UNIVERSITY OF CALIFORNIA,
IRVINE

Coding for Efficient and Robust Storage

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Marwen Zorgui

Dissertation Committee:
Professor Zhiying Wang, Chair
Professor Hamid Jafarkhani
Professor Syed Jafar

2019

DEDICATION

To my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I am deeply grateful to my advisor Professor Zhiying Wang. I am lucky to have been advised by Zhiying, who inspired in me a deep love for research and a commitment and striving for quality. Zhiying helped me look at problems from different perspectives and encouraged me to work on problems that excite me, while always being available for guidance, insights and support. Thanks to her, I have enjoyed my PhD experience and gained much from it, not only at the academic level, but also at a personal level.

I would like to thank Professor Hamid Jafarkhani, Professor Syed Jafar for serving on my dissertation committee, and Professor Ender Ayanoglu for serving on my qualifying examination committee.

I would like to thank Cyril Guyot for offering me the opportunity to join his research team at Western Digital during the summer of 2017. I also thank my team members Robert Mateescu and Filip Blojevic, with whom and Cyril, I have had thoughtful interactions, insightful whiteboard brainstorming sessions and fun discussions.

I would like to thank my collaborator Mohammed E. Fouda, who introduced me to the sneak path problem in resistive memories which inspired a rich research direction, contributing to one chapter in my thesis. I would also like to thank my labmate Alireza Javani with whom I had fun collaborating on research papers related to age of information.

My life in Irvine throughout my PhD has been enjoyable thanks to my friends who have enriched my life and made me feel home. I am grateful to my dear friend Ahmed Eissa Khorshid, with whom I shared countless great memories. I cannot thank enough my dear friends Islam Farragh (Rabie) and Mohammed Othman (Kimo) who are always there for me. I thank Selima for the nice memories and events we shared. I also thank my friends Jordan, John, Annika, Yana, David, Mokthar, Yehia, Tarek, and Farghali. My friends from outside UCI made my life easier and enjoyable: Mensi, Rola, Zied, Ridha, Talel, Najm, Ahmed, Maroua, Sadok, Halim, Bilel, Brahim, Wassim, Khadija, Tareq, Eyleen, and Omar.

I would also like to thank my friends and colleagues from our research lab: Weiqi, Peng, and Zhen, with whom I shared a research office and endless fun stories.

Finally, I would like to thank my parents and my brothers whose unconditional support and encouragement has always been driving me throughout my academic journey.

CURRICULUM VITAE

Marwen Zorgui

EDUCATION

Doctor of Philosophy in Electrical and Computer Engineering **2019**
University of California, Irvine *Irvine, CA, USA*

Master of Science in Electrical Engineering **2015**
King Abdullah University of Science and Technology *Thuwal, KSA*

Engineering degree **2013**
Tunisia Polytechnic School *La Marsa, Tunisia*

ABSTRACT OF THE DISSERTATION

Coding for Efficient and Robust Storage

By

Marwen Zorgui

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2019

Professor Zhiying Wang, Chair

Driven by the growth of data-centric applications, efficient data storage and retrieval has become crucial for modern service providers. Storage systems are becoming increasingly complex, in response to the various challenges rising from the different layers of storage systems. This dissertation aims at designing novel data coding techniques improving the device-level reliability and reducing storage and communication costs at the system level.

We consider the repair problem in distributed storage systems (DSSs), which refers to the process of regenerating the content of inaccessible or failed nodes. Repairing failures is necessary to maintain a desired level of fault-tolerance in DSSs. Specifically, we consider the repair problem of multiple erasures in a centralized manner. Using information-theoretic tools, we derive fundamental limits for the problem. Our analysis suggests a fundamental tradeoff between the storage overhead required for reliability and the amount of data being transferred during repair. Based on that, we propose several data coding techniques operating at various points of the tradeoff, with provable optimality for certain configurations.

A standard assumption in coding theory for distributed storage is that messages are static, and the write and the read are completed instantaneously for every non-failed storage node. In this work, we study the fundamental problem of storing evolving information in distributed networks, from a coding-theoretic perspective. Specifically, we study the design of storage-

efficient algorithms for emulating atomic shared memory over an asynchronous, distributed message-passing system, a fundamental question in the field of distributed algorithms, as well as a ubiquitous problem in distributed computing applications. We propose new erasure-code based algorithms and compare their merits to existing algorithms. In particular, our algorithms feature a parameter ν that allows a system designer to tradeoff between the storage size and liveness of read operations, while guaranteeing the strictest consistency requirement, atomicity.

Finally, on the device level, we study the reliability of resistive memories (RRAMs). RRAMs are a promising non-volatile memory technology with high storage densities. However, the readout reliability of RRAMs is impaired due to the sneak path problem. In crossbar arrays, sneak paths result in storage cells with different reliability levels. Motivated by this problem, we study the framework of polar coding over channels with different reliability levels, and we propose a coding technique improving its bit error rate (BER) performance. We then apply our framework to the crossbar array and demonstrate a significant reduction in BER.

Chapter 1

Introduction

Driven by the growth of data-centric applications, efficient data storage and retrieval has become crucial for modern service providers. Network solution providers such as Cisco [1] estimate that overall mobile data traffic is expected to grow to 77 exabytes per month by 2022 (see Figure 1.1). Therefore, service providers face urgent challenges for maintaining a reliable, scalable, and efficient data storage layer, which constitutes the backbone of today's big data systems. These challenges call for improvements at all levels of storage systems, from the device to system level.

Information theory constitutes a mathematical framework for studying the compression, storage, and communication of information. It has proved to be a powerful tool for analyzing the fundamental limits of information systems and generating insights improving their performances. In particular, with the proliferation of big data systems, further research into modern storage systems has been called for and undertaken by both industrial and academic researchers. The present dissertation pertains to this line of research and seeks to develop new efficient and reliable techniques for data storage through information-theoretic tools. Issues related to modern storage systems at both the system level and the device level are investigated in the dissertation. In particular, three main subjects are considered: coding for distributed storage, coding for resistive memories, and coding-based distributed algorithms for consistent distributed storage.

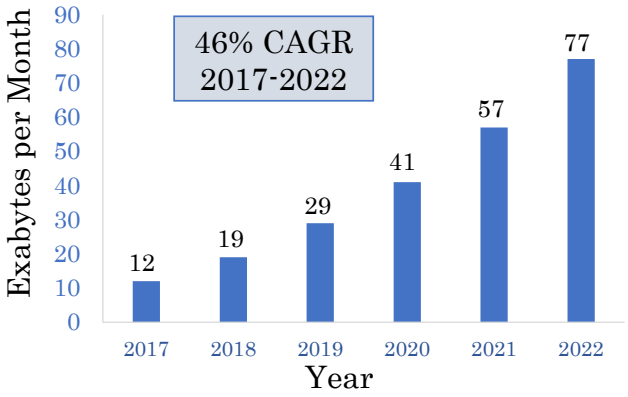


Figure 1.1: Mobile data traffic will grow at a compound annual growth rate (CAGR) of 46 percent from 2017 to 2022 [1].

