last three reads after eliminating weak ICI cells. For example, we classify a cell as S_1 iff {Read_1=0 and Read_2=1} OR {Read_4=0 and Read_5=1 and Read_3=0}.

This strategy brings slightly more error between S_2 and S_3 than NAC, which performs an additional read on the aggressor wordline. In order to reduce this error C_2 is slightly shifted, as shown in Fig. 4.5. However, the error is minor and this strategy will save one read. Moreover, when ICI from the nearest neighbor is large, this method provides lower BER than any other with reads on a single page ever could.

Fig. 4.7 compares the channel capacity of a regular scheme using six reads to produce soft information [63] with NAC and the multi-page read method for MLC flash memories. It is assumed that S_0, S_1, S_2, S_3 are Gaussian distributed [13] with the same variance $\sigma = 0.15$ and means $0, 1, 2, 3$, respectively. The read thresholds for all three methods were numerically optimized to maximize the resulting capacity. The results show that as γ increases, the NAC and multi-page read method provide significantly better performance. In fact, when ICI dominates over the Gaussian noise, the proposed method would always provide higher capacity than reading the desired page alone, regardless of how many reads the latter employs. This is due to the fact that the multi-page read method provides some amount of equalization for the channel. The figure also shows that the performance of multi-page read is very close to that for the NAC method despite it requires one less read operation.

Partial Erase

The erase operation consists of sending a series of voltage pulses into the gates of all the cells in a block, until all the floating gates have been discharged [64]. All cells in the block suffer the same pulses, despite some can be "fast erased" and others need more negative pulses [65]. These pulses damage the dielectric barrier in the cells, increasing BER and shortening their lifetime. This subsection shows how to apply the multi-page read method to reduce the number of erase pulses sent, so that this damage can be reduced and the erase operation can be accelerated.

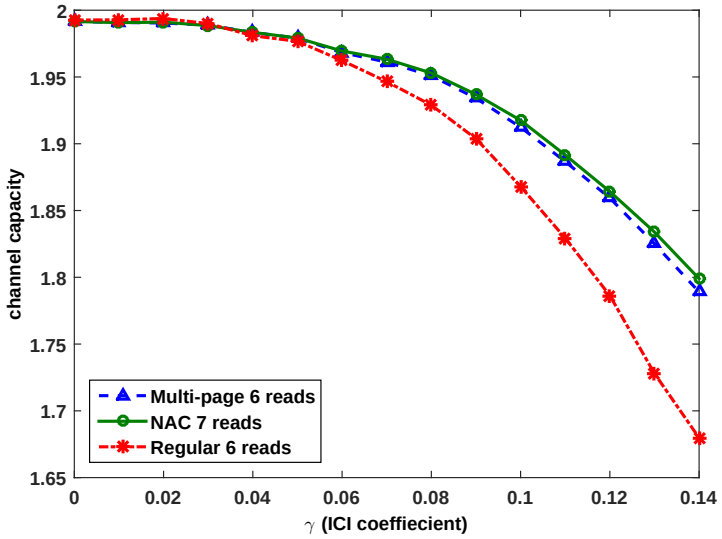


Figure 4.7.: Channel capacity for an MLC cell after 6 traditional reads, 6 multi-page reads, and NAC with 7 reads.

Flash memories generally use a log-structured file system, with a background garbage collection process that keeps a pool of erased blocks ready to be written [66]. As new information arrives, the controller writes it in these blocks, filling one before moving on to the next.

Unlike the traditional erase algorithm, which continues issuing pulses until the whole block is totally erased, we propose a partial-erase option where fewer pulses are sent and a small number of cells remain incompletely erased. By reading all the wordlines simultaneously with a very low read threshold, the controller can detect which bitlines have cells that have not been completely erased, and store their indices as part of the block's metadata. The controller may then chose to skip those bitlines during the writing process, use a constraint code to mask the errors [52], or ignore this information and rely on the ECC block to correct any errors that might arise.

Similar ideas can also be applied to worn-out flash memories. Although it is common to assume a uniform wear among all the cells, not all of them present the same level of tolerance towards program-erase (P/E) cycles. This makes the reuse

of worn-out blocks meaningful. Lab collected data shows that erase errors, which typically trigger the permanent retirement of a block, are caused by a few broken cells that remain unerased, but most of the other cells are healthy. By setting all wordlines to a small read threshold, we can detect which bitlines have broken cells and skip them in subsequent writes.

Unfortunately, the multi-page read step may slow down the erase operation. The gap between the read and threshold voltages could be relatively small for many cells along the bitline, thereby limiting the current and extending the evaluation time. However, the read step still takes much less time than sending the erase pulses. According to [67], the erase operation for one block takes about $500\mu s$ while the read operation takes only several nanoseconds. So the latency brought by the multi-page read step is negligible in the partial erase operation.

WOM Codes

WOM (Write-Once-Memory) codes were designed for memories where bits can change in one direction (*e.g.* , 0 to 1) but not the other [2,68], and they have recently been proposed to allow multiple overwrites of a flash page without erasing [69]. A simple example of a WOM code is shown in Fig. 4.8. This example is using three SLC (binary value) cells to write two bits twice. Initially, all three cells are erased (state 000). The first two bits are written by transitioning to one of the states in the first generation, according to the labels shown in the plot. The second pair of bits, if different from the first, is written by transitioning to one of the states in the second generation. All transitions involve charging, not erasing, the cells so they are feasible.

However, WOM codes present some practical limitations that prevent their adoption by the flash memory industry: they significantly reduce the capacity and speed of the memory. The example shown in Fig. 4.8 provides two bits of information for every three cells read, so page length and read throughput are 33% lower than with a traditional scheme.

In order to address these challenges, we propose aligning the WOM codewords vertically, across wordlines, instead of storing the whole codeword on the same page. The multi-page read allows us to rapidly compute the OR operation of multiple wordlines, accelerating the decoding. Observe that, denoting the state of the cells by $b_1b_2b_3$, the two bits in the first generation are given by "b_2 OR b_3" and "b_1 OR b_3", respectively. A single multi-page read of the last two wordlines would provide the first information bit and a single multi-page read of the first and third wordlines the second information bit, as shown in Fig. 4.9. Each multi-page read provides a sequence of information bits of the same length as a page, so the read throughput is the same as in the traditional scheme.

Unfortunately, a similar idea cannot be applied to the second generation of writes. Some codewords may still be in first generation states so it is necessary to read all three pages individually to obtain two pages of information. This yields the same rate as the regular WOM scheme, which performs one read to obtain $\frac{2}{3}$ of a page of information.

On average, for the example shown in Fig. 4.8, the proposed scheme with multi-page reads would provide an information rate of 0.8 (4 pages of information after 5 reads), which is a significant improvement over the 0.66 rate shown by the usual WOM scheme. Similar approaches might be possible for more advanced WOM codes.

Other Applications

The multi-page read provides a way of obtaining the bit-wise OR (or bit-wise AND, if the discharged state is denoted by 1) of the information stored in multiple wordlines using a single read operation[3]. There are multiple applications that could benefit from such feature: group testing, masking, constrained codes, hash lookups, etc. Instead of reading multiple pages and storing them in registers to perform these

[3]For MLC memories, finding the OR of an MSB page would require two multi-read threshold comparisons, just like in a traditional MSB read.

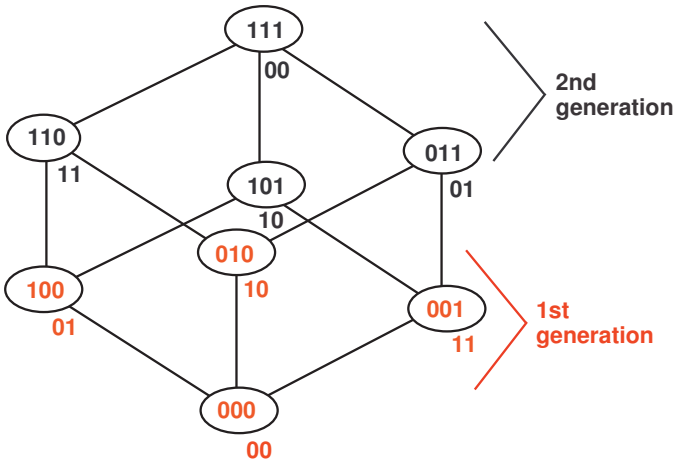


Figure 4.8.: The WOM code on the cube.

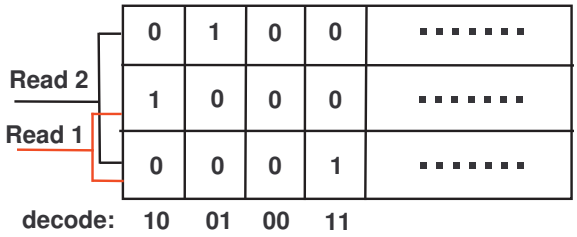


Figure 4.9.: Multi-page read to decode WOM code.

operations, the multi-page read allows us to obtain the result in a fraction of the time by performing a single read.

4.4 Spreading Modulation for NAND Flash Memories

This section will explain a new signal processing approach: the spreading modulation. This section is organized as follows: Subsection 4.4.1 introduces the system model used in the rest of the chapter. Subsection 4.4.2 explains the spreading data representation approach, analyzing its performance under different types of noise, and Section 4.4.3 provides guidelines on how to adjust the spreading parameter. Sec-

tions 4.4.4 and 4.4.5 respectively show how the spreading approach can be used to generate soft information and to hide data in the memory. Finally, Section 4.4.6 presents simulation results to validate the method.

4.4.1 System Model

In order to better illustrate the features of the proposed scheme, this thesis will consider multiple scenarios with different noise distributions and memory types. From a high level perspective, it will be assumed that in a write operation the host provides a vector of (possibly encoded) information symbols $\mathbf{b} \in \mathcal{X}^M$ from an alphabet $\mathcal{X}$, which are then mapped to a vector of voltages $\mathbf{v}^0$ to be programmed on the cells. By the time that the cells are read, the voltages $\mathbf{v}^0$ will have suffered some amount of white Gaussian noise [13, 70], denoted $\mathbf{n}^w$, as well as inter-cell interference (ICI), denoted $\mathbf{n}^{\text{ICI}}$. Therefore, the voltage actually stored in the cells at read time is

$$\mathbf{v} = \mathbf{v}^0 + \mathbf{n}, \qquad \mathbf{n} = \mathbf{n}^w + \mathbf{n}^{\text{ICI}}. \tag{4.4}$$

The noise due to leakage is also assumed to be Gaussian and is therefore absorbed into the $\mathbf{n}^w$ term.

ICI occurs when a shift in the threshold voltage of one cell changes the threshold voltage of its neighbors due to the parasitic capacitance between cells, known as “floating-gate interference" [50]. Extensive measurements have shown that the change in threshold voltage suffered by the victim cell is proportional to the threshold voltage of the aggressor cell, with a proportionality factor that depends on the parasitic capacitance between the aggressor cell and the victim cell. This factor is commonly known as coupling ratio and will be denoted by γ. Hence,

$$\mathbf{n}^{\text{ICI}} = \gamma \mathbf{v}_{\text{aggressor}}. \tag{4.5}$$

With the usual data representation scheme, each symbol b is mapped to a fixed nominal voltage v^0. So, for the sake of simplicity, it will be assumed that they both share the same alphabet $\mathcal{X}$ and $b = v^0$. In SLC memories these symbols are binary,

in MLC they can take four values (representing two bits of information), in TLC they take 8 values (3 bits), etc. In general, the number of levels is chosen to be as large as possible while still avoiding potential overlap between the levels and excessive damage when programming the largest voltage, denoted $V_{\max}$.

Damage to flash memory cells is caused by program/erase (P/E) cycling. According to [14] and the experimental results presented in Section 4.4.6, the damage suffered by a cell when programmed to a voltage V_{th} is approximately proportional to V_{th}^2. Most of the damage happens when cells are programmed to the largest voltage $V_{\max}$, so writing data patterns that are represented by a lower threshold voltage could prolong the lifetime of the flash [14, 53].

The proposed data representation scheme will use a linear mapping between the symbols $\mathbf{b}$ and the nominal voltages $\mathbf{v}^0$, to be described in the next section. This mapping will extend the number of possible voltages to be programmed, but also reduce the number of cells programmed to $V_{\max}$, attenuate the ICI, and increase robustness to impulse noise. This will result in increased capacity and extended lifetime for the memory.

In practice, the discharged state in which the cells are left after being erased sets a lower limit for the range of programmable voltages and the write procedure can only push the cells towards higher voltages. Thus it is not possible to program NAND flash cells with a negative voltage. However, for our derivations it will be useful to assume that the range of programmable voltages is symmetric and the voltages $\mathbf{v^0}$ and symbols $\mathbf{b}$ can take both positive and negative values. SLC cells will therefore have their voltage levels relabeled as -0.5 and $+0.5$, while MLC cells will be assumed to take voltage levels -1.5, -0.5, 0.5, and 1.5. In general, if there are $2S$ symbols in the alphabet $\mathcal{X}$, they will be labeled as $\mathcal{X} = \{\pm 0.5, \pm 1.5, \ldots, \pm(S - 0.5)\}$. The largest symbol in the alphabet will be denoted by $V_{\max} = S - 0.5$. This represents a simple shift of the physical reference system. The rest of the chapter assumes that the symbols b_i and voltages v_i^0 have zero mean.

4.4.2 The Spreading Approach

This section introduces the spreading modulation and then analyzes its performance against three types of noise: Gaussian, ICI, and impulse noise. We first study the trade-off between damage and Gaussian noise. The SNR can be increased by widening the range of programmed voltages, but doing so increases the damage suffered by the cells. Then we study how the proposed modulation can attenuate ICI and impulse noise, respectively.

In a traditional flash memory, each cell stores a fixed number of bits. There is usually some redundant bits introduced by the ECC or RAID schemes but, ultimately, each bit is stored in a specific cell. Cells have a fixed number of voltage levels to which they can be programmed and all the levels are written with the same frequency. This section proposes a new data representation scheme which uses orthogonal codes to spread each bit across multiple cells, similar to DS-CDMA transmission in wireless communications. This data representation scheme reduces the variability of the voltages being programmed in the cells, resulting in improved endurance and additional robustness towards impulse noise and ICI.