1.1 Coding for distributed storage systems

1.1.1 The repair problem

Facing an ever-growing demand for data storage, distributed storage systems (DSSs), consisting of several nodes connected over networks, have become an appealing architecture in current file systems. Such systems are expected to store, distribute, and access large amounts of data on a daily basis. Examples of such systems include Google's Big Table [7], Hadoop HDFS [8], and Microsoft Azure [9].

A serious challenge for DSSs, however, is node failure, which may occur for several reasons. Naturally, as the system scales, the number of node failures scales with it. Moreover, for cost reduction purposes, DSS providers usually opt for using a large number of cheap, fault-prone storage devices rather than a small number of expensive highly reliable devices. For these reasons, node failure frequency has become a challenging factor for DSSs. For instance, in [10], it is reported that the annual replacement rate of the storage disks is commonly around 2 to 4 %, and can be as high as 13 % in some systems. While the aforementioned failure rate takes into consideration only hardware failure, another common type of node failures are *temporary* failures, in which the node content is temporarily unavailable. The

temporary unavailability of a node can be triggered by several factors, such as power outage, maintenance interrupts, and software crashes and updates. For example, it is reported in [11] that in a system of 3000 nodes at Facebook, it is quite typical to have 20 or more node failures per day.

The robustness of a DSS is related to its capacity to tolerate permanent and/or transient failures. In the presence of failure, the storage system's functionality should not be disturbed and the user data should still be available. To meet such a robustness requirement, *redundancy* should be provisioned in the design of the storage system. Traditionally, simple replication of data has been adapted in many systems. For instance, Google file systems opted for a triple replication policy [12]. However, for the same redundancy factor, replication systems fall short of providing the highest level of reliability. On the other hand, erasure codes can be optimal in terms of the redundancy-reliability tradeoff. In erasure codes, a file of size $\mathcal{M}$ is divided into k fragments, each of size $\frac{\mathcal{M}}{k}$. The k fragments are then encoded into n fragments using an (n, k) maximum distance separable (MDS) code and then stored at n different nodes. Using such a scheme, the data is guaranteed to be recovered from any $n - k$ node erasures, providing the highest level of worst-case data reliability for a given redundancy. Examples of MDS codes are Reed-Solomon (RS) codes. A $(14, 10)$ RS code is used in Facebook distributed storage systems [13].

When a failure occurs, the content of the failed node needs to be regenerated in order to maintain the system reliability or to satisfy a user request. This process is known as *repair*. The amount of information transmitted during a repair process is termed *bandwidth*. Bandwidth is a significant parameter in DSSs, since it directly translates to network traffic cost and data access latency. Proportional to the data size of the failed node, bandwidth becomes more and more critical with the growth of the storage node capacity. In a replication-based scheme, repair is easy, since identical copies of the content of a failed node are available, and it is enough to make a new copy from a surviving replica. However, traditional erasure

symbol 1	symbol 2	symbol 3	symbol 4
a	c	$a+c$	$a+d$
b	d	$b+d$	$b+c+d$

Table 1.1: EVENODD code [6]: an example of an MSR code. The code uses the binary field $\mathbb{F}_2$. "+" denotes XOR.

codes exhibit a limiting drawback when it comes to repairing a node. Indeed, to repair a single node storing a fragment of size $\frac{\mathcal{M}}{k}$, traditional erasure codes require downloading the entire data of size $\mathcal{M}$. This k-expansion factor makes erasure codes impractical for some applications in DSSs.

In the last decade, the repair problem has gained increasing interest and motivated research for a new class of erasure codes with better repair capabilities. The seminal work in [14] proposed a new class of erasure codes, referred to as *regenerating codes*, which addresses the repair bandwidth problem.

An $(\mathcal{M}, n, k, d, \alpha, \beta)$ regenerating code consists of n storage nodes, each storing α amount of information. Regenerating codes satisfy the *reconstruction property*. That is the entire data $\mathcal{M}$ can be recovered from any k of the n nodes. Moreover, the repair is carried out by downloading $\beta \leq \alpha$ amount of information from any $d \geq k$ nodes (also called helpers). We illustrate below an example of a minimum storage regenerating (MSR) code, known as EVENODD code [6].

Example 1.1. *Consider* $(n, k, d) = (4, 2, 3)$. *The code is described in Table 1.1. We refer to the codeword symbols as symbols. Each symbol is a binary vector of length* 2*, namely, an element of* $\mathbb{F}_2^2$. *Therefore,* $\alpha = 2$. *The information bits are* $a, b, c, d \in \mathbb{F}_2$. *That is,* $\mathcal{M} = 4$. *It can be checked that from any* 2 *symbols, all the information bits can be recovered. Therefore,* $k = 2$.

The repair of each of the 4 *symbols can be carried out efficiently. For example, to repair symbol* 1*, symbol* 2 *sends* c*, symbol* 3 *sends* $a + c$*, and symbol* 4 *sends* $(a + d) + (b + c + d) =$

$(a + c) + b$. *Other symbols can be repaired in a similar way. Since each helper contributes* $\frac{1}{2}$ *of its size* ($\beta = 1 = \alpha/2$)*, we say that each helper contributes a half symbol in* $\mathbb{F}_2^2$ *in the repair process. More generally, the contribution of a helper in the repair process is a fraction of a symbol.*

In the regenerating codes framework, the goal is to reduce the repair bandwidth by contacting a large number of helpers, where $d \geq k$. It is also desirable in some systems to have a repair strategy where the number of involved helpers is small, that is $d < k$. The number of helpers in the latter scenario is known as *locality*, and codes with a specific locality parameter are known are *locally repairable codes*. Various bounds and code constructions for locally repairable codes have also been proposed in the literature, e.g., [15, 16]. The focus of this dissertation is on the regenerating codes framework.

1.1.2 Multi-node recovery

In many practical scenarios, such as in large scale storage systems, multiple failures are more frequent than a single failure. Moreover, many systems (e.g., [17]) apply a lazy repair strategy, which seeks to limit the repair cost of erasure codes. Instead of immediately repairing every single failure, a lazy repair strategy waits until e erasures occur, $e \leq n - k$, then, the repair is done by downloading the equivalent of the total information in the system to regenerate the erased nodes. A natural question of interest is whether one can still reduce the amount of download in such scenarios, similar to the single erasure case. We distinguish between two ways of repairing multiple failures.

Cooperative regenerating codes: In this framework, the repair is done in a distributed way. Each replacement node of the lost nodes first downloads information from d nodes (helpers). Then, the replacement nodes exchange information between themselves before regenerating the lost nodes.

Centralized regenerating codes: Upon failure of e nodes, the repair is carried out in a centralized way by contacting any d helpers, and downloading β amount of information from each helper.

In this dissertation, the focus is on studying the fundamental limits of the centralized multi-node repair problem. A centralized repair of multiple node failures can be desirable in many scenarios. For instance, there are situations in which, due to architectural constraints, it is more desirable to regenerate the lost nodes at a central server before dispatching the regenerated content to the replacement nodes [17]. One may think of a rack-based node placement architecture [18] in which failures frequently occur to nodes corresponding to one rack. In this scenario, a centralized repair of the entire rack is favorable as opposed to repairing the rack on a per-node basis. Furthermore, [18] showed that a centralized repair framework can have interesting applications in communication-efficient secret sharing. Finally, centralized repair can be used in a broadcast network, where the repair information is transmitted to all replacement nodes (e.g. [19]).

For clarity of exposition, we include Table 1.2, which contains a list of symbols and abbreviations used throughout our study of the centralized multi-node repair problem.

Prior to our works, no explicit expression for the storage-bandwidth tradeoff was known for multi-node centralized repair. Moreover, explicit code constructions only existed for high-rate and minimum storage regenerating codes (codes with smallest α). In the first part of the dissertation, we present the explicit tradeoff for *functional repair*, where the repaired information is not exactly the same as the lost information, and we show several impossibility results for *exact repair*. We find, contradictory to the single-node repair, the functional tradeoff point with minimum bandwidth β is not achievable under exact repair. Besides, we present explicit codes for exact multi-node repair, including low-rate minimum storage regenerating codes, adaptive minimum bandwidth codes, and codes for the interior points between these two extreme points.

Symbol	Meaning
$\mathcal{M}$	File size
n	Number of storage nodes
α	Storage per node
β	Bandwidth per helper
$\bar{\alpha}$	Normalized storage per node
$\bar{\beta}$	Normalized bandwidth per helper
k	Reconstruction parameter
d	Number of helpers
γ	Total bandwidth from all helpers
e	Number of erasures
MDS	Maximum distance separable
MSR	Minimum storage regenerating
MBR	Minimum bandwidth regenerating
MSMR	Minimum storage multi-node regenerating
MBMR	Minimum bandwidth multi-node regenerating
MBCR	Minimum bandwidth cooperative regenerating

Table 1.2: List of symbols and abbreviations for regenerating codes.

1.2 Coding for resistive memories

Transistor-based memories are rapidly approaching their maximum density per unit area, which has stimulated growing research into new storage devices. Emerging nonvolatile memories (NVMs), such as phase change memory (PCRAM), ferroelectric memory (FeRAM), spin transfer torque magnetic memory (STT-MRAM), and resistive memory (RRAM), have shown high potential as alternatives for floating-gate-based nonvolatile memories [2]. Figure 1.2 shows the emerging NVMs' capacities in the recent years.

RRAMs are considered one of the best candidates for the next generation nonvolatile memory due to their high reliability, fast access speed, multilevel capabilities and stack-ability creating 3D memory architectures. Moreover, RRAMs have been proposed as an attractive candidate for hardware-based implementation of neuro-inspired computing [20].

Crossbar arrays are a common architecture for RRAMS. To achieve higher density memories,

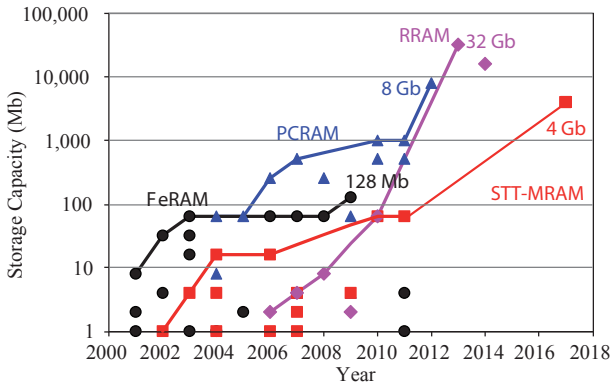


Figure 1.2: Trends in memory capacity for emerging NVMs adopted from [2].

switching devices are sandwiched between the crossbar metal layers, without using access devices such as transistors, diodes and selectors. The main disadvantage of the selector-less (gate-less) crossbar-based memories is the *sneak path* effects which limit the readability of the array.

The sneak path problem arises because of two reasons. First, the existence of many paths from the inputs to the outputs through which current flows. Fig. 1.3 illustrates the sneak path problem in a (2×2) crossbar array. The sneak path currents highly disturb the desired current which is proportional to the stored resistance. This can be ideally eliminated by parallel reading of the entire row, where all the columns (bitlines) are grounded to absorb the current as illustrated in Fig. 1.3 [21]. Second, the existence of wire resistance is inevitable in nano-crossbar arrays and can reach up to 90 Ω per cell for nano feature sizes (i.e.$F = 5nm$) [21]. The wire resistance creates voltage drops, which are functions of the stored data and accumulate throughout the array. Fig. 1.4 shows the measured current of each cell in a (32×32) array with 25Ω wire resistance, storing random data. Clearly, the sensed current of low resistance state decreases in both vertical an horizontal directions in the array. The top left cells have less disturbed behavior and stored ones and zeros are distinguishable. On the other hand, the bits of the right-bottom cells are indistinguishable due the read margin overlap. Figure 1.5 shows the histogram of the measured currents of the 1-st, the 16-th and the 32-nd bitline (column), respectively. Clearly, the larger the bitline index is, the more

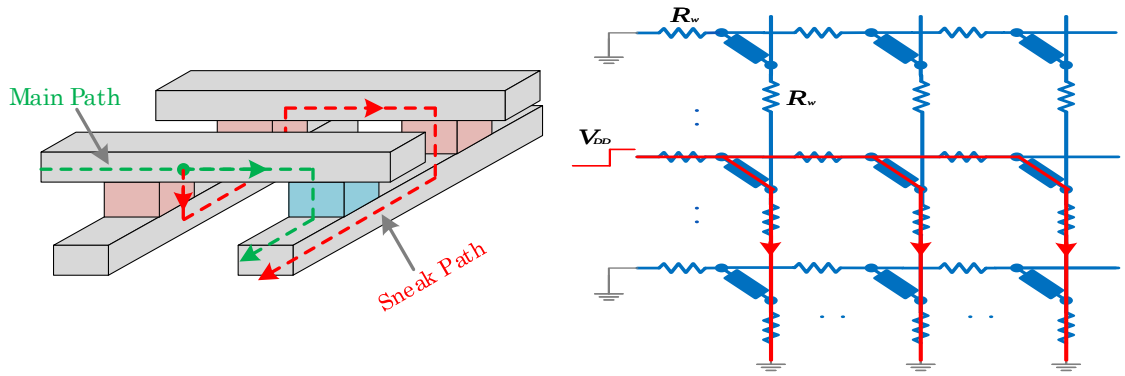


Figure 1.3: Crossbar array with the sneak path problem with (a) one port reading, on the left and (b) parallel reading technique, on the right.

errors occur. Thus, the crossbar array cells have varying reliability levels.