Instead of mapping each symbol b_i to a fixed voltage v_i^0, the proposed scheme uses a matrix with orthogonal columns $\mathbf{C}$ (e.g., a Walsh matrix) to map the symbols $\mathbf{b}$ into voltages $\mathbf{v}^0$ to be programmed. For example, when mapping four symbols $\mathbf{b} \in \mathcal{X}^4$ into four cells, the voltages to be programmed are:

$$\begin{bmatrix} v_1^0 \\ v_2^0 \\ v_3^0 \\ v_4^0 \end{bmatrix} = \frac{k}{4} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix},$$

where k is an adjustable parameter that controls the range of voltages being programmed. By scaling k, we can introduce more separation between the programmed

voltage levels, but the damage suffered by the cells and the power consumed would also increase. In general, when M symbols are to be programmed into $N \geq M$ cells,

$$\mathbf{v}^0 = \frac{k}{M}\mathbf{C}\mathbf{b}, \tag{4.6}$$

where $\mathbf{C}$ is a $\{-1, 1\}^{N\times M}$ matrix with orthogonal columns and kV_{max} is the maximum voltage to be programmed (remember that $\mathbf{v}^0$ are not the programmed voltages, but shifted versions of them). By scaling k, we can introduce more separation between the programmed voltage levels, but the noise is not affected by this scaling. We will refer to this operation as spreading.

By spreading each information symbol across multiple cells, we increase the number of possible programmed voltages in each cell, so symbols and nominal voltages no longer share the same alphabet. For example, in the SLC case where $b_i \in \{-0.5, 0.5\}$ and $M = N = 4$, our scheme would have five possible levels for each cell: $v_i^0 \in \{-0.5k, -0.25k, 0, 0.25k, 0.5k\}$. In general, if $V_{\max}$ is the largest symbol in the alphabet $\mathcal{X}$, the voltage levels after spreading are in the range $[-kV_{\max}, kV_{\max}]$.

When the read operation is performed, the voltages are multiplied by a de-spreading matrix $\mathbf{C}^T$, which is the left inverse of the spreading matrix. Because of the properties of Walsh sequences, the de-spreading matrix is the transpose of the spreading matrix. Continuing with the previous example, when $N = M = 4$

$$\begin{bmatrix} \hat{b}_1 \\ \hat{b}_2 \\ \hat{b}_3 \\ \hat{b}_4 \end{bmatrix} = \frac{1}{k} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix},$$

where $\hat{b}_i,\ i = 1, 2, 3, 4$ represent the information estimates after reading. In general,

$$\hat{\mathbf{b}} = \frac{M}{Nk}\mathbf{C}^T\mathbf{v}, \tag{4.7}$$

The processes of spreading and de-spreading discussed above are shown in Fig. 4.10.

Figure 4.10.: Illustration of the spreading approach.

Combining Eqs. (4.4), (4.6), and (4.7), the estimated information symbols can be represented as:

$$\hat{b_i} = b_i + \frac{M}{Nk} \sum_{j=1}^{N} \pm n_j, \quad i = 1, 2, \ldots, M. \tag{4.8}$$

The noise can be arbitrarily attenuated by decreasing $\frac{M}{Nk}$, but that involves sacrificing capacity by decreasing $\frac{M}{N}$ or using a wider range of programmed voltages by increasing k. Since most practical applications are not willing to compromise capacity, the rest of the chapter assumes $M = N$, which means that the storage space is the same as in the regular scheme.

Gaussian Noise and Damage

For fixed voltage range ($k = 1$) and signal-independent Gaussian noise, spreading actually decreases the signal-to-noise ratio (SNR) at read time. Assuming independent and identically distributed noise components $n_i \sim \mathcal{N}(0, \sigma^2)$ [70], the SNR of the regular and spreading schemes are:

$$SNR_{\text{regular}} = \frac{P_s}{\sigma^2} \qquad SNR_{\text{spread}} = \frac{P_s}{\frac{N}{k^2}\sigma^2}, \tag{4.9}$$

where $P_s = E[b_i^2]$ represents the power of the stored symbols. It is easy to increase SNR_{spread} when needed by increasing the scaling constant k, but doing so widens the range of programmed voltages and thus causes more damage and consumes more power. This subsection studies such tradeoff.

One of the advantages of the spreading scheme is that it reduces the probability of programming the maximum voltage as shown in Fig. 4.11, thus reducing the damage to the flash memory. The amount of damage suffered by the memory is approximately

proportional to the square of the voltage programmed. As mentioned in Section 4.4.1, cell voltages must be non-negative so in practice they are shifted to $b_i + V_{\max}$ in the regular scheme and to $v_i^0 + kV_{\max}$ in the spreading scheme when $E[b_i] = 0$ and $E[v_i^0] = 0$. Denote T_{spread} and T_{regular} the damage with the spreading and the regular scheme, respectively. Then,

$$\begin{aligned} T_{\text{regular}} &= a \cdot E\left[(b_i + V_{\max})^2\right] \\ &= a(E[b_i^2] + V_{\max}^2), \\ T_{\text{spread}} &= a \cdot E\left[(v_i^0 + kV_{\max})^2\right] \\ &= a\left(\frac{k^2}{N} E[b_i^2] + k^2 V_{\max}^2\right) \end{aligned} \tag{4.10}$$

for some constant a. For $k = 1$ (i.e., both schemes have identical programming range), spreading causes less damage than the regular scheme but it lowers the SNR. For $k = \sqrt{N}$ both schemes have the same SNR, but spreading causes more damage. Section 4.4.3 will elaborate how to choose an optimal k in between.

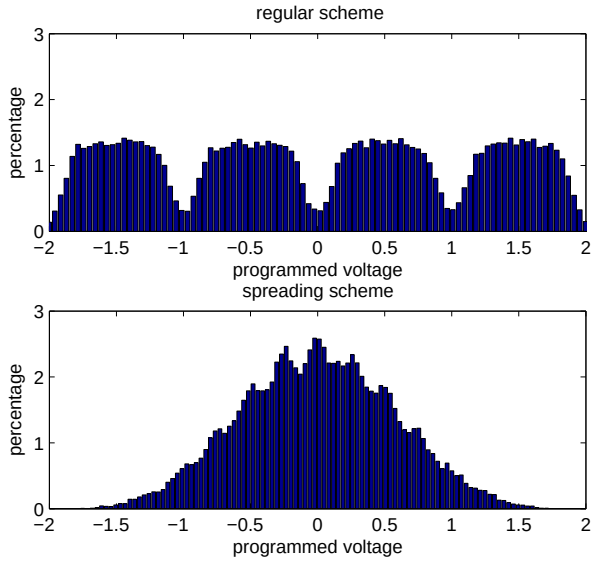


Figure 4.11.: Distribution of cell voltages for both modulation schemes when M=N=4, k=1, $\sigma = 0.1$, $\gamma = 0.2$. Spreading leads to a distribution with less variance.

Inter-cell Interference

The previous section showed that when the noise is independent from the voltages being programmed, our spreading scheme does not provide any improvement in terms of BER unless $k \geq \sqrt{N}$. However, the main source of noise in new memory generations is ICI, which is proportional to the voltages being programmed in the cells.

In most memories, flash cells are organized in an array structure, where all the cells in a wordline are programmed simultaneously and wordlines are programmed in increasing order. The ISPP [58] algorithm used to program wordlines can compensate for the inter-cell interference caused by previously programmed wordlines, but not for the interference of subsequent program operations. Hence, most of the ICI suffered by a specific cell is caused by the direct-neighbor. This will be the only ICI component considered in our analysis, but the simulations in Section 4.4.6 will include 3 neighbors.

Assuming $n^{ICI} \gg n^{w}$ and $M = N$, Eq. (4.8) becomes

$$\hat{b_i} = b_i + \frac{1}{k}\sum_{j=1}^{N} \pm n_j^{\text{ICI}}, \quad i = 1, 2, \ldots, M,$$

where n^{ICI} is proportional to the programmed voltages. According to Eq. (4.5), n^{ICI} can be represented as:

$$n^{\text{ICI}} = \frac{k\gamma}{N}\sum_{j=1}^{N} \pm b_j.$$

So the estimated symbol can be represented as $\hat{b_i} = b_i + \Delta b_{\text{spread}}$, where

$$\Delta b_{\text{spread}} = \frac{\gamma}{N}\sum_{j=1}^{N^2} \pm b_j.$$

If the distance between the symbols is d, errors happen only when $\Delta b_{\text{spread}} \geq \frac{d}{2}$. The distribution of Δb_{spread} is approximately $\mathcal{N}(0, \gamma^2 E[b_i^2])$ when N is large according to

the Central Limit Theorem. Then the probability of error for non-extreme symbols[4] is approximately

$$P_e^{\text{spread}} \approx 2\phi \quad \frac{-d}{2\gamma\sqrt{E[b_i^2]}}\Bigg)\Bigg(\tag{4.11}$$

where $\phi(u) = \int_{-\infty}^{u} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy$. Note that in Eq. (4.11), the scaling parameter k has no effect on ICI.

In the regular scheme, the estimated symbol is:

$$\hat{b_i} = b_i + \gamma b_j,$$

with probability of error for non-extreme symbols

$$P_e^{\text{regular}} = P\left(|b_j| > \frac{d}{2\gamma}\right)\Bigg(\tag{4.12}$$

The main advantages of the proposed spreading scheme comes from the fact that it leads to less variance in the programmed voltages than the regular scheme, as shown in Fig. 4.11. As γ increases, the regular scheme introduces much more probability of error than our spreading scheme. For example, if $\gamma = 0.35$ and $\mathcal{X} = \{\pm 0.5, \pm 1.5\}$ then $d = 1$ and for MLC memories

$$P_{e(MLC)}^{\text{regular}} \simeq 0.25 \qquad P_{e(MLC)}^{\text{spread}} \simeq 0.2, \tag{4.13}$$

for intermediate (non-extreme) symbols when Gaussian noise is negligible according to Eq. (4.12) and Eq. (4.11). The lowest and highest symbol would suffer half as much probability of error in both cases.

As γ increases, P_e^{spread} increases slower than P_e^{regular}. So, when γ is large enough, the spreading scheme has a better performance. It is also important to take into account that the grouping of cells into spreading blocks must be done carefully. If the same cells, say $1-4$, were taken as a spreading block in two consecutive wordlines, the ICI would have the form of a scaled codeword, and would therefore not be attenuated by the de-spreading.

[4]Non-extreme symbols refer to the symbols which are not programmed to the highest or lowest voltage levels.

Impulse Noise

Another important advantage of the spreading approach lies on its increased robustness to impulse noise. Flash memories are currently being used in a wide variety of environments. In most of them they compete with HDD and DRAM but there are some cases in which flash is the only viable option. One of those cases are satellite applications. Hard drives have moving parts, and need a certain air pressure for the head to fly appropriately. DRAM memories are volatile and require frequent refreshing to avoid losing the information. Flash memories, however, are perfect for satellite applications. Their lack of moving parts makes them very compact and shock resistant, and they can be powered off for extended periods of time without losing information.

Satellites suffer a significant amount of radiation, constituting one of the leading causes of electrical component failures [71]. A high energy particle impacting on a NAND flash cell usually causes what is known as a stuck-at defect [52]. The cell effectively breaks and will henceforth be read as storing the same voltage value, regardless of what it was meant to be programmed to. In the regular scheme, any bit written to that cell will most likely be lost. The scheme proposed in this chapter, on the other hand, spreads each bit across multiple cells, and has a chance to recover the bit even if one of the cells is stuck at a given value.

Broken cells can usually be identified before they are read. The ISPP programming mechanism checks the cell voltages after sending each pulse and, when the controller detects that the cell voltage has not changed after having sent multiple pulses, the cell is marked as broken. If this were known *before* the programming started, we could just ignore that cell altogether and not store anything in it. Unfortunately, if the broken cell is detected *during* programming, it is too late to stop the programming of the other cells in the page.

Let p denote the probability of a cell breaking. We assume that the controller knows which cells are broken, and can therefore assign them an arbitrary voltage at

read time, independently from the actual state they are in. In order to minimize the resulting noise, broken cells will be read as having a voltage of 0, the average voltage stored by a healthy cell.

Equation (4.6) shows that the nominal voltage programmed in the i-th cell is $v_i^0 = c_i^T b$, where c_i^T represents the i-th row of the spreading matrix $\mathbf{C}$. If the i-th cell is broken, the controller interprets $v_i = 0$, which is equivalent to replacing i-th column of the de-spreading matrix by zeros when the read operation is performed. Denote by $\hat{\mathbf{C}}$ the matrix $\mathbf{C}$ with the i-th row replaced by zeros, the estimated data symbols can then be represented as:

$$\begin{aligned}\hat{\mathbf{b}} &= \frac{1}{N} \cdot \hat{\mathbf{C}}^{\mathbf{T}} \cdot \mathbf{C} \cdot \mathbf{b} \\ &= \frac{1}{N} \begin{bmatrix} N-1 & \pm 1 & \cdots & \pm 1 \\ \pm 1 & N-1 & \cdots & \pm 1 \\ \vdots & & & \vdots \\ \pm 1 & \pm 1 & \cdots & N-1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix},\end{aligned}$$

where all the noise except impulse noise has been neglected. In other words, the estimated information symbol $\hat{b_i}$ can be represented as:

$$\hat{b_i} = \frac{N-1}{N} b_i + \frac{1}{N} \sum_{j \neq i} \pm b_j \quad i = 1, 2, \ldots, N,$$

where the signs of the error terms depend on the spreading matrix and the signs of the different bits.

In SLC flash memories, an error will occur if the sign of $\hat{b_i}$ is different from the sign of b_i. This can only happen if the signs of the other bits align just right so that the $N-1$ error terms cancel out the correct $\frac{N-1}{N}$ contribution. It happens with probability $\frac{1}{2^{N-1}}$. Moreover, even when the signs align just right to give $\hat{b_i} = 0$, we still have a 50% chance of guessing the sign correctly. So the probability of error due to broken cells is $\frac{N}{2^N} p(1-p)^{N-1} + O(p^2)$.

However, for the regular scheme, whatever bits were stored in the broken cells are completely lost. The ECC will have to recover them if possible. If a cell is broken,

it has a $\frac{1}{2}$ chance of storing the correct value, so the probability of error is $\frac{p}{2}$ for the regular scheme, which is much larger than with the spreading scheme.

In MLC flash memories, however, our scheme no longer offers advantages towards impulse noise. Since there are more programming voltage levels, the error terms may play a more important role because it may contain some large voltage levels. But space applications generally use SLC memories because they are more reliable than MLC.