While traditional error correcting codes can be used in resistive memories, they are not adapted for the specific type of storage channel and do not give the best reliability level. In this dissertation, we propose error correcting codes that address the varying reliability nature of the crossbar array cells induced by the sneak path currents, and evaluate their performance using a SPICE-like simulator [21].

1.3 Coding for distributed computing

We study the fundamental problem of storing *evolving* information in distributed networks from a coding-theoretic perspective. A standard assumption in coding for distributed storage is that a *static* message is to be written and read, and the write and the read are completed *instantaneously* for every non-failed storage node. On the other hand, emulation of shared memory in an asynchronous and unreliable message passing network is a fundamental question in the field of distributed algorithms. Such emulation algorithms make it possible to view the shared-memory model as a higher-level language for designing algorithms in asynchronous distributed systems. In practice, shared memory emulation forms an integral part of several data storage products, such as Amazon Dynamo [22], Apache Cassandra [23], and

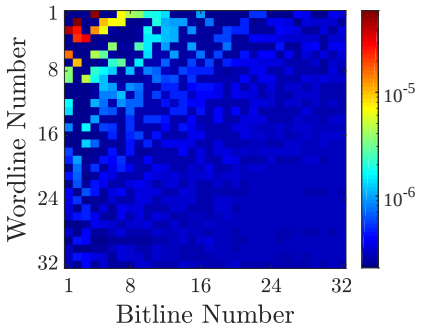


Figure 1.4: Measured current per cell

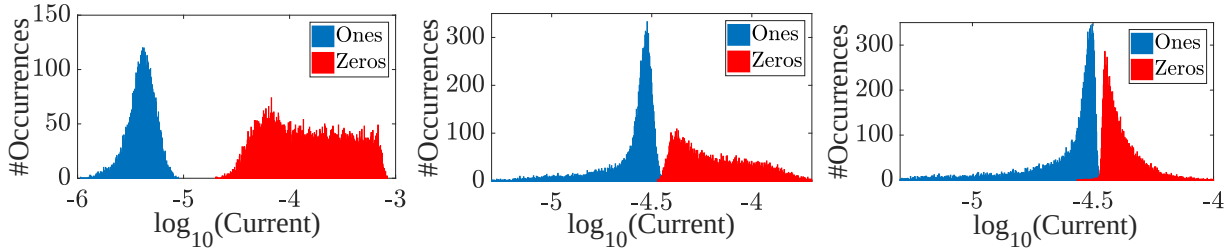


Figure 1.5: Histogram for bitline number 1, 16, and 32, respectively.

CouchDB [24].

In this problem, one allows many writes and reads with delays that are not known a priori, and each write has a version of the dynamic message to be stored. An emulation algorithm should be robust to a predetermined level of node failures (i.e., fault-tolerance capability), which requires the introduction of some form of redundancy in the system. Moreover, shared memory emulation algorithms need to satisfy certain *consistency* and *liveness* requirements. Informally, a consistency requirement specifies what operation executions are allowed to happen in the system, while a liveness requirement specifies the conditions under which client operations should complete. In particular, of relevance to our work, we describe the following two properties which are desirable properties in practical systems:

Atomicity: A shared atomic object is one that supports concurrent access by multiple clients and where the observed global external behaviors look like the object is being accessed sequentially.

Wait-freedom: An operation of a non-failed client is guaranteed to terminate provided that

the number of server failures is bounded, and irrespective of the failures of other clients.

Atomicity and wait-freedom are among the strictest consistency and liveness requirements, and variations of these requirements exist and have been implemented in several practical systems [25].

The seminal paper [26] showed for the first time that it is possible to emulate an atomic wait-free shared memory in an asynchronous and unreliable message-passing networks. Attiya, Bar-Noy, and Dolev [26] proposed a fault-tolerant algorithm (ABD algorithm) that is based on replication. The ABD algorithm is quorum-based, where read and write operations communicate with a quorum of nodes (a quorum is a group of at least $N - f$ servers, $N = 2f + 1$ being the number of nodes, and f is the server failure tolerance level of the system) through read and write protocols. As ABD uses replication, and since the read and write protocols require multiple communication phases where entire replicas are sent, the algorithm incurs high storage and communication costs.

In order to reduce the storage and communication overhead of replication-based algorithms, it is natural to seek to apply erasure codes to the shared memory emulation problem. However, compared to the assumptions of the shared memory emulation model, traditional erasure coding for distributed storage neglects two temporal aspects: (1) the stored messages are dynamic, and (2) the write and the read progress over time. Introducing these new factors makes the problem non-trivial and requires careful management. For instance, assume that a read operation is concurrent with several writes, including failed writes, i.e., writes invoked by failed writers. Then, in erasure coding, it is possible that the reader obtains information of different values, but does not have a sufficient number of coded symbols to decode and return any value. With various guarantees, several algorithms based on traditional erasure codes have been proposed [27, 28, 29]. The structure of the algorithms are more complicated than ABD algorithm in order to handle the difficulty brought by concurrent writes and coding. Moreover, the storage and/or communication cost of these algorithms still calls

for further improvement.

In this dissertation, we are interested in storage-efficient shared memory emulation algorithms. We utilize multi-version codes [30], which are erasure codes tailored to mimic the above temporal aspects. We propose new erasure-code based emulation algorithms and compare their merits to existing erasure-code based algorithms. In particular, our algorithms feature a parameter ν that allows a system designer to tradeoff between the storage size and liveness of read operations, while guaranteeing the strictest consistency requirement, atomicity. Several of our algorithm has the advantages of simple algorithm structure, which requires little computing and memory for the storage nodes. Moreover, the storage cost of our algorithm is reduced by up to a factor of 2 compared to previous methods.

1.3.1 Dissertation organization

The centralized repair problem is the main focus of this dissertation, and is studied over the next three chapters. In Chapter 2, we study centralized functional repair and also derive outer bounds for exact repair. In Chapter 3 and 4, we present our exact-repair code constructions and analyze their properties. The problem of coding for resistive memories is studied in Chapter 5. In Chapter 6, we investigate the shared memory emulation problem and propose our corresponding schemes. Finally, Chapter 7 concludes the dissertation with summary and future directions.

In each chapter, we will introduce the respective motivation, related work, problem settings, and our contributions. Some technical details such as proofs of certain theorems are relegated to the appendix after each chapter. Most of the results of this dissertation are a result of our work in [31, 32, 33, 34, 3, 35, 36, 37].

Notation. Throughout the dissertation, we will use the following notation. Chapter-specific notations will be introduced where needed. For two sets $\mathcal{A}, \mathcal{B}$, $\mathcal{A} \backslash \mathcal{B}$ denotes the set of

elements that are in $\mathcal{A}$ but not in $\mathcal{B}$. $|\mathcal{A}|$ denotes the size of $\mathcal{A}$. The symbol $\mathbb{1}_{\{E\}}$ denotes the indicator function of an event E, which is 1 if E is true, and 0 otherwise. The notations $e \mid k$ and $e \nmid k$ are used to denote whether k is a multiple of e, or not, respectively. For a matrix A, $|A|$ denotes its determinant, A^t denotes its transpose, and $A_{i,j}$ refers to its entry at position (i, j). I_n denotes the identity matrix of size n and $\text{diag}\{\lambda_1, \ldots, \lambda_n\}$ denotes the $(n \times n)$ diagonal matrix with the corresponding elements. Vectors are denoted with lower-case bold letters. $\mathbf{u} = [u_1, \ldots, u_m]$ denotes a vector of length m. $[n]$ denotes the set of elements $\{1, \ldots, n\}$. Note that the notation $[k]$ may refer to a vector of size 1, or the set $\{1, \ldots, k\}$, however the meaning is clear from the context. $[m, n]$ denotes the set $\{m, m + 1, \ldots, n\}$. $\mathbf{e}_i$ denotes the i-th standard basis vector whose dimension is clear from the context. The notations $\lceil \cdot \rceil, \lfloor \cdot \rfloor$ represent the ceiling and the floor functions.

Chapter 2

Converse bounds for centralized multi-node regenerating codes

2.1 Introduction

Ensuring data reliability is of paramount importance in modern storage systems. Reliability is typically achieved through the introduction of redundancy. While traditionally simple replication has been adopted in many systems, erasure codes have been gaining more grounds in practical systems thanks to their reduced storage fingerprint. The repair problem, consisting of regenerating the content of failed or unavailable nodes is a frequent task in modern distributed storage systems (DSSs), generating huge amount of traffic across networks. Traditional erasure codes are oblivious to the repair problem, which makes their integration in DSSs impractical. Regenerating codes [14] were introduced to tackle the repair problem, while leveraging the benefits of erasure codes.

If we recover the exact same information as the failed node, we call it *exact repair*, otherwise we call it *functional repair*. Using network coding [38, 39], the authors in [14] proved that it is possible to construct functional regenerating codes. Interestingly, [14] showed that the repair bandwidth per node β can be reduced, if the storage per node α is increased. Moreover, the entire storage-bandwidth region has been characterized under functional repair [14]. Namely,

an optimal $(\mathcal{M}, n, k, d, \alpha, \beta)$ regenerating code satisfies

$$\mathcal{M} = \sum_{i=0}^{k-1} \min\{\alpha, (d-i)\beta\}. \tag{2.1}$$

The parameters and notations are introduced in Table 1.2 in Chapter 1. Equation (2.1) describes the fundamental tradeoff between the storage capacity α and the bandwidth β, under functional repair. Two extreme points can be obtained from the tradeoff. Minimum storage regenerating (MSR) codes correspond to the highest storage efficiency with $\alpha = \frac{\mathcal{M}}{k}$, while minimum bandwidth regenerating (MBR) codes achieve the lowest possible bandwidth at the expense of extra storage per node. Following [14], there has been a flurry of interest in designing exact-repair regenerating codes that achieve the optimal tradeoff, focusing mainly on the extreme MSR and MBR points, e.g., [40, 41, 42, 43, 44, 45, 46, 47, 48, 49].

In this chapter, we consider the centralized repair problem of multiple failures. Our first objective is to characterize the functional repair tradeoff between the storage per node α and the repair bandwidth β. We then seek to investigate converse bounds for exact repair, and as a result understand the achievability of the functional tradeoff and derive outer bounds under exact repair.

Related work

There has been a growing literature focused on understanding the fundamental limits of exact-repair regenerating codes for single erasure. The authors in [50] showed that most points that are between the MBR and MSR points in the tradeoff of (2.1), are not achievable for exact repair. Other outer bounds for exact repair include [51, 5, 52] for general parameters, and [53] for linear codes.

Cooperative regenerating codes (also known as coordinated regenerating codes) have been

studied to address the repair of multiple erasures [54, 55] in a distributed manner. The functional repair tradeoff for cooperative repair has been established in [55], and various exact-repair code constructions and outer bounds under the cooperative repair model has been proposed [55, 56, 57, 58, 59].

In [18], the authors studied the centralized repair of multiple failures for two family of codes: MDS codes and minimum bandwidth multi-node repair codes, defined as codes satisfying the property of having the downloaded information $d\beta$ matching the entropy of e nodes*. The authors in [19] studied the problem of broadcast repair for wireless distributed storage which is equivalent to the model we study in this paper. Specifically, the functional repair tradeoff is obtained as the solution to an optimization problem, which is solved numerically.

Contributions of the chapter

The main contributions of this chapter are summarized as follows.

- We first establish the explicit tradeoff between the repair bandwidth and the storage size for functional repair (Theorems 2.1, 2.2, 2.3). We obtain the tradeoff using information flow graphs. From the functional tradeoff, we characterize the minimum storage multi-node repair (MSMR) point, and the minimum bandwidth multi-node repair (MBMR) point.

- We prove that, to our surprise, functional MBMR point is not achievable for linear exact repair codes for $1 < e < k$ (Theorems 2.4, 2.5), while linear codes achieve such point for single erasure [60].