4.4.3 Choice of Spreading Parameter

Increasing k can improve SNR through noise attenuation, but the range of programmed voltages becomes wider. It was shown in Fig. 4.11 that the probability of programming a very large or small voltage with the spreading scheme is very low, so it could be helpful to increase k and then crop those extremes. If the gains in terms of noise attenuation obtained by increasing k make up for the cropping noise, the overall SNR will increase.

Instead of increasing k and then cropping the largest voltages, our scheme crops both high and low voltages symmetrically, so as to minimize the cropping noise. Assuming that the desired range of programmed voltages is $[-V_{\max}, V_{\max}]$, the quantization noise introduced by cropping is

$$q_i = \begin{cases} 0 & \text{if } v_i^0 \in [-V_{\max}, V_{\max}] \\ -v_i^0 + V_{\max} & \text{if } v_i^0 > V_{\max} \\ -v_i^0 - V_{\max} & \text{if } v_i^0 < -V_{\max}, \end{cases}$$

where $i = 1, 2, \ldots, N$ and v_i^0 is the i-th component of the programmed voltage $\mathbf{v}^0$ defined in Eq.(4.6). The information symbols read can be represented as:

$$\hat{\mathbf{b}} = \mathbf{b} + \frac{1}{k}\mathbf{C}^T\mathbf{n} + \frac{1}{k}\mathbf{C}^T\mathbf{q},$$

where $\mathbf{n}$ is write noise and ICI noise as defined in Eq.(4.4) and $\mathbf{q} = [q_1, q_2, \ldots, q_N]^T$ is the quantization noise.

In other words, for each estimated information value $\hat{b_i}$:

$$\hat{b_i} = b_i + \frac{1}{k}\sum_{j=1}^{N} \pm(n^{\mathrm{ICI}} + n^{w}) + \frac{1}{k}\sum_{j=1}^{N} \pm q_j.$$

To minimize the overload distortion introduced by cropping, we hope to crop only the largest and smallest voltages in our scheme, $\pm kV_{max}$. These levels are programmed with probability $\frac{2}{L^N}$, where L is the number of possible voltage levels, hence q_j can be represented as:

$$q_j = \begin{cases} \pm(k-1)V_{\max} & \text{with probability } \frac{2}{L^N} \\ 0 & \text{with probability } \frac{L^N-2}{L^N} \end{cases}$$

The Gaussian noise, ICI, and quantization noise are uncorrelated, so the total noise power P_N can be found by a simple sum of the components $P_N = P_w + P_{\mathrm{ICI}} + P_q$, where:

$$\begin{aligned} P_w &= \frac{N}{k^2}\sigma^2 \\ P_{\mathrm{ICI}} &= \gamma^2 E[b_i^2] \\ P_q &= \frac{2N(k-1)^2}{k^2 L^N} V_{\max}^2. \end{aligned} \tag{4.14}$$

As k increases the write noise decreases but the quantization noise increases. There is a trade-off between quantization noise and write noise. As shown in Fig. 4.12, the optimal k which minimizes the total noise power and consequently maximizes the SNR is:

$$k^{\star} = \arg\min_{k} \left(\frac{2N(k-1)^2}{k^2 L^N} V_{max}^2 + \frac{N}{k^2}\sigma^2 \right)$$

The scaling parameter k should not be too large, so that the range of the programmed voltage of the spreading approach is close to that of the regular scheme. However, if k is small, the distance between any two adjacent levels will be reduced or compressed. For some memories, it may be hard to control the small voltage increments between the levels in the spreading scheme, specially if k is small. The over-programming could introduce Gaussian noise, but the total power of this noise

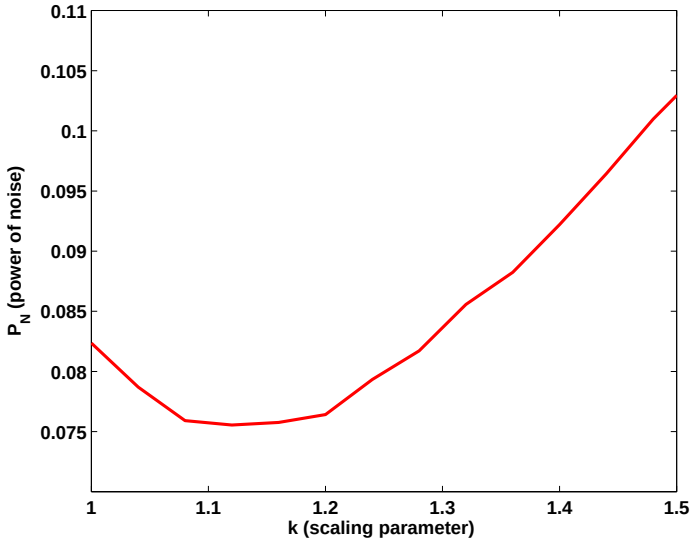


Figure 4.12.: Quantization noise power as a function of k for a SLC flash memory with M=N=4, $\sigma = 0.1$, and $\gamma = 0.2$

would still be lower than that in the regular scheme, since the programming pulses would also be smaller.

In addition to cropping, there are other ways to increase SNR and at the same time maintain the same programming range: we can reassign the programmed voltages to reduce the probability of error. That is, we can increase the distance between the voltage levels with higher probability and decrease the distance between the voltage levels with lower probability. This scheme is more complex than cropping and is suitable for flash memories with high computational capability. We will not discuss it in detail in this chapter.

4.4.4 Obtaining Soft Input

There are two types of decoders in flash: hard-decoders and soft-decoders. The difference between them lies in the input and output dictionary: Hard-decoders usually have the same input and output dictionary which is a fixed set of deterministic

symbols; Soft-decoders operate on log-likelihood ratios (LLR), specifying the probability of each input being a noisy version of each symbol. Soft-decoders can correct more errors, but they require more reads and a more complex decoding algorithm. Some flash controllers use a hard-decoder when BER is low and switch to a soft one when the former one fails [72].

In flash memories, cells are read by comparing their voltage with a number of reference thresholds. If a total of l reads have been performed on a page, each cell can be classified as falling into one of the $l+1$ intervals between the read thresholds. The problem of reliably storing information on the flash is therefore equivalent to the problem of error-free transmission over a Pulse amplitude modulation (PAM) channel [63]. The channel inputs represent the levels to which the cells are written, the outputs represent read intervals, and the channel transition probabilities specify how likely it is for cells programmed to a specific level to be found in each interval at read time [55, 73].

When we perform the minimum required number of reads on a page, cells can only be classified into the nominal symbols. However, if we perform additional reads, we can achieve a finer quantization of the cell voltages. It is then possible to assign an LLR value to each of these voltage intervals. The LLR value associated with a read interval r between level i and level j is defined as $LLR_r = \log(P_{ir}/P_{jr})$, where P_{ab} denotes the transition probability from a to b. A hard decoder takes a greedy approach mapping each interval to the most possible nominal symbol and returning the closest codeword. A soft decoder operates on the LLR values and uses those probabilities to perform a maximum likelihood estimation of the codeword.

As mentioned in section 4.4.2, the spreading approach brings more possible programming voltage levels and requires more reads to distinguish them. This results in a finer quantization of the cell voltages and provides soft inputs to the decoder. This holds even if we reduce the number of reads to be the same as in the regular scheme, since the de-spreading step will combine the read voltages increasing the total number of possible values. For example, conventional SLC memories use a single

read to classify the cells into two states. The channel with the regular scheme is then equivalent to a Binary Symmetric Channel (BSC). In the spreading scheme, however, a single read will still classify each cell into one of two states, but after de-spreading with $N = 4$, each component can take five possible values. The channel observed by each symbol is then equivalent to the PAM channel with five outputs illustrated in Fig. 4.13. As an example, Table 4.2 shows the transition probabilities for Fig. 4.13 when write noise is Gaussian with variance $\sigma = 0.3$ and ICI parameter $\gamma = 0.5$. Soft information plays an important role when noise is large and helps to minimize the probability of error. In our numerical simulation with strong noise in SLC mentioned above(i.e., write noise with $\sigma = 0.3$ and ICI noise with $\gamma = 0.5$), the regular scheme causes probability of error 0.1043 and the spreading scheme causes probability of error 0.0893.

Fig. 4.14 shows the symmetric capacity (i.e., capacity under uniform distribution of inputs) of the channel in the MLC case with write noise $\mathcal{N}(0, 0.1^2)$, as a function of the ICI parameter γ. When the ICI is weak, the regular scheme (using 3 reads) has higher capacity than the spreading one, even when the latter uses 12 reads. However, the capacity of the regular scheme decreases rapidly when the ICI increases, falling below the capacity of the spreading scheme, even when the latter uses 3 reads.

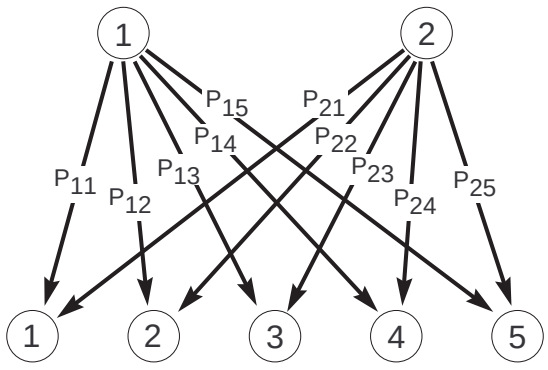


Figure 4.13.: PAM channel equivalent to SLC flash read channel in spreading scheme with $M = N = 4$.

Table 4.2.: Transition probabilities and LLR values for SLC cells.

P_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	0.0555	0.2048	0.1784	0.0578	0.0041
$i=2$	0.0040	0.0580	0.1786	0.2034	0.0555
LLR_j	2.6301	1.2616	-0.0011	-1.2582	-2.6054

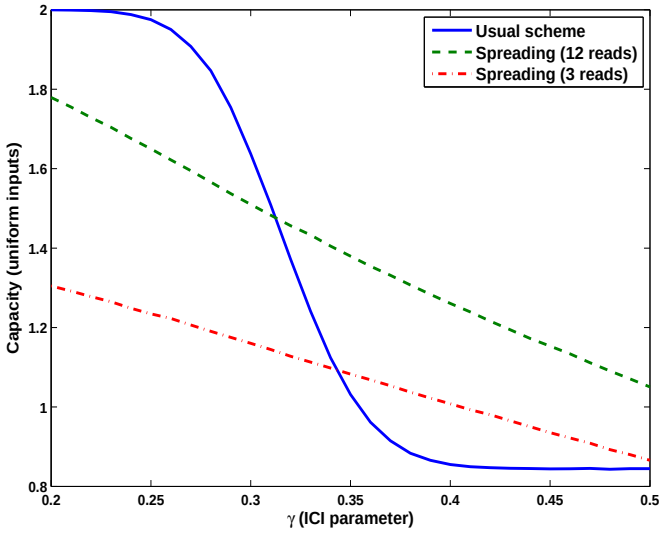


Figure 4.14.: Comparison of the channel capacity of the spreading scheme and regular scheme for MLC flash.

4.4.5 Security

Section 4.4.2 has shown how spreading can be beneficial to reduce the probability of error when ICI is large. This section will show that it can also be used to hide information using a technique known as superposition coding [74]. This technique has been widely used in Direct-sequence spread spectrum (DSSS) communications to make spread-spectrum signals appear wide-band and noise-like, thus making them hard to detect [75].

The key idea of hiding information using superposition coding is making the modulated hidden information look like additional noise. In this chapter, we are going to use a long Pseudo noise (PN) sequence [76] to spread a single symbol of hidden information over many cells. This will create a very long sequence of voltage components, which will be added on top of the original information stored in flash in plain view. The spreading and de-spreading process are described as follows: Denote the spreading sequence (PN sequence) by $\mathbf{d} \in \{+1, -1\}^L$ and a hidden information symbol by h. The voltage components for the hidden information can be represented as

$$\widetilde{\mathbf{v}} = \varepsilon \mathbf{d} h,$$

where ε is a (small) scaling parameter that controls the range of programmed voltages.

The combined voltage $\mathbf{v}^{\mathrm{c}}$ for the original and hidden information is:

$$\mathbf{v}^{\mathrm{c}} = \mathbf{b} + \varepsilon \mathbf{d} h + \mathbf{n},$$

where $\mathbf{b}$ is the vector of L plain view information symbols and $\mathbf{n}$ is the noise as defined in Eq. (4.4).

If the distribution of the combined voltage $\mathbf{v}^{\mathrm{c}}$ still looks similar to the original information to any unauthorized reader, hence making it difficult for them to notice the existence of the hidden information. For example, as shown in Fig. (4.15), the distribution of the combined voltage of the original symbols and hidden information is still similar to the distribution of the original information when we choose the scaling factor ε appropriately. To make the combined voltage as random as possible, we may decrease ε so that the power of hidden information is reduced. However, small ε also brings higher probability of error when decoding the hidden information because write noise will play a comparatively larger influence.

Two steps are required to decode the hidden information. The first step is subtracting the original information. Assuming that we can recover the original infor-

mation with low probability of error, the voltage left after subtracting the original information is approximately:

$$\mathbf{v}^{\mathrm{s}} \approx \varepsilon \mathbf{d} h + \mathbf{n}.$$

The second step is de-spreading using $\mathbf{d}$, the decoded hidden information symbol is:

$$\begin{aligned} \widehat{h} &= \frac{\mathbf{v}^{\mathrm{s}} \mathbf{d}^T}{L\varepsilon} \\ &= h + \sum_{i=1}^{L} \left(\pm \frac{1}{\varepsilon L} n_i, \right. \end{aligned}$$

where L is the length of the spreading sequence.

Assume the write noise to be Gaussian with variance σ^2, the SNR after the de-spreading process is:

$$SNR_{\mathrm{PN}} = \frac{P_s L \varepsilon^2}{\sigma^2}, \tag{4.15}$$

where P_s is the power of each hidden information symbol.

According to Eq. (4.15), another way to increase the accuracy of the recovered hidden information is to increase the length of the spreading sequence L. As shown in Fig. 4.16, P_e^{hidden} is lower for the same P_e^{original} (equivalently, for the same noise variance) when the length of the spreading sequence L is larger; and vice versa. However, as L increases, so does the number of cells required to store each hidden information symbol for hidden information, thereby reducing the effective capacity of the memory.