- We show that the functional repair tradeoff is not achievable under exact repair for interior points between MBMR and MSMR points, except for maybe a small portion

*The definition of minimum bandwidth multi-node repair codes in this dissertation is simply the minimum bandwidth point on the functional tradeoff, which is different from [18] for $e \nmid k$ as shown later in this chapter.

near the MSMR point, for k, d being multiples of e and $k > 2e$ (Theorems 2.6, 2.7), which parallels the results for single erasure repair [50].

- Finally, for linear codes, we derive exact-repair outer bounds that improve upon the functional repair outer bounds for certain configurations (Theorem 2.8), and illustrate its performance under several settings.

2.2 Functional storage-bandwidth tradeoff

2.2.1 System model

The centralized mutli-node repair problem is characterized by parameters $(\mathcal{M}, n, k, d, e, \alpha, \beta)$. We consider a distributed storage system with n nodes storing $\mathcal{M}$ amount of information. The data elements are distributed across the n storage nodes such that each node can store up to α amount of information. Every node corresponds to a codeword symbol. The system should satisfy the following two properties:

- Reconstruction property: a data collector (DC) connecting to any $k \leq n$ nodes should be able to reconstruct the entire data.

- Regeneration property: upon failure of e nodes, a central node is assumed to contact d helpers, $k \leq d \leq n-e$, and download β amount of information from each of them. New replacement nodes join the system and the content of each is determined by the central node. β is called the repair bandwidth. The total bandwidth is denoted $\gamma = d\beta$.

We consider functional repair and exact repair. In the former case, the replacement nodes are not required to be exact copies of the failed nodes, but the repaired code should again satisfy the above two properties. Our objective is to characterize the tradeoff between the

storage per node α and the repair bandwidth β under the centralized multiple failure repair framework.

In the chapter, we write $k = \eta e + r$, such that $\eta = \lfloor \frac{k}{e} \rfloor$ and $0 \leq r \leq e-1$. We now study the fundamental tradeoff between the storage size α and the repair bandwidth β for e erasures under functional repair. We use the technique of evaluating the minimum cut of a multicast information flow graph similar to the single erasure codes [14] and the cooperative regenerating codes [55].

2.2.2 Information flow graphs

The performance of a storage system can be characterized by the concept of information flow graphs (IFGs). Our constructed IFG depicts the amount of information transferred, processed and stored during repair. We design our IFG with the following different kinds of nodes (see Figure 2.1). It contains a single source node s that represents the source of the data object. Each storage node $x^i, i \in [n]$, of the IFG is represented by two distinct nodes: an input storage node x^i_{in} and an output storage node x^i_{out}. Each output node x^i_{out} is connected to its input node x^i_{in} with an edge of capacity α, reflecting the storage constraint of each individual node. The information flow graph is formed with n initial storage nodes, connected to the source node with edges of capacity ∞. The IFG evolves with time whereupon failure of e nodes, e new nodes simultaneously join the system. Each of the replacement nodes $x^j, j \geq n$, is similarly represented by an input node x^j_{in} and an output node x^j_{out}, linked with an edge of capacity α. To model the centralized repair nature of the system, we add a virtual node $x^i_{virt}, i \geq 1$, that links the d helpers to the new storage nodes. The virtual node x^i_{virt} is connected to the d helpers through d incoming edges each of capacity β. The same node x^i_{virt} is also connected to the input nodes x^j_{in} of the replacement nodes, with edges of capacity ∞. We define a *repair group* to be any set of e nodes that have been repaired simultaneously. In an IFG, a repair group is then associated with the virtual

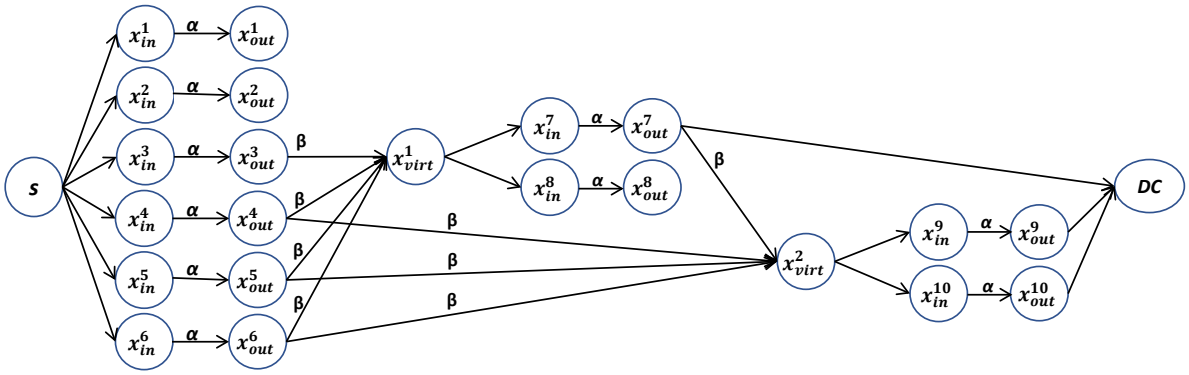


Figure 2.1: Example of an information flow graph: $k = 3, d = 4, n = 6, e = 2$. The unlabeled edges have capacity ∞. Nodes 1 and 2 are repaired in the first stage and nodes 3 and 8 are repaired in the second stage. A data collector connecting to any 3 nodes should be able to recover the entire information.

node that performs the repair operation.

Each IFG represents one particular history of the failure patterns. The ensemble of IFGs is denoted by $\mathcal{G}(n, k, d, e, \alpha, \beta)$. For convenience, we drop the parameters whenever it is clear from the context. Given an IFG $G \in \mathcal{G}$, there are $\binom{n}{k}$ different data collectors connecting to k output storage nodes in G with edges of capacity ∞. The set of all data collectors (DCs) nodes in a graph G is denoted by $\mathrm{DC}(G)$. For an IFG $G \in \mathcal{G}$ and a data collector $t \in \mathrm{DC}(G)$, we partition the nodes of G into two disjoint sets, $U, \bar{U}$, $s \in U, t \in \bar{U}$, and a cut is the sum of the capacities for all edges from nodes in U to nodes in $\bar{U}$. The minimum cut (min-cut) value separating the source node s and the data collector t is denoted by $\mathrm{mincut}_G(s, t)$.

2.2.3 Network coding analysis

The key idea behind representing the repair problem by an IFG lies in the observation that the repair problem can be cast as a multicast network coding problem [14]. Celebrated results from network coding [38, 39] are then invoked to establish the fundamental limits of the repair problem.