In order to both increase the accuracy of the recovered hidden information and save storage space, we may group several information symbols together. Grouping means that we can write several information symbols in one group of cells using orthogonal spreading sequences and decode them separately. For example, Fig. 4.16 shows that choosing the spreading sequence length to be $L = 32$ and using two overlapping orthogonal sequences has a better performance than the case with $L = 16$ and a single sequence, despite both schemes use the same storage space.

Additionally, the spreading approach supports multiple access: different hidden information sequences can be written at different times using orthogonal spreading

sequences and without erasing the cells between writes. This can be achieved by shifting $\tilde{v}$ up to be non-negative, so that each write only needs to introduce a small voltage increment on the cells, and shifting the read voltages down before despreading.

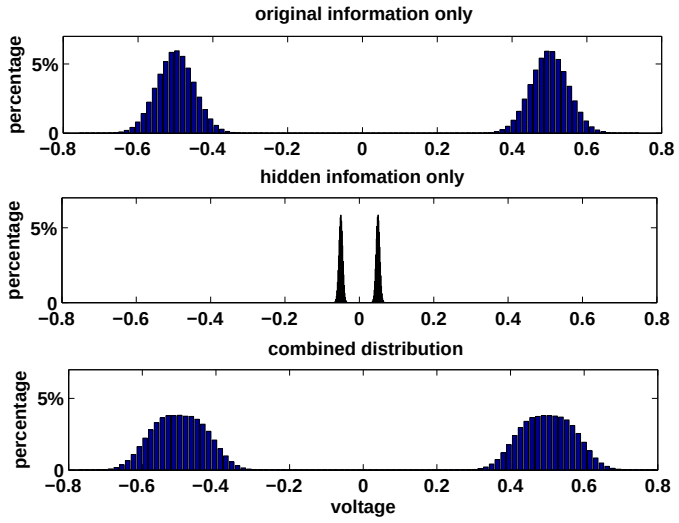


Figure 4.15.: Voltage distribution for a SLC cell with $\sigma = 0.1$, spreading factor $\varepsilon = 0.1$ and length of spreading sequence $L = 16$.

4.4.6 Simulation Results

This section compares the proposed data representation scheme with the traditional one through simulations. It evaluates both of them in terms of BER and damage caused to the memory. We simulate 10 memory blocks with 128 pages per block and 8096 cells in each page. Each cell is assumed to suffer ICI from 3 neighbors in the next wordline, with ICI coefficients $(\gamma_y, \gamma_{xy}) = (0.08, 0.006)$, where γ_y is the ICI coeffcient for the direct neighbor (the one in the same bitline) and γ_{xy} is the ICI coefficient for the two diagonal ones [13, 77]. The write noise is assumed to be Gaussian with zero mean and $\sigma = 0.1$, so $n^w \sim \mathcal{N}(0, 0.1^2)$.

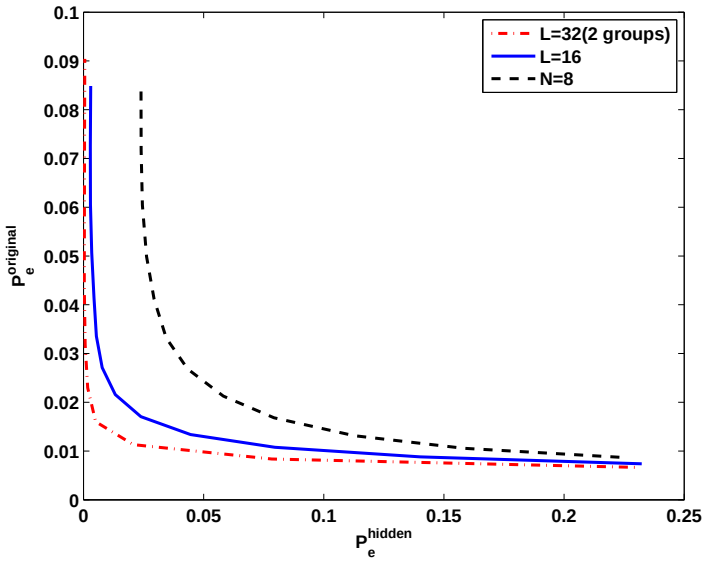


Figure 4.16.: SLC cell with $\sigma = 0.2$. P_e^{original} represents the probability of error of the decoded original information and P_e^{hidden} represents the probability of error of the decoded hidden information.

First, we study how BER increases with ICI when $M = N$, so that the storage efficiency is the same as that in the regular scheme. Assume $M = N = 4$, $k = 1.1$, and the voltages v^0 are cropped to be in the range $[-1.5, 1.5]$ for a MLC memory and $[-3.5, 3.5]$ for a TLC memory. The first two curves in Fig. 4.17 and Fig. 4.18 (no redundancy) show the results for MLC and TLC, respectively. When ICI is small the regular scheme performs better in both MLC and TLC cells, but when ICI increases, the spreading scheme provides lower BER.

We also study the case when there is redundancy. We assume $N = 4$ and $M = 3$, that is, spreading 3 symbols over 4 cells, so that the code rate is 75% in the spreading scheme. In the regular scheme, we use $(15, 11)$ Hamming code to encode the input information so that the code rate is almost the same as that in the spreading scheme. The last two curves in Fig. 4.17 and Fig. 4.18 show that the spreading scheme provides lower BER as ICI increases.

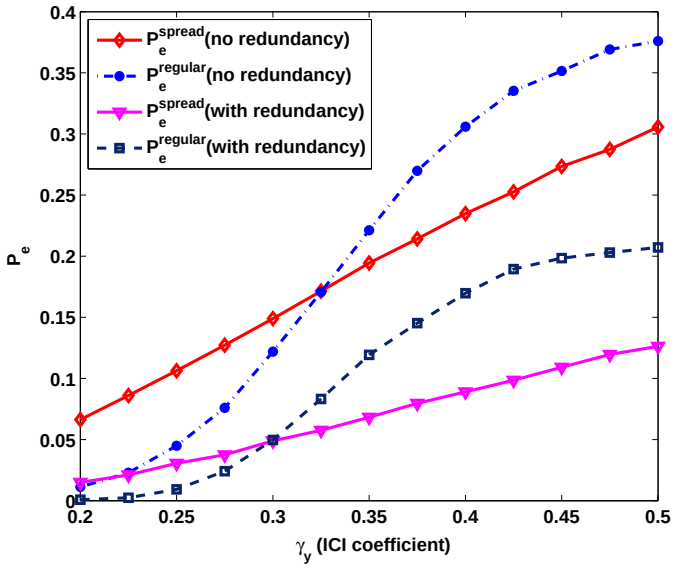


Figure 4.17.: Evolution of the probability of error as ICI increases for an MLC memory (i.e., $\mathbf{b} \in \{-1.5, -0.5, 0.5, 1.5\}^4$), when k=1.1, $\sigma = 0.1$, $(\gamma_y, \gamma_{xy}) = (0.08, 0.006)$, and broken rate $p = 0.001$.

Then, we study the case when the impulse noise dominates the BER in SLC. In order to focus on the impulse noise, both the Gaussian noise and ICI are assumed to be small. The results without parity are given by the first two curves in Fig.4.19 (no LDPC), showing that the BER with the traditional scheme is much larger than with the spreading scheme. Additionally, we analyzed the performance when the spreading modulation was combined with an LDPC encoding of the information. Specifically, we used the (64800,58320) LDPC code which is embedded in matlab R2015b. When the write noise and ICI noise is negligible, the problem of writing and reading information from a flash memory with the traditional modulation is equivalent to transmission over a binary erasure channel (BEC). The spreading scheme, on the other hand, spreads out the noise caused by broken cells, effectively transforming the BEC channel into a binary input AWGN channel, which is better for soft decoding. Fig.4.19 shows that the spreading scheme begins to fail at a larger p and has lower BER among the output bits.

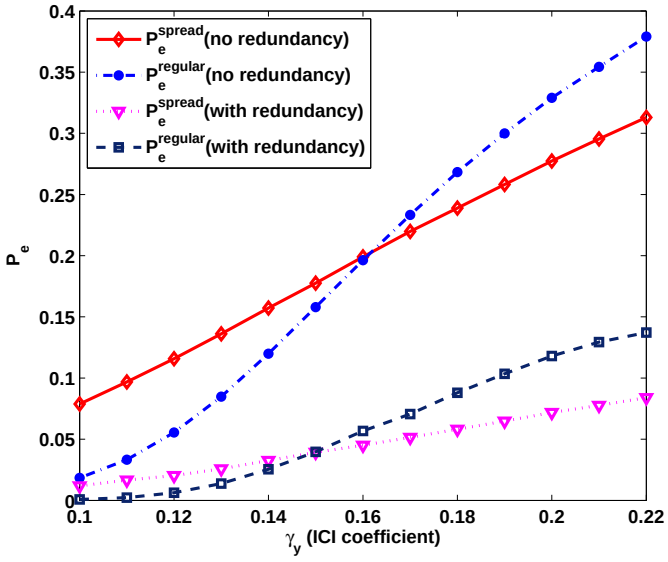


Figure 4.18.: Evolution of the probability of error as ICI increases for an TLC memory (i.e., $\mathbf{b} \in \{\pm 3.5, \pm 2.5, \pm 1.5, \pm 0.5\}^4$), when k=1.1, $\sigma = 0.1$, $\gamma_y : \gamma_{xy} = 0.08 : 0.006$, and broken rate $p = 0.001$.

Finally, we designed an experiment to evaluate how the voltage level to which a cell is programmed influences the damage that it suffers. Our preliminary results showed that when memories are programmed with highly structured data (e.g. 50% of the cells in a wordline written to the same level), they behave abnormally. Hence we tried to use random data in our experiment, while still imposing enough structure to observe different amount of damage in different cells. Four blocks in a 19nm MLC flash were repeatedly erased and programmed with random data, generated according to a different distribution for each cell. For example, some cells were programmed to the highest level 90% of the time, while others only reached that level on 10% of the PE cycles. After the wearing phase, each cell was programmed 100 more times with uniform random data and a dwell time of 1 hour at a temperature of 60C between writes. The information was read back before each new write, so as to obtain an average BER (proxy for damage) at the end of the 100 writes.

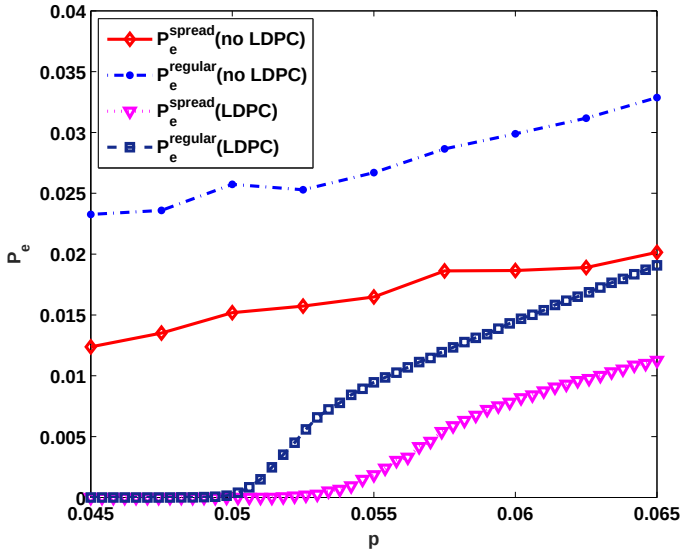


Figure 4.19.: Evolution of BER as cell broken rate p increases for an SLC memory (i.e., $\mathbf{b} \in \{-0.5, 0.5\}^4$), when M=N=4, k=1.1, $\sigma = 0.1$, ICI coefficient $\gamma = 0.1$, the impulse noise dominates the BER.

Once the cells had been worn (to a different number of PE cycles for each block) and the BER data had been collected, we performed a least squares fit to the model

$$BER_i = \alpha_1 P_i^{(1)} + \alpha_2 P_i^{(2)} + \alpha_3 P_i^{(3)} + \alpha_4 P_i^{(4)}, \quad i = 1, \ldots, 10^8,$$

where $P_i^{(j)}$ denotes the probability of programming the i-th cell to level j on each cycle of the wearing phase. The coefficients obtained for each of the four blocks, which should be proportional to the damage caused by programming each level, are shown in Fig. 4.20. Assuming that the voltage levels are equally spaced[5], a clear superlinear behavior can be observed. A quadratic model was adopted for simplicity, yielding the expression in Eq. (4.10) for the damage with each scheme (without loss of generality, we assumed $a = 1$).

[5]Unfortunately, we were not able to verify this fact from our memory manufacturer.

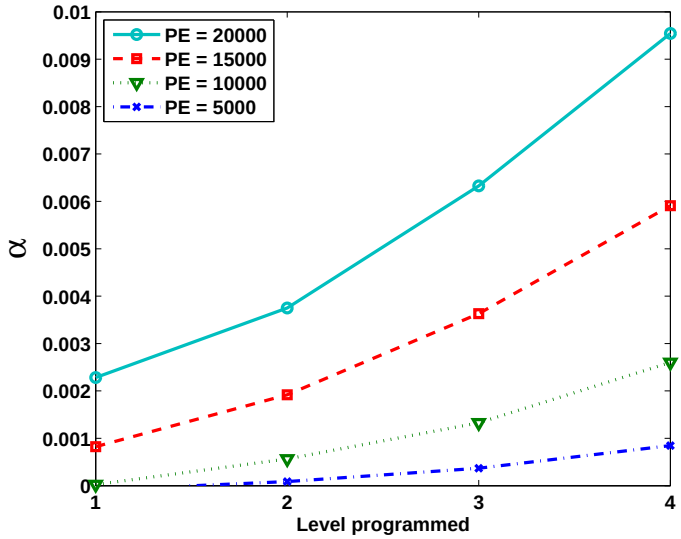


Figure 4.20.: Coefficients modeling the damage suffered by a 19nm MLC cell when programmed to each voltage level, for different numbers of PE cycles.

4.5 Summary

This chapter proposes two signal processing approaches to improve the reliability of NAND flash memories. The first one is called multi-page read method. This method reads multiple wordlines together and returns a bitwise OR (or AND, depending on notation) of their stored information. We show that this read method can improve the reliability by equalizing ICI noise, reduce the damage caused by erase operations, and accelerate the decoding of certain WOM codes. Furthermore, the proposed read method provides a very fast way of operating on the data without actually having to read it, so it is very likely that there exist other applications that can be studied in future research.

We also propose a novel data representation scheme where Walsh codes are used to store the information in a NAND flash memory, so that M symbols are spread out over N cells. We only discuss the case where $M = N$ so that the storage efficiency is the same as that in the regular scheme in this chapter. However, we could have better

performance in the proposed scheme at the cost of more storage space when $N > M$. By increasing the number of possible voltage levels in each cell, disregarding the fact that these levels could overlap, the proposed scheme can provide significant gains in terms of robustness towards inter-cell interference and impulse noise. Additionally, higher levels are programmed less frequently, reducing the damage suffered by the cells and thereby extending the endurance of the memory. We also show that this spreading technique can be used to overlap a hidden layer of information on top of the one stored in plain view. This chapter provides analytical expressions for the SNR and BER of this spreading scheme under Gaussian noise, ICI, and impulse noise. Its performance is then studied through simulations. In next chapter, we will extend these signal processing approaches to a new emerging non-volatile memory, Resistive RAM.

5. SIGNAL PROCESSING FOR CROSSPOINT RESISTIVE MEMORIES

5.1 Introduction

Resistive RAM (ReRAM) has become a research hit in memory industry because of its high density, fast access time and low power consumption. It uses memory resistor ("memristors" in short) to store information. A memristor is a nonlinear resistor whose value can be adjusted by pushing current across its terminals. An SLC ReRAM cell has two states: high resistance state (HRS) and low resistance state (LRS), while MLC cells may have other intermediate levels.

Memristors are implemented in the form of a metal oxide layer sandwiched between two metal electrodes. When a voltage is applied to a ReRAM cell, conductive filaments (CF) are either formed or ruptured depending on the voltage polarity. The cell's resistance depends on the strength of the CFs and thus can be controlled by the programming current. This feature makes multiple-level ReRAM cells possible.

There are two types of architectures for ReRAM memories: MOS-accessed and crosspoint. In MOS-accessed architectures, each memristor is paired with a transistor that isolates the cell from the rest of the array when it is not in use. This 1T1R (one transistor one resistor) architecture provides superior isolation between neighboring cells, power efficiency, and access time, but the transistors dominate the cell size, increasing the area and consequently the cost of the memory. Crosspoint architectures employ diodes (1D1R) or no selector device at all (0T1R) instead of MOS transistors to control cell access [20,78]. Crosspoint architectures for ReRAM offer higher density, power efficiency, and endurance than most other emerging memory technologies, but they suffer sneak currents that cause significant write and read noise.

Typically, writing or reading the information stored in a cell is done by biasing the corresponding wordline with a given voltage, grounding the corresponding bitline, and measuring the flow of current coming out of the bitlines, as shown in Fig. 5.1. Other wordlines and bitlines are partially biased with a smaller voltage, so as to reduce the current through the unselected cells. Ideally, all the current would be flowing through the selected cell, but in practice there are some additional currents flowing through the some unselected cells, especially the cells in the same wordline or bitline as biased (half-selected cells). These additional currents are called sneak currents [20], as illustrated in Fig. 5.1.

The magnitude of the sneak currents increases dramatically with the size of the memristor array. This is specially critical during programming: cells located far from the driver can experience significantly different voltages depending on the state of the other cells in the same bitline and wordline, low when they are all LRS and high when they are HRS. If the voltage drop across the cells is too weak they do not get programmed, but increasing the driver's voltage can cause undesired programming of the cells closer to it. This voltage drop problem can be mitigated using dual-port write as proposed in [79], but at the cost of doubling the area and power required for the drivers.

Sneak currents are also a problem during read operations. The state of a cell is read out by measuring the current leaving the selected bitline [78]. Large currents correspond to cells in the LRS and small currents to cells in the HRS.However, sneak currents introduce noise in the measured currents and can cause errors in the estimation of cell resistances.

The sneak current problem is even more severe in MLC memories, which use intermediate resistance levels between LRS and HRS to store multiple bits in each memristor. MLC memories have higher density but also lower noise margins than SLC memories [80]. The scaling of the technology is causing the HRS resistance to increase while the LRS resistance remains nearly constant, effectively increasing

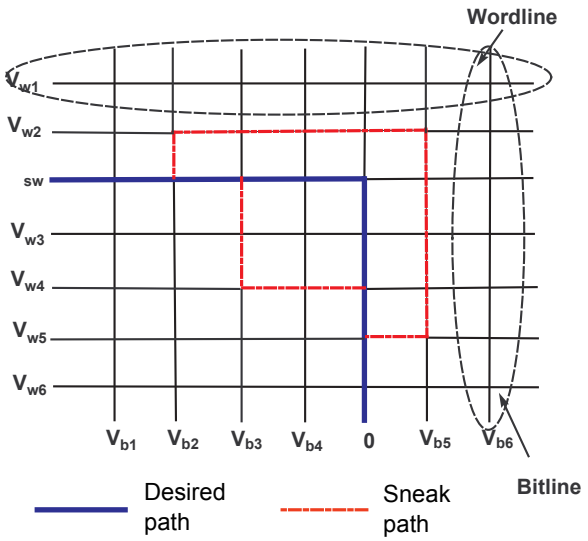


Figure 5.1.: Illustration of sneak currents.

the noise margins and making room for additional levels[1]. Consequently, MLC is becoming the norm in the industry.

Several approaches have been proposed to deal with sneak currents in crosspoint ReRAM arrays. At the device level, it is desirable to have the current through a cell decrease superlinearly with the voltage across its terminals, so as to reduce the sneak currents through half-selected cells. This non-linearity is achieved with special materials or by including a selector in each cell [20]. There also exist methods which perform additional reads to estimate the sneak current (often called background noise) and then subtract it from the current obtained at read time [78]. A more precise alternative is multistage reading, which performs three or more reads of the target cell [80]. These methods can cancel the sneak current, but they sacrifice speed and power efficiency.

This chapter proposes and analyzes the potential of multiple signal processing methods to fight the sneak currents problem. The rest of this chapter is organized as follows: Section 5.2 explains our model of sneak currents as a form of inter-cell interference and states the assumptions that will be made throughout the chapter.

[1]HRS can increase up to $16M\Omega$ when the cell size shinks to $10nm$ [81].

Section 5.3 proposes the different techniques. Section 5.4 presents simulation results to validate the methods. Finally, Section 5.5 summarizes the chapter. The results of this chapter are published in [21].

5.2 System Model

Writes and reads of a MLC ReRAM memory can be done on a cell by cell basis as shown in Fig. 5.1, multiple cells at a time, or even a entire wordline at a time by grounding and sensing the current in all the bitlines. Experiments show that the latter is more power efficient than single cell operations [82], but it has significant disadvantages in terms of area and reliability. This chapter assumes that multiple cells are written and read at a time, but not necessarily the whole wordline.

A critical feature of crosspoint ReRAM architectures is the non-linearity of the cells. That is, the current through a memristor decreases superlinearly as the voltage across its terminals is reduced. This relationship is captured in a non-linearity coefficient

$$k_r(p, V) = p \times \frac{R(V/p)}{R(V)}, \tag{5.1}$$

where $R(V/p)$ and $R(V)$ are the resistances of the cell biased at V/p and at V, respectively. If $k_r(p, V)$ is large, the resistance increases rapidly as the voltage drops. For example, when $k_r(2, V) = 2000$ the resistance of a half-biased cell is 1000 times larger than that of a fully biased cell. Memristors have an inherent amount of non-linearity, but it is not sufficient for typical array sizes [82]. The rest of the chapter will assume that each memristor is paired with a dedicated selector to increase the non-linearity coefficient to $k_r(2, V) = 2000$ [20].

The resistance of the highest and lowest levels are fixedby the device but there exist multiple choices of intermediate resistance levels. An $ISO - \Delta R$ allocation spaces the resistances linearly, resulting in low power consumption but long sensing latency. An $ISO - \Delta I$ allocation spaces the currents (inverse resistances) linearly, increasing noise and power consumption to facilitate read operations. The $ISO - \Delta \log(R)$

allocation provides a trade-off between the previous two by spacing the resistances geometrically [83].

There exist multiple read schemes, with different biasing for wordlines and bitlines. The biasing voltages for selected and non-selected wordlines will be denoted by V_{sw} and V_{w}, respectively. Selected bitlines will be assumed to be grounded and non-selected bitlines will be biased to a voltage V_{b}. Two of the most popular biasings are $V_{\mathrm{w}} = V_{\mathrm{b}} = 0$ (ground-ground) and $V_{\mathrm{w}} = V_{\mathrm{b}} = V_{\mathrm{sw}}/2$ (half-biasing) [84]. The former reduces the sneak currents but consumes a lot of power and causes significant voltage drop in long wordlines [80]. Half biasing alleviates the voltage drop problem and lowers power consumption but suffers stronger sneak currents. The rest of the chapter assumes a half-biasing scheme for reading.

Reads are subject to two main sources of noise: sneak currents and voltage drop. The estimated resistance $\widehat{R_{ij}}$ at crossing (i, j) can be represented as:

$$\widehat{R_{ij}} = R_{ij} + Z_{drop} - Z_{sneak}, \tag{5.2}$$

where R_{ij} is the exact resistance, Z_{drop} is the error caused by the voltage drop along the wordline, and Z_{sneak} is the error caused by sneak currents flowing into the bitline being read. The voltage drop noise is most prominent in bitlines far from the driver. It reduces the measured current, hence increasing estimated resistance. Sneak currents, on the other hand, are similar for all wordlines. They increase the measured current, hence reducing the estimated resistance. The next subsections will develop simplified models for both sources of noise.

5.2.1 Sneak Currents

With half-biasing, non-selected wordlines keep a nearly constant voltage of $V_{sw}/2$ throughout. Consequently, the sneak current flowing into a selected bitline depends mostly on the resistances on that bitline. Ignoring the voltage drop results in the circuit model shown in Fig. 5.2. Let R_{ij} denote the resistance of the memristor

located on the i^{th} wordline and j^{th} bitline of a $n \times n$ array, then the current measured when reading cell (i, j) is

$$I_{ij}^{\text{total}} = I_{ij} + I_{ij}^{\text{sneak}}, \tag{5.3}$$

where

$$I_{ij} = \frac{V_{\text{sw}}}{R_{ij}}, \qquad I_{ij}^{\text{sneak}} = \sum_{1 \le w \le n, w \ne i} \frac{V_{\text{sw}}}{k_r \cdot R_{wj}} \tag{5.4}$$

represent the desired and sneak currents, respectively. If the resistance is estimated as the read voltage divided by the measured current, *i.e.* $\widehat{R_{ij}} = \frac{V_{sw}}{I_{ij}^{\text{total}}}$, then

$$\widehat{R_{ij}^s} = R_{ij} \quad \frac{1}{1 + \sum_{w \ne i} \frac{R_{ij}}{k_r R_{wj}}} \tag{5.5}$$

The sneak currents shown in Eq. (5.4) can be understood as a form of inter-cell-interference (ICI) between the cells in the same bitline. ICI has been extensively studied for other types of memories, such as NAND Flash [13, 17, 49]. Section 5.3 shows how some of the existing methods for dealing with ICI in other technologies can be used in ReRAM memories.

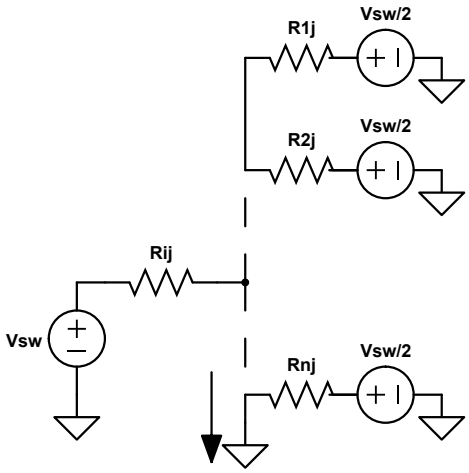


Figure 5.2.: Circuit model for sneak currents in ReRAM.

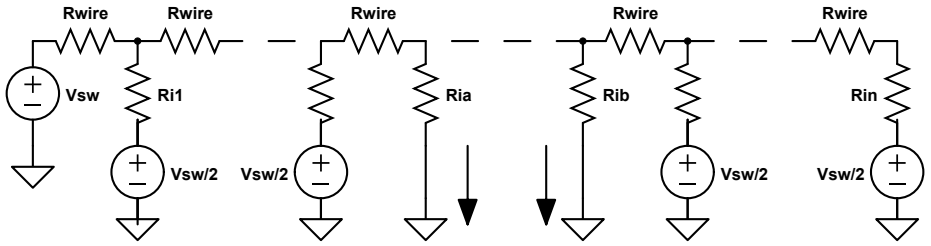


Figure 5.3.: Circuit model for voltage drop in ReRAM.

5.2.2 Voltage Drop

The second source of noise is the voltage drop along the selected wordline. Ignoring the sneak currents through other wordlines results in the circuit model shown in Fig. 5.3, where R_{wire} represents the resistance of the connection between adjacent memristors on the same wordline. Denoting by $I_{i1}, I_{i2}, \ldots, I_{in}$ the currents leaving the selected wordline i through each bitline, the voltage at the j-th cell is given by:

$$V_{ij} = V_{\text{sw}} - R_{\text{wire}} \cdot [1\ 2\ 3 \ldots\ j\ j \ldots j] \cdot \begin{bmatrix} I_{i1} \\ I_{i2} \\ \vdots \\ I_{in} \end{bmatrix}. \tag{5.6}$$

If all the currents are known, it is possible to cancel the voltage drop through equalization: applying Ohm's law ($R_{ij} = \frac{V_{ij}}{I_{ij}}$) to Eq. (5.6) yields:

$$\mathbf{R} = \text{diag}(\mathbf{1}/\mathbf{I}) \cdot (\mathbf{V}_{\text{sw}} - R_{\text{wire}} \cdot \mathbf{A} \cdot \mathbf{I}),$$

where $\mathbf{R}$ and $\mathbf{I}$ are vectors of resistances and currents for the selected wordline and $A_{ij} = \min(i, j)$. However, in large arrays it is only possible to read a few bitlines at a time. Otherwise, the power consumed becomes too large and the excessive voltage drops introduce non-linear effects.