According to the max-flow bound of network coding [38], for a data collector to be able

to reconstruct the data, the min-cut separating the source to the data collector should be larger than or equal to the data object size $\mathcal{M}$. Considering all possible data collectors and all possible failure patterns, and assuming that the number of failures/repairs is bounded, the following condition is necessary and sufficient for the existence of centralized multi-node repair codes [14, Proposition 1]

$$\min_{G\in\mathcal{G}} \min_{t\in \mathrm{DC}(G)} \mathrm{mincut}_G(s,t) \geq \mathcal{M}. \tag{2.2}$$

Analyzing the minimum cut of all IFGs results in the following theorem.

Theorem 2.1. *For fixed system parameters $(\mathcal{M}, n, k, d, e, \alpha, \beta)$, assuming that the number of failures/repairs is bounded, regenerating codes satisfying the centralized multi-node repair condition exist if and only if*

$$\mathcal{M} \leq \min_{\mathbf{u}\in\mathcal{P}} f(\mathbf{u}), \tag{2.3}$$

where

$$f(\mathbf{u}) = \sum_{i=1}^{g} \min(u_i\alpha, (d - \sum_{j=1}^{i-1} u_j)\beta), \tag{2.4}$$

$$\mathcal{P} = \{\mathbf{u} = [u_1, \ldots, u_g] : 1 \leq u_i \leq e, g \in \mathbb{N}, \sum_{i=1}^{g} u_i = k\}. \tag{2.5}$$

Note that g in (2.5) corresponds to the support of $\mathbf{u}$, and it satisfies $\lceil \frac{k}{e} \rceil \leq g \leq k$. We call the vector $\mathbf{u} \in \mathcal{P}$ a recovery *scenario*.

Proof. Consider a data collector that connects to a subset of k nodes $\{x^j_{out} : j \in I\}$, where I is the set of k contacted nodes. Then, the reconstruction process can be described by a scenario $\mathbf{u} \in \mathcal{P}$ as follows. The size of the support of $\mathbf{u}$ corresponds to the number of repair groups of size e taking part in the reconstruction process, while u_i corresponds to the number

of nodes contacted from repair group i.

We first show that the min-cut minimized over all possible IFGs and over all data collectors is lower bounded by $\min_{\mathbf{u}\in\mathcal{P}} f(\mathbf{u})$. As all incoming edges of DC have infinite capacity, we only examine cuts $(U, \bar{U})$ with $s \in U$ and $\{x^i_{out} : i \in I\} \subseteq \bar{U}$. Every directed acyclic graph has a topological sorting, which is an ordering "$<$" of its vertices such that the existence of an edge $x \to y$ implies $x < y$. We recall that nodes within the same repair group are repaired simultaneously, hence it is possible that all input (or output) nodes in a repair group are adjacent in the ordering. We thus order the g repair groups connected to DC according to the sorting. Since nodes are sorted, nodes in the i-th repair group do not have incoming edges from nodes in the j-th repair group, with $j > i, i, j \in [g]$. Considering the i-th repair group, we consider two cases:

- $x^i_{virt} \in U$: as x^i_{virt} is connected with edges of capacity ∞ to x^j_{in}, for j in repair group i, we only consider the case $x^j_{in} \in U, x^j_{out} \in \bar{U}$, for all j in repair group i such that x^j_{out} is connected to the DC. The contribution to the cut is $u_i\alpha$.

- $x^i_{virt} \in \bar{U}$: since the i-th repair group is the topologically i-th repair group, at most $\sum_{j=1}^{i-1} u_j$ edges come from output nodes in $\bar{U}$ and are not part of the cut. The contribution to the cut is at least $(d - \sum_{j=1}^{i-1} u_j)\beta$.

Thus, the contribution of the i-the repair group is at least $\min(u_i\alpha, (d - \sum_{j=1}^{i-1} u_j)\beta)$. Finally, summing all contributions from different repair groups and considering the worst-case for $\mathbf{u} \in \mathcal{P}$ implies that

$$\min_{G\in\mathcal{G}} \min_{t\in \mathrm{DC}(G)} \mathrm{mincut}_G(s,t) \geq \min_{\mathbf{u}\in\mathcal{P}} \Big(\sum_{i=1}^{g} \min(u_i\alpha, (d - \sum_{j=1}^{i-1} u_j)\beta)\Big),$$

with $\mathcal{P}$ defined as in (2.5).

Now, considering a scenario $\mathbf{u} \in \mathcal{P}$, we show that there exists an information flow graph where the the min-cut is equal to $f(\mathbf{u})$. In this graph, there are initially nodes labeled from 1 to n, and we consider g repair groups (i.e., eg newcomers). Nodes in the i-th repair group are labeled from $n + ie + 1$ to $n + ie$. The i-th repair group connects to nodes $n - d - (\sum_{j=1}^{i-1} u_i) + 1, \ldots, n$, and the first u_j nodes from repair group j, for $j \leq \ i - 1$. Figure 2.1 illustrates the graph for $n = 6, k = 3, d = 4, e = 2, \mathbf{u} = [1, 2]$. Consider a DC that connects to the first u_i nodes from the i-th repair group, for $i \in [g]$. According to the first part of the proof, the min-cut $(U, \bar{U})$ should be as follows. For each $i \in [g]$, if $u_i\alpha \leq (d - \sum_{j=1}^{i-1} u_j)\beta)$, then we include the u_i output nodes of the contacted nodes from the i-th repair group in U and their corresponding output nodes in $\bar{U}$; otherwise, we include x^i_{virt} in $\bar{U}$. Then, this cut $(U, \bar{U})$ achieves $f(\mathbf{u})$. Hence, the min-cut minimized over all graphs and over all data collectors should be equal to $\min_{\mathbf{u}\in\mathcal{P}} f(\mathbf{u})$. The theorem follows according to the necessary and sufficient condition in (2.2). □

Our characterization of Theorem 2.1 relies on the boundedness assumption of the total number of failures/repairs. A future direction is to investigate the correctness of Theorem 2.1 for arbitrary number of failures/repairs, similar to [55, 61].

2.2.4 Solving the minimum cut problem

In this section, we derive the structure of the optimal scenario $\mathbf{u}$ in (2.3) for any set of parameters (α, β). For instance, we show that for $me < k \leq (m + 1)e$, the number of optimal repair groups g^* (the support of $\mathbf{u}$) is equal to $m + 1$. The result is formalized in the following theorem. Recall that we denote $\eta = \lfloor k/e \rfloor, r = k - \eta e$.

Theorem 2.2. *For fixed system parameters $(\mathcal{M}, n, k, d, e, \alpha, \beta)$, functional regenerating codes*

satisfying the centralized multi-node repair condition exist if and only if

$$\mathcal{M} \leq f(\mathbf{u}^*) = \sum_{i=1}^{\lceil \frac{k}{e} \rceil} \min(u_i^* \alpha, (d - \sum_{j=1}^{i-1} u_j^*)\beta),$$

where

$$\mathbf{u}^* = \begin{cases} [k], & \text{if } k \leq e, \\ [\underbrace{e, \ldots, e}_{\eta \text{ times}}], & \text{else if } k = \eta e, \\ [r, \underbrace{e, \ldots, e}_{\eta \text{ times}}], & \text{else if } k = \eta e + r \text{ and } \alpha \leq \frac{d+\eta r - \eta e}{r}\beta, \\ [\underbrace{e, \ldots, e}_{\eta \text{ times}}, r], & \text{otherwise}, \end{cases} \tag{2.6}$$

where $0 < r < e$.

Note that $[k]$ in (2.6) means a vector with a single entry k. We note that [18, 19] have independently developed Theorem 1 or an equivalent of Theorem 1, without entirely characterizing the optimal solution. The authors of [62] independently proved via a different approach Theorem 2, except for the last case in (2.6).

We denote by $[\mathbf{v}, \mathbf{u}, \mathbf{w}]$ the vector that is the concatenation of the vectors $\mathbf{v}, \mathbf{u}, \mathbf{w}$. The next lemma shows that the minimum cut can be obtained by optimizing any subsequence of $\mathbf{u}$ first. The proof follows directly from the definition of $f()$ in (2.4) and is omitted.

Lemma 2.1. *Consider vectors* $\mathbf{v}, \mathbf{w}, \mathbf{u}, \mathbf{u}'$ *such that* $\sum_i u_i = \sum_i u'_i$. *If* $f(\mathbf{u}) \geq f(\mathbf{u}')$, *then,* $f([\mathbf{v}, \mathbf{u}, \mathbf{w}]) \geq f([\mathbf{v}, \mathbf{u}', \mathbf{w}])$.

Lemma 2.2. *Let* α, β *be non-negative reals,* u_1, u_2, d, e, s, l *be non-negative integers such that*

$u_1 + u_2 = s \leq e$, *then the following inequality holds*

$$f([u_1, \underbrace{e, \ldots, e}_{l \text{ times}}, u_2]) \geq \min(f([s, \underbrace{e, \ldots, e}_{l \text{ times}}]), f([\underbrace{e, \ldots, e}_{l \text{ times}}, s])),$$

where $f(\mathbf{u})$ *is defined as in (2.4).*

Proof. To prove the result, we cast it as an optimization problem:

$$\begin{aligned} \underset{\mathbf{u}=[u_1,u_2]}{\text{minimize}} \quad & \min(u_1\alpha, d\beta) + \sum_{i=0}^{l-1} \min(e\alpha, (d - ie - u_1)\beta) + \min(u_2\alpha, (d - (l+1)e - u_1)\beta) \\ \text{subject to} \quad & 0 \leq u_1 \leq s, \\ & 0 \leq u_2 \leq e, \\ & u_1 + u_2 = s. \end{aligned} \tag{2.7}$$

Substituting u_2 by $s - u_1$ in (2.7), using the identity $\min(x, y) = \frac{x+y-|x-y|}{2}$ and after eliminating constant terms, (2.7) becomes equivalent to

$$\begin{aligned} \underset{u_1}{\text{minimize}} \quad & -u_1 l\beta - |u_1\alpha - d\beta| - \sum_{i=0}^{l-1} |e\alpha - d\beta + ie\beta + u_1\beta| - |s\alpha - u_1(\alpha - \beta) - (d - le)\beta| \\ \text{subject to} \quad & 0 \leq u_1 \leq s. \end{aligned} \tag{2.8}$$

The objective function in (2.8), as a function of u_1, is concave over the interval $[0, s]$. The concavity is due to the convexity of $x \to |x|$. Therefore, the minimum is achieved at one of the extreme values. Equivalently, $u_1^* = s$ or $u_1^* = 0$. □

Lemma 2.2 addresses the case $u_1 + u_2 \leq e$. Generalizing it to the case where $e \leq u_1 + u_2 \leq 2e$ follows the same approach.

Lemma 2.3. *Let* α, β *be non-negative reals,* u_1, u_2, d, e, s, l *be non-negative integers such that*

$u_1 + u_2 = e + s$ *and* $0 \leq u_1, u_2, s \leq e$. *Then, the following inequality holds*

$$f([u_1, \underbrace{e, \ldots, e}_{l \text{ times}}, u_2]) \geq \min(f([s, \underbrace{e, \ldots, e}_{l+1 \text{ times}}]), f([\underbrace{e, \ldots, e}_{l+1 \text{ times}}, s])),$$

where $f(\mathbf{u})$ *is defined as in (2.4).*

Proof. First, we notice that $u_1 = e + s - u_2 \geq s$ as $u_2 \leq e$. Then, the proof follows along similar lines as that of Lemma 2.2 by replacing the constraint in (2.8) by $s \leq u_1 \leq e$. □

In proving the result of Theorem 2.2, we first characterize the optimal solution in the case of $k \leq e$. Insight and intuition gained from this case are used to motivate and derive the general optimal solution. We state the following lemma, which represents a key step towards proving our result.

• **Case** $k \leq e$

In this scenario, the data collector connecting to k nodes from the same repair group yields the worst-case scenario from an information flow perspective. Given a particular repair scenario characterized by a vector $\mathbf{u}$, for any two adjacent repair groups (i.e., two adjacent entries in $\mathbf{u}$) with u_1 and u_2 nodes respectively, we have $u_1 + u_2 \leq e$. One can combine these two groups into a single repair group to achieve a lower cut value. Indeed, from the cut expression in (2.3), the contribution of the initial set $[u_1, u_2]$ to the cut is $\min(u_1\alpha, l\beta) + \min(u_2\alpha, (l-u_1)\beta)$, for some non-negative integer l. After combining the groups into a single repair group, the contribution of the newly formed repair group is $\min((u_1+u_2)\alpha, l\beta)$, which is lower than the initial contribution by virtue of Lemma 2.2, thus achieving a lower cut. This means that starting from an IFG, we construct a new IFG that has one less repair group and lower min-cut value. This process can be repeated until we end up with a single repair

group consisting of $k \leq e$ nodes, which corresponds to the minimum cut over all graphs in this case.

Therefore, the tradeoff in (2.3) is simply characterized by $\mathcal{M} \leq \min(k\alpha, d\beta)$. Moreover, $\alpha_{\text{MSMR}} = \alpha_{\text{MBMR}} = \frac{\mathcal{M}}{k}$ and $\beta_{\text{MSMR}} = \beta_{\text{MBMR}} = \frac