If the resistances are estimated as $\widehat{R_{ij}} = \frac{V_{sw}}{I_{ij}}$ then

$$\widehat{R_{ij}^v} \approx R_{ij} \left(1 + R_{\text{wire}} \left[1 \; 2 \ldots \; j \ldots j \right] \begin{bmatrix} \alpha_1 / R_{i1} \\ \vdots \\ \alpha_n / R_{in} \end{bmatrix} \right), \tag{5.7}$$

where $\alpha = 1/k_r$ for half-biased bitlines and $\alpha = 1$ otherwise. Once again, the voltage drop can be understood as a form of Inter-cell-interference (ICI), this time between the cells in the same wordline.

If both sneak currents and voltage drop are taken into account and resistance levels are exponentially spaced, the estimated resistance $\widehat{R_{ij}}$ can be approximated as:

$$\begin{aligned} \log(\widehat{R_{ij}}) = \log(R_{ij}) - \log \quad & 1 + \sum_{w \neq i} \frac{R_{ij}}{k_r R_{wj}} \\ & + \log \left(1 + R_{wire} [1 \; 2 \ldots j \ldots j] \begin{bmatrix} \alpha_1 / R_{i1} \\ \vdots \\ \alpha_n / R_{in} \end{bmatrix} \right). \end{aligned} \tag{5.8}$$

5.3 Compensation for Sneak Currents

In ReRAM memories, information is modulated into cell resistances: SLC cells take two resistance levels, storing one bit of information, and MLC cells take four resistance levels, storing two bits of information. However, there is no inherent limitation in the number of levels that a cell can take and it is relatively easy to program other resistance levels with little write noise [83]. It is therefore possible to increase the number of levels that the cells take and use the redundancy to improve the reliability of the channel. This is commonly known as coded modulation.

Coded modulation can be used to reduce the number of errors in the memory [85, 86] or to reduce ICI and other sources of noise [17, 51]. This section proposes two techniques for reducing the voltage drop and sneak current noise: spreading modulation and distribution shaping.

5.3.1 Spreading Modulation

Spreading modulation was proposed in [17] to reduce ICI in NAND flash. This section shows how the same method can be used to address the sneak currents in ReRAM, with some additional advantages. The main idea is to use orthogonal spreading sequences to store multiple information symbols in the same cells without interfering, similar to the Code Division Multiple Access (CDMA) method used in wireless communications.

In its most general form, the spreading modulation uses a $N \times M$ Walsh submatrix (i.e., with ± 1 entries and mutually orthogonal columns) denoted $\mathbf{C}$ to map a vector of M data symbols $\mathbf{b}$ (assumed zero-mean) to a vector of N cell currents $\mathbf{I}$, where $N \geq M$. The corresponding cells are then programmed to appropriate resistances so that, when biased at V_{sw}, their currents are $\mathbf{I}$. For example, when $N = 4$ and $M = 3$

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{bmatrix} = \alpha \cdot \begin{bmatrix} 1 & 1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \\ -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} + \beta \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}, \tag{5.9}$$

where the scaling and shifting parameters α and β can be used to conform with the feasible range of currents. In general, the modulation can be expressed as

$$\mathbf{I}^{\mathbf{written}} = \alpha \mathbf{C}\mathbf{b} + \beta \mathbf{1},$$

and the demodulation (or despreading) as

$$\hat{\mathbf{b}} = \frac{1}{\alpha N} \mathbf{C}^T \left(\mathbf{I}^{\text{read}} - \beta \mathbf{1} \right).$$

The benefits of this scheme are two-fold. First, it programs intermediate resistance levels more often than extreme ones, thereby reducing the variance in the currents through the cells. Sneak currents are then more predictable and it becomes simpler to compensate for them, as shown in Fig. 5.4. Furthermore, the voltage drop problem during programming is significantly alleviated by the reduction in number of LRS

cells. The second benefit is that each information symbol is spread over many cells, so the information becomes less vulnerable to cell failures [17]. ReRAM cells do not suffer gradual drifts in their resistances, unlike Flash or PCM, but they are prone to large magnitude errors caused by sudden transitions to their lowest or highest resistance states [83].

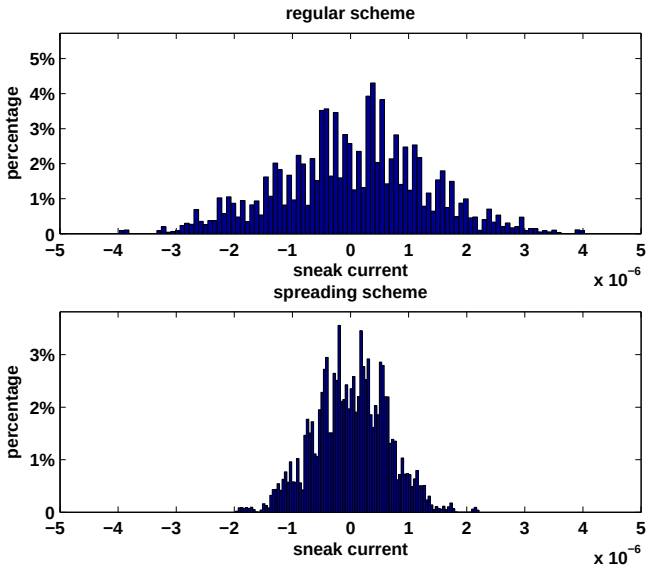


Figure 5.4.: Distribution of sneak currents centered at 0.

The spreading can also be done vertically, across cells on the same bitline. In this case, sneak currents are automatically canceled during despreading for symmetric spreading sequences, since sneak currents are nearly identical for all the cells in a bitline. However, it requires reading multiple wordlines to recover the information. Reading all N wordlines allows for recovering of all M information symbols, but it is also possible to recover a single symbol using two reads. The first read activates the wordlines corresponding to positive entries in the spreading sequence, and measures the sum of their currents flowing down the bitline. The second read does the same with the negative entries. Then, it is just a matter of subtracting both currents and estimating the information symbol.

5.3.2 Distribution Shaping

Another alternative for addressing the sneak current and voltage drop problems without expanding the modulation is to shape the distribution of the programmed levels to approach the channel capacity. The magnitude of the sneak currents and voltage drops depend on the data written and the bitlines being read, as shown by Eq. (5.8). Figure 5.5 illustrates this dependence for a 256×256 SLC ReRAM with $LRS = 10^3\Omega$, $HRS = 10^6\Omega$. The mean shift and variance for the HRS remain almost the same for all bitlines but for the LRS they both increase with the distance to the drivers.

Large resistances cause smaller voltage drops and sneak currents on other positions but they suffer more noise themselves. The sneak current flowing into a wordline is independent of the position within the array, but bitlines far from the drivers suffer and create stronger voltage drops. Intuitively, the distribution should favor high resistances at bitlines far from the driver so as to reduce the noise in all other positions, and use a balance distribution for the first bitlines since they barely suffer any voltage drop.

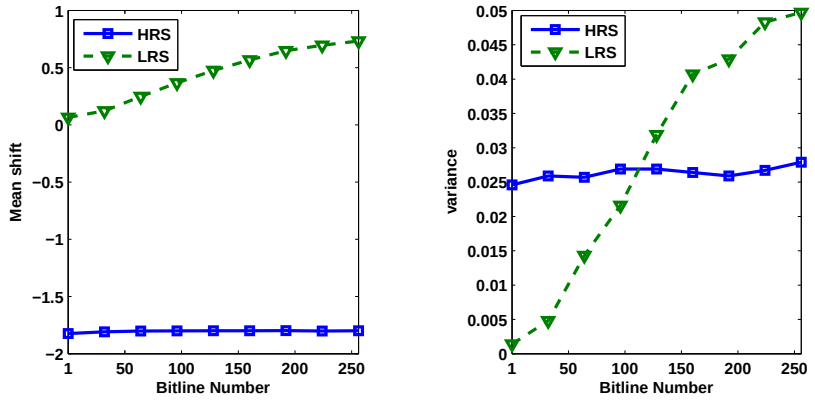


Figure 5.5.: Noise shift and variance for 256×256 SLC ReRAM with $LRS = 10^3\Omega$, $HRS = 10^6\Omega$, $R_{\text{wire}} = 1\Omega$, and $k_r = 2000$.

The optimal distribution depends strongly on the array's characteristics and needs to be found numerically for each case. Once the desired distribution has been found, it is necessary to design an encoder and decoder tailored to that distribution. One possible way of achieving this is to use a lossless data compressor as decoder and the corresponding de-compressor as encoder. Iterative source/channel decoding can then be used to correct errors [87].

5.4 Simulations

This section provides simulation results to validate the model proposed in Section 5.2 and analyze the techniques proposed in Section 5.3. The simulations are based on a system of linear equations for the cross-point ReRAM obtained from Kirchhoff's Current Law (KCL), which was shown in [82] to be highly accurate. Except where stated otherwise, all simulations are for a 512x512 array with $R_{\text{wire}} = 1\Omega$, $k_r = 2000$, and $V_{\text{sw}} = 1$. Resistance levels are $(10^3,\ 10^4,\ 10^5,\ 10^6)$ and reads are done 16 cells at a time, evenly spaced along the wordline. For example, bitlines $1, 33, \ldots, 481$ are read simultaneously, as are bitlines $2, 34, \ldots, 482$.

First, we compare the resistance estimates predicted by the model in Eq. (5.8) with those obtained from the simulations as $\widehat{R_{ij}} = \frac{V_{sw}}{I_{ij}}$. Figure 5.6 shows the average difference between both estimates, normalized by the exact resistance, for different bitlines and resistance levels. It can be observed that the relative error is quite small, specially for large resistances and bitlines far from the voltage drivers, which are the most critical cases.

Figure 5.7 compares the BER observed with the spreading modulation scheme in Eq. (5.9) with that of a typical MLC modulation concatenated with an error correcting code of similar rate. For small arrays, both of them show negligible BER, but the spreading modulation provides significantly lower error rates for arrays larger than 256×256. The spreading modulation requires multiple reads on different wordlines

(four in this case, since the spreading is done vertically), but it also returns more data so read throughput is not affected.

A significant portion of the gains obtained by the spreading modulation technique stem from the lower fraction of cells in LRS state. Figure 5.8 shows the capacity-maximizing distribution of levels for each bitline, discretized to the closest percentage point. It can be observed that, for the case under consideration, far bitlines should reduce the frequency of LRS and use the other three levels with equal probability. The information-theoretic capacity increases from 1.81 bits per cell when all four levels are equiprobable to 1.96 bits per cell with the distribution in Fig. 5.8. The attenuation of sneak currents and voltage drops more than compensates for the loss in capacity due to the asymmetric input distribution.

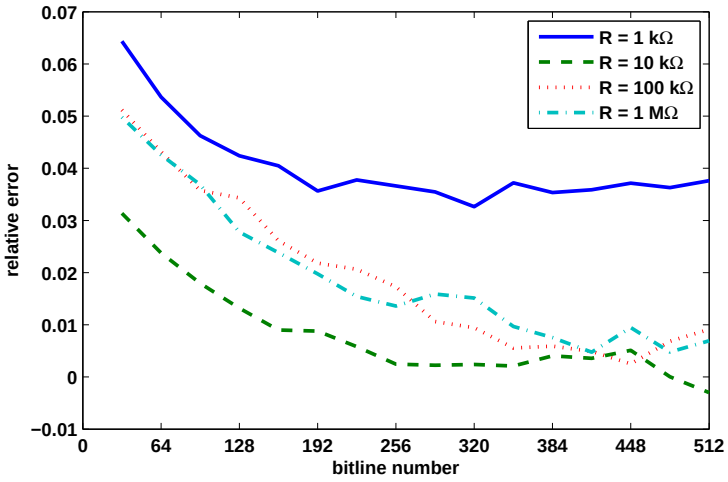


Figure 5.6.: Relative error between the simulated resistance estimates and those predicted by Eq. (5.8).

5.5 Summary

Crosspoint resistive memories are a very promising storage technology providing high density, fast access time, and low power consumption. As the memristors scale, the gap between the LRS and the HRS is becoming wider, so it can be expected

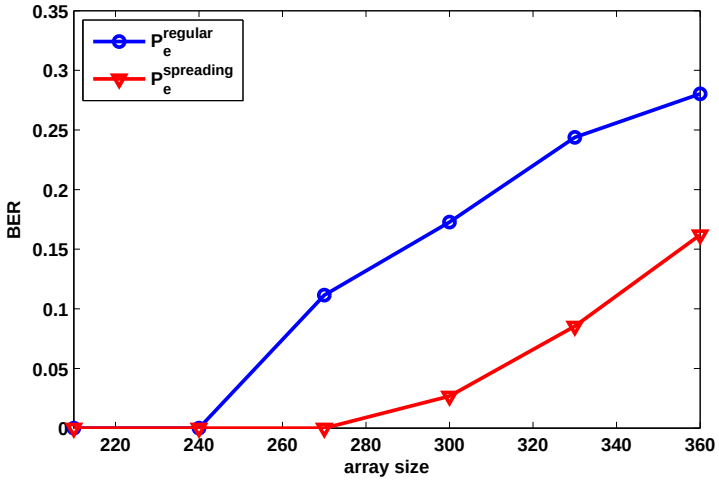


Figure 5.7.: Evolution of the BER as the array size grows. The regular scheme uses a (15,11) Hamming code and the spreading modulation has $N = 4$, $M = 3$, so the rate is approximately 75% for both.

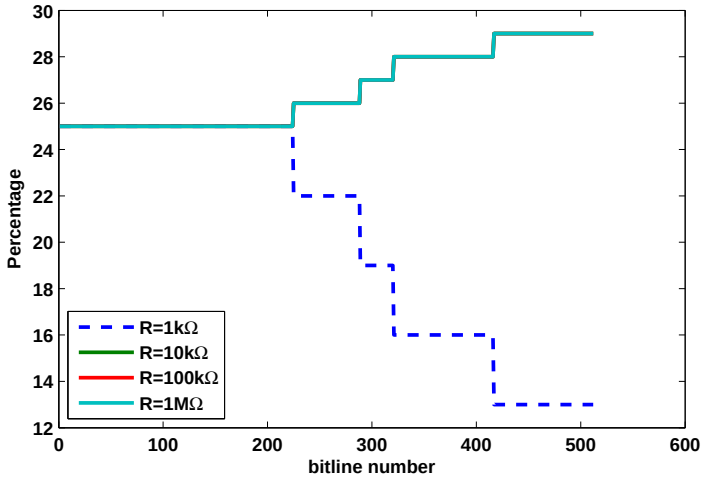


Figure 5.8.: Capacity-maximizing distribution of resistance levels per bitline. LRS cells cause more noise, so they are chosen less often; the other levels are equiprobable so their graphs overlap.

that most ReRAM memories will soon store multiple bits in each cell. Unfortunately, resistive memories suffer from voltage drops and sneak currents which limit the size of the arrays. A lot of effort is being put into increasing the non-linearity of the cells

through better materials and selectors, but there is little research available from the perspective of signal processing.

This chapter proposed a simple analytical model for estimating the read noise and two data representation techniques for reducing it. The first is a coded modulation scheme which spreads each data symbol across multiple cells, reducing the variance in the programmed resistances and thereby making noise more predictable. The second scheme consists of shaping the distribution of programmed levels to reduce noise levels. Both schemes were evaluated through simulations.

6. SUMMARY AND FUTURE WORK

In this thesis, we utilize signal processing approaches to solve problems in two storage systems: caching networks and non-volatile memories. Algorithms are proposed to improve the efficiency of information delivery in networks and reliability in storage hardware. Section 6.1 and Section 6.2 present a summary of our work and some suggested directions for future research in each of these two systems.

6.1 Caching Networks

Caching has been investigated as a useful technique to reduce the network load by prefetching some contents during off-peak hours. It contains two phases: placement and delivery. In the placement phase, the users have access to the database fill their caches. In the delivery phase, each user requests one file and only the server has access to all files. The server delivers messages to the users to fulfil their requests. Our work focuses on designing caching and delivery strategies for content delivery networks bases on Maddah-Ali and Niesen's coded caching scheme [1].

We first propose coded caching algorithms for reducing the peak data rate in multi-user multi-server systems with distributed storage and different levels of redundancy in Chapter 2. The content delivery network is assumed to be flexible, in the sense that there is a path from every server to every user. Files are encoded using erasure codes and distributed among servers to fight with disk failures. Some systems use stripping across multiple servers, while others store whole file as a single unit for simplifying book-keeping. We propose algorithms for both cases in this thesis. It shows that, by striping each file across multiple servers, the peak rate can be reduced proportionally to the number of servers. Then it addresses the case where each file

is stored as a single unit in one server and proposes different caching and delivery schemes depending on the size of the cache memories.

One possible direction for future work is extending our scheme to distributed system with more advanced erasure codes. In Chapter 2, we developed RAID-4 codes and RAID-6 codes to combine coded caching and distributed storage. However, more advanced erasure codes are used in real network systems. Therefore, it will be interesting to study how this process can be generalized to larger systems and more advanced erasure codes, such as fractional repetition codes [35, 36] or other RAID-6 [39] structures. It is also interesting to study the case where files have different popularity. All the former work [3,88] has focused on the case where all data nodes (disks, servers,...) have identical probability of failure and cost of recovery. This makes sense for error correction applications but when the data consists of files with different size or popularity, it would be useful to make some files easier to recover than others. Yet another interesting problem would be to find erasure codes for systems where different nodes have different failure probabilities.

Our second contribution is study of the traffic load-I/O trade-off for coded caching in Chapter 3. In this work, we study the caching and delivery scheme for the system identical to [1]: users connect to a single server through a broadcast link. The I/O performance for the coded caching scheme proposed by Maddah-Ali and Neisen [1] is suboptimal when there are redundant requests. When the server constructs messages, the same segment could be read multiple times if it used to construct different messages, which dramatically increases I/O reads. This thesis proposes caching and delivery algorithms which combine coded and uncoded transmission to leverage the trade-off between traffic load and disk I/Os. Our algorithms can improve both the average and worst case performance in terms of the user requests. In the future, it would be interesting to extend our work to study the traffic load-I/O trade-off with coded prefetching.

6.2 Non-volatile Memories

Our research on non-volatile memories mainly focuses on improving hardware reliability and endurance. In Chapter 4, we study NAND flash memories. A NAND flash memory is fundamentally an array of floating gate transistors, known as flash cells, whose threshold voltages can be programmed to represent different information symbols. In order to reduce the cost, manufacturers are trying to shrink the cell and pack more bits in one cell, which brings reliability challenges, specially ICI noises. ICI is a phenomenon by which programming a cell increases the voltage of its neighbors.

Two methods are proposed in this thesis to improve the reliability of NAND flash: multi-page read and spreading modulation. The multi-page read method reads multiple wordlines together and returns a combination of their stored information. It is shown that this read method can improve the reliability of the stored information by equalizing ICI noise, reduce the damage caused by erase operations, and accelerate the decoding of certain constrained codes. The spreading modulation spreads stored information across multiple cells using Walsh codes. Higher levels are used less frequently than in the regular scheme, providing significant gains in terms of robustness towards ICI and reducing the damage suffered by the cells. We also show that this spreading technique can be used to overlap a hidden layer of information on top of the one stored in plain view.

In Chapter 5, we focus on another type of promising non-volatile memories, Resistive RAM. Re-RAM uses memory resistors to store information, whose value can be adjusted by pushing current across its terminals. In order to increase the density, crosspoint architectures are used. But it brings other problems, mainly related to sneak currents flowing through deactivated cells. We propose a simple analytical model for estimating the read noise and two data representation techniques for reducing it. The first one is spread modulation which is extended from that of NAND. It reduces the variance in the programmed resistances and thereby making noise more

predictable. The second scheme consists of shaping the distribution of programmed levels to reduce noise levels.

One interesting future work could be extending these signal processing approaches to some new emerging memory architectures like 3D flash memories [89], which offer much higher capacity and have become widespread among flash manufacturers. Instead of shrinking cells within a 2D plane, 3D flash memories stack up cells in the vertical direction [90, 91] and read noises become even more pronounced in this 3D array architecture [92].

APPENDIX

A. PROOF FOR LEMMA 2.4.3

In this appendix, we will elaborate on the pairing scheme in Lemma 2.4.3 from Chapter 2, specially for the case with even K and symmetric requests.

Definition A.0.1 *Let χ_A denote a set of messages (or, equivalently, subsets of $t+1$ users) to be sent by server A and χ_B denote a set of messages to be sent by server B. If there is an injective function providing each element in χ_A with an effective pair in χ_B, we say that there is a* ***saturating matching*** *for χ_A.*

In order to reduce the peak rate we want to separate all the messages to be transmitted (equivalently, subsets of $t+1$ users) into two groups χ_A and χ_B such that there are as many effective pairs as possible, as we shall see.

To better illustrate the allocation scheme, the problem of finding effective pairs is mapped to a graph problem. Let G be a finite bipartite graph with bipartite sets χ_A and χ_B, where each message (or user subset) is represented as a vertex in the graph and edges connect effective pairs from χ_A and χ_B. The idea of our design is to allocate as many messages as possible to χ_A, while guaranteeing the existence of a saturating matching for χ_A based on Hall's marriage Theorem [93].

Theorem A.0.1 (Hall's Marriage Theorem [93]) *Let G be a finite bipartite graph with bipartite sets χ_A and χ_B. For a set $\mathbf{u}$ of vertices in χ_A, let $N_G(\mathbf{u})$ denote its neighbourhood in G, i.e. the set of all vertices in χ_B adjacent to some element of $\mathbf{u}$. There is a matching that entirely covers χ_A if and only if*

$$|\mathbf{u}| \leq |N_G(\mathbf{u})|$$

for every subset $\mathbf{u}$ of χ_A.

Corollary A.0.1.1 *If all vertices in χ_A have the same degree d_A and all the vertices in χ_B have the same degree d_B ($d_A \geq d_B$), then there is a saturating matching for χ_A.*

Proof For any $\mathbf{u} \subseteq \chi_A$, all edges connected to $\mathbf{u}$ are also connected to $N_G(\mathbf{u})$, hence $|N_G(\mathbf{u})| \cdot d_B \geq |\mathbf{u}| \cdot d_A$. Since $d_A \geq d_B$, we know that $|\mathbf{u}| \leq |N_G(\mathbf{u})|$. According to Theorem A.0.1, there is a saturating matching for χ_A. ■

In order to compute the peak rate in the worst case, we assume that all K users request different files. Since each subset contains $t+1$ files, there are $\binom{K}{t+1}$ messages to allocate between χ_A and χ_B. We classify these subsets according to the number of requests from server A: sets of *type* w will have w requests from server A and $t+1-w$ from server B. The following proposition states that the messages of the same type are not able to pair with each other.

When t is even and the demands are symmetric, *type* w sets and *type* $t+1-w$ sets form a symmetric bipartite graph, so there exists a saturating matching according to Corollary A.0.1.1. When t is odd, *type* $(t+1)/2$ sets are paired with the union of *type* $(t-1)/2$ sets and *type* $(t+3)/2$ sets. Since the vertices in *type* $(t-1)/2$ sets and *type* $(t+3)/2$ sets are connected to the same number of vertices in *type* $(t+1)/2$ sets, this bipartite graph also fulfills the condition in Corollary A.0.1.1. Other sets are paired as in the case with t even, that is, *type* w sets are paired with *type* $t+1-w$ sets. These pairings are illustrated in Fig.A.1.

When t is even, there is a matching for every candidate file set, thus the peak rate is cut by half compared with the traditional single server scheme. When t is odd, some vertices of types $(t-1)/2$, $(t+1)/2$, or $(t+3)/2$ could fail to be paired. Denote the ratio of unpaired messages when t is odd by Δ. Any two servers can collaborate to fulfill those requests, so the normalized overall peak rate R_T with symmetric demands is given by:

$$R_T(K,t) = \begin{cases} \frac{1}{2} R_C(K,t) & \text{if } t \text{ is even} \\ \left(\frac{1}{2} + \frac{1}{6}\Delta\right) R_C(K,t) & \text{if } t \text{ is odd,} \end{cases}$$

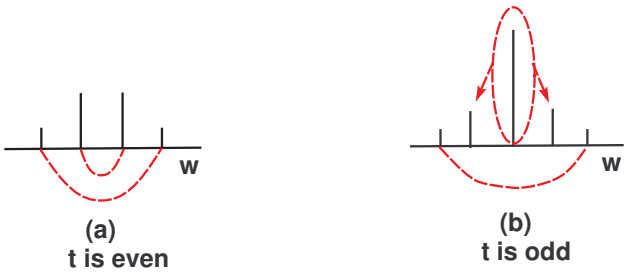


Figure A.1.: Pairing illustration. w is the number of files from server A in a message.

The pairing loss Δ is limited. The worst case occurs when there is a big difference between the number of vertices of type $(t+1)/2$ and the number of vertices of types $(t-1)/2$ or $(t+3)/2$. In both cases, the pairing loss Δ is bounded by $\frac{1}{3}$.

VITA

VITA

Tianqiong Luo received her B.S. from Fudan University, Shanghai, China, in 2013. She is currently pursuing the Ph.D. degree of Electrical and Computer Engineering at Purdue University. Her research interests involve signal processing and coding for caching networks and non-volatile storage systems.

REFERENCES

REFERENCES

[1] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Inf. Theory*, vol. 60, no. 5, pp. 2856–2867, 2014.

[2] R. L. Rivest and A. Shamir, "How to reuse a "write-once" memory," *Information and control*, vol. 55, no. 1, pp. 1–19, 1982.

[3] A. G. Dimakis, P. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, 2010.

[4] J. S. Plank, "The RAID-6 liber8tion code," *International Journal of High Performance Computing Applications*, 2009.

[5] C. Tian and J. Chen, "Caching and delivery via interference elimination," *arXiv preprint arXiv:1604.08600*, 2016.

[6] T. Luo, V. Aggarwal, and B. Peleato, "Coded caching with distributed storage," *arXiv preprint arXiv:1611.06591*, 2016.

[7] D. Beaver, S. Kumar, H. C. Li, J. Sobel, P. Vajgel *et al.*, "Finding a needle in haystack: Facebook's photo storage." in *OSDI*, vol. 10, no. 2010, 2010, pp. 1–8.

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[9] K. Rashmi, P. Nakkiran, J. Wang, N. B. Shah, and K. Ramchandran, "Having your cake and eating it too: Jointly optimal erasure codes for I/O, storage, and network-bandwidth." in *FAST*, 2015, pp. 81–94.

[10] O. Khan, R. C. Burns, J. S. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads." in *FAST*, 2012, p. 20.

[11] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "The exact rate-memory trade-off for caching with uncoded prefetching," in *IEEE Int. Symp. on Information Theory Proceedings (ISIT)*. IEEE, 2017, pp. 1613–1617.

[12] T. Luo and B. Peleato, "The rate-I/O trade-off for coded caching," *submitted to IEEE Commun. Lett.*, 2018.

[13] G. Dong, S. Li, and T. Zhang, "Using data postcompensation and predistortion to tolerate cell-to-cell interference in MLC NAND flash memory," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 57, no. 10, pp. 2718–2728, Oct. 2010.

[14] W. Wang, T. Xie, and D. Zhou, "Understanding the impact of threshold voltage on MLC flash memory performance and reliability," in *Proc. 28th ACM Int. Conf. on Supercomputing (ICS)*. ACM, 2014, pp. 201–210.

[15] S. Moshavi, "Multi-user detection for DS-CDMA communications," *IEEE Commun. Mag.*, vol. 34, no. 10, pp. 124–136, Oct. 1996.

[16] F. Adachi, M. Sawahashi, and H. Suda, "Wideband DS-CDMA for next-generation mobile communications systems," *IEEE Commun. Mag.*, vol. 36, no. 9, pp. 56–69, Sep. 1998.

[17] T. Luo and B. Peleato, "Spread programming for NAND flash," in *IEEE Int. Conf. on Communications (ICC)*. IEEE, 2015, pp. 277–282.

[18] ——, "Spreading modulation for multilevel nonvolatile memories," *IEEE Trans. Commun.*, vol. 64, no. 3, pp. 1110–1119, 2016.

[19] ——, "Multipage read for NAND flash," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 1, pp. 76–80, 2017.

[20] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *IEEE 21st International Symp. on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 476–488.

[21] T. Luo, O. Milenkovic, and B. Peleato, "Compensating for sneak currents in multi-level crosspoint resistive memories," in *49th Asilomar Conference on Signals, Systems and Computers*. IEEE, 2015, pp. 839–843.

[22] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *FAST-2004: 3rd Usenix Conference on File and Storage Technologies*, 2004.

[23] H. Ghasemi and A. Ramamoorthy, "Improved lower bounds for coded caching," in *IEEE Int. Symp. on Information Theory Proceedings (ISIT)*. IEEE, 2015, pp. 1696–1700.

[24] U. Niesen and M. A. Maddah-Ali, "Coded caching with nonuniform demands," in *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*. IEEE, 2014, pp. 221–226.

[25] J. Zhang, X. Lin, C.-C. Wang, and X. Wang, "Coded caching for files with distinct file sizes," in *IEEE Int. Symp. on Information Theory Proceedings (ISIT)*. IEEE, 2015, pp. 1686–1690.

[26] M. Ji, A. M. Tulino, J. Llorca, and G. Caire, "Caching and coded multicasting: Multiple groupcast index coding," in *IEEE Global Conf. on Signal and Information Processing (GlobalSIP)*. IEEE, 2014, pp. 881–885.

[27] M. Ji, A. Tulino, J. Llorca, and G. Caire, "Caching-aided coded multicasting with multiple random requests," in *IEEE Information Theory Workshop (ITW)*. IEEE, 2015, pp. 1–5.

[28] J. Hachem, N. Karamchandani, and S. Diggavi, "Effect of number of users in multi-level coded caching," in *IEEE Int. Symp. on Information Theory Proceedings (ISIT)*. IEEE, 2015, pp. 1701–1705.

[29] S. P. Shariatpanahi, S. A. Motahari, and B. H. Khalaj, "Multi-server coded caching," *arXiv preprint arXiv:1503.00265*, 2015.

[30] R. Blom, "An optimal class of symmetric key generation systems," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1984, pp. 335–338.

[31] C. Suh and K. Ramchandran, "Exact-repair MDS code construction using interference alignment," *IEEE Trans. Inf. Theory*, vol. 57, no. 3, pp. 1425–1442, 2011.

[32] S. El Rouayheb, A. Sprintson, and C. Georghiades, "On the index coding problem and its relation to network coding and matroid theory," *IEEE Trans. Inf. Theory*, vol. 56, no. 7, pp. 3187–3195, 2010.

[33] Z. Bar-Yossef, Y. Birk, T. Jayram, and T. Kol, "Index coding with side information," *IEEE Trans. Inf. Theory*, vol. 57, no. 3, pp. 1479–1494, 2011.

[34] M. A. R. Chaudhry and A. Sprintson, "Efficient algorithms for index coding," in *IEEE INFOCOM Workshops*. IEEE, 2008, pp. 1–4.

[35] S. El Rouayheb and K. Ramchandran, "Fractional repetition codes for repair in distributed storage systems," in *Proc. 48th Annu. Allerton Conf. Commun., Control, and Comput. (Allerton)*. IEEE, 2010, pp. 1510–1517.

[36] Q. Yu, C. W. Sung, and T. H. Chan, "Irregular fractional repetition code optimization for heterogeneous cloud storage," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 1048–1060, 2014.

[37] C. Huang, M. Chen, and J. Li, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," *ACM Trans. Storage (TOS)*, vol. 9, no. 1, p. 3, 2013.

[38] J. R. Santos, R. R. Muntz, and B. Ribeiro-Neto, *Comparing random data allocation and data striping in multimedia servers*. ACM, 2000, vol. 28, no. 1.

[39] Y. Wang, X. Yin, and X. Wang, "MDR codes: A new class of RAID-6 codes with optimal rebuilding and encoding," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 1008–1018, 2014.

[40] U. Niesen and M. A. Maddah-Ali, "Coded caching with nonuniform demands," *IEEE Trans. Inf. Theory*, vol. 63, no. 2, pp. 1146–1158, 2017.

[41] M. Ji, A. M. Tulino, J. Llorca, and G. Caire, "On the average performance of caching and coded multicasting with random demands," in *11th Int. Symp. on Wireless Communications Systems (ISWCS)*. IEEE, 2014, pp. 922–926.

[42] S. A. Saberali, H. E. Saffar, L. Lampe, and I. Blake, "Adaptive delivery in caching networks," *IEEE Commun. Lett.*, vol. 20, no. 7, pp. 1405–1408, 2016.

[43] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *IEEE/ACM Trans. Netw. (TON)*, vol. 23, no. 4, pp. 1029–1040, 2015.

[44] E. J. O'neil, P. E. O'neil, and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering," *ACM SIGMOD Record*, vol. 22, no. 2, pp. 297–306, 1993.

[45] Z. Chen, P. Fan, and K. B. Letaief, "Fundamental limits of caching: Improved bounds for small buffer users," *arXiv preprint arXiv:1407.1935*, 2014.

[46] R. Frickey, "Data integrity on 20nm SSDs," in *Flash Memory Summit*, 2012.

[47] P. Pavan, R. Bez, P. Olivo, and E. Zanoni, "Flash memory cells-an overview," *Proc. IEEE*, vol. 85, no. 8, pp. 1248–1271, Aug. 1997.

[48] B. Shin, C. Seol, J.-S. Chung, and J. J. Kong, "Error control coding and signal processing for flash memories," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2012, pp. 409–412.

[49] M. Qin, E. Yaakobi, and P. H. Siegel, "Constrained codes that mitigate inter-cell interference in read/write cycles for flash memories," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 836–846, May 2014.

[50] J.-D. Lee, S.-H. Hur, and J.-D. Choi, "Effects of floating-gate interference on NAND flash memory cell operation," *IEEE Electron Device Lett.*, vol. 23, no. 5, pp. 264–266, 2002.

[51] Y. Kim, B. Kumar, K. L. Cho, H. Son, J. Kim, J. J. Kong, and J. Lee, "Modulation coding for flash memories," in *International Conf. on Comput., Netw. and Commun. (ICNC)*. IEEE, 2013, pp. 961–967.

[52] Y. Kim and B. V. Kumar, "Coding for memory with stuck-at defects," in *IEEE Int. Conf. on Communications (ICC)*. IEEE, 2013, pp. 4347 – 4352.

[53] H.-W. Tseng, L. Grupp, and S. Swanson, "Understanding the impact of power loss on flash memory," in *Proc. 48th Design Automation Conf. (DAC)*. ACM, 2011, pp. 35–40.

[54] A. Jagmohan, M. Franceschini, L. Lastras-Montano, J. Karidis *et al.*, "Adaptive endurance coding for NAND flash," in *Proc. IEEE GLOBECOM Workshops*. IEEE, Dec. 2010, pp. 1841–1845.

[55] B. Peleato and R. Agarwal, "Maximizing MLC NAND lifetime and reliability in the presence of write noise," in *IEEE Int. Conf. on Communications (ICC)*. IEEE, 2012, pp. 3752–3756.

[56] B. Peleato, R. Agarwal, and J. Cioffi, "Probabilistic graphical model for flash memory programming," in *IEEE Statistical Signal Processing Workshop (SSP)*. IEEE, 2012, pp. 788–791.

[57] R. Katsumata, M. Kito, Y. Fukuzumi, M. Kido, H. Tanaka, Y. Komori, M. Ishiduki, J. Matsunami, T. Fujiwara, Y. Nagata *et al.*, "Pipe-shaped BiCS flash memory with 16 stacked layers and multi-level-cell operation for ultra high density storage devices," in *Proc. IEEE Symp. on VLSI Technol.*, 2009, pp. 136–137.

[58] K.-D. Suh, B.-H. Suh, Y.-H. Lim, J.-K. Kim, Y.-J. Choi, Y.-N. Koh, S.-S. Lee, S.-C. Kwon, B.-S. Choi, J.-S. Yum *et al.*, "A 3.3 v 32 mb NAND flash memory with incremental step pulse programming scheme," *IEEE J. Solid-State Circuits*, vol. 30, no. 11, pp. 1149–1156, Nov. 1995.

[59] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, O. Unsal, A. Cristal, and K. Mai, "Neighbor-cell assisted error correction for MLC NAND flash memories," in *ACM Int. Conf. on Meas. and Modeling of Comput. Syst.* ACM, 2014, pp. 491–504.

[60] J. E. Brewer and M. Gill, *Nonvolatile Memory Technologies with Emphasis on Flash.* Wiley, 2008.

[61] R. Cernea, L. Pham, F. Moogat, S. Chan, B. Le, Y. Li, S. Tsao, T.-Y. Tseng, K. Nguyen, J. Li *et al.*, "A 34MB/s-program-throughput 16Gb MLC NAND with all-bitline architecture in 56nm," in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2008, pp. 420–624.

[62] R. Micheloni, L. Crippa, and A. Marelli, *Inside NAND Flash Memories.* Springer, 2010.

[63] J. Wang, T. Courtade, H. Shankar, and R. Wesel, "Soft information for LDPC decoding in flash: Mutual information optimized quantization," in *IEEE Global Communications Conf. (GLOBECOM).* IEEE, 2011, pp. 5–9.

[64] G. Wu and X. He, "Reducing SSD read latency via NAND flash program and erase suspension," in *Proc. of the 10th USENIX Conf. on File and Storage Technol.*, vol. 12, 2012, pp. 10–10.

[65] S. C. Hollmer, C.-Y. Hu, B. Q. Le, P.-l. Chen, J. Su, R. Gutala, and C. Bill, "Erase verify scheme for NAND flash," Dec. 28 1999, US Patent 6,009,014.

[66] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Computing Surveys (CSUR)*, vol. 37, no. 2, pp. 138–163, 2005.

[67] J. Cooke, "Flash memory 101: An introduction to NAND flash," *Micron Technology Inc: Boise*, 2006.

[68] E. Yaakobi, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf, "Codes for write-once memories," *IEEE Trans. Inf. Theory*, vol. 58, no. 9, pp. 5985–5999, 2012.

[69] R. Gabrys and L. Dolecek, "Constructions of nonbinary WOM codes for multi-level flash memories," *IEEE Trans. Inf. Theory*, vol. 61, no. 4, pp. 1905–1919, Apr. 2015.

[70] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling," in *Proc. Design, Autom., Test in Eur. Conf. (DATE)*, Mar. 2013, pp. 1285–1290.

[71] S. Gerardin, M. Bagatin, A. Paccagnella, K. Grurmann, F. Gliem, T. Oldham, F. Irom, and D. Nguyen, "Radiation effects in flash memories," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 3, pp. 1953–1969, 2013.

[72] B. Peleato, R. Agarwal, J. Cioffi, M. Qin, and P. H. Siegel, "Towards minimizing read time for NAND flash," in *IEEE Global Communications Conf. (GLOBECOM).* IEEE, 2012, pp. 3219–3224.

[73] Y. Maeda and H. Kaneko, "Error control coding for multilevel cell flash memories using nonbinary low-density parity-check codes," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.* IEEE, 2009, pp. 367–375.

[74] L. Wang, E. Sasoglu, B. Bandemer, and Y.-H. Kim, "A comparison of superposition coding schemes," in *IEEE Int. Symp. on Information Theory (ISIT)*. IEEE, 2013, pp. 2970–2974.

[75] B. Sklar, *Digital communications*. Prentice Hall NJ, 2001, vol. 2.

[76] T. S. Rappaport *et al.*, *Wireless communications: principles and practice*. prentice hall PTR New Jersey, 1996, vol. 2.

[77] D.-h. Lee and W. Sung, "Direct and indirect measurement of inter-cell capacitance in NAND flash memory," in *Proc. of IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2014, pp. 1–6.

[78] C. Xu, X. Dong, N. P. Jouppi, and Y. Xie, "Design implications of memristor-based RRAM cross-point structures," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011, pp. 1–6.

[79] Y. Zheng, C. Xu, and Y. Xie, "Modeling framework for cross-point resistive memory design emphasizing reliability and variability issues," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*. IEEE, 2015, pp. 112–117.

[80] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, "Memristor-based memory: The sneak paths problem and solutions," *Microelectronics Journal*, vol. 44, no. 2, pp. 176–183, 2013.

[81] H. Nazarian, "Versatile RRAM technology and applications," in *Flash Memory Summit*, 2015.

[82] D. Niu, C. Xu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, "Design trade-offs for high density cross-point resistive memory," in *Proc. ACM/IEEE international symp. on Low power electronics and design*. ACM, 2012, pp. 209–214.

[83] C. Xu, D. Niu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, "Understanding the trade-offs in multi-level cell ReRAM memory design," in *50th ACM/EDAC/IEEE on Design Automation Conference (DAC)*. IEEE, 2013, pp. 1–6.

[84] J. Zhou, K.-H. Kim, and W. Lu, "Crossbar RRAM arrays: Selector device requirements during read operation," *IEEE Trans. Electron Devices*, vol. 61, no. 5, pp. 1369–1376, 2014.

[85] H.-L. Lou and C.-E. W. Sundberg, "Coded modulation for digital storage in analog memory devices," Apr. 3 2001, uS Patent 6,212,654.

[86] B. M. Kurkoski, "Coded modulation using lattices and reed-solomon codes, with applications to flash memories," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 900–908, 2014.

[87] R. Bauer and J. Hagenauer, "On variable length codes for iterative source/channel decoding," in *Proc. of Data Compression Conference(DCC)*. IEEE, 2001, pp. 273–282.

[88] N. B. Shah, "On minimizing data-read and download for storage-node recovery," *IEEE Commun. Lett.*, vol. 17, no. 5, pp. 964–967, 2013.

[89] Y.-H. Hsiao, H.-T. Lue, T.-H. Hsu, K.-Y. Hsieh, and C.-Y. Lu, "A critical examination of 3D stackable NAND flash memory architectures by simulation study of the scaling capability," in *IEEE International Memory Workshop (IMW)*. IEEE, 2010, pp. 1–4.

[90] Y. Kim, R. Mateescu, S.-H. Song, Z. Bandic, and B. V. Kumar, "Coding scheme for 3D vertical flash memory," in *IEEE Int. Conf. on Communications (ICC)*. IEEE, 2015, pp. 264–270.

[91] Y.-M. Chang, Y.-H. Chang, T.-W. Kuo, Y.-C. Li, and H.-P. Li, "Disturbance relaxation for 3D flash memory," *IEEE Trans. Comput*, vol. 65, no. 5, pp. 1467–1483, 2016.

[92] S. Buzaglo, P. H. Siegel, and E. Yaakobi, "Coding schemes for inter-cell interference in flash memory," in *IEEE Int. Symp. on Information Theory Proceedings (ISIT)*. IEEE, 2015, pp. 1736–1740.

[93] P. Hall, "On representatives of subsets," *J. London Math. Soc*, vol. 10, no. 1, pp. 26–30, 1935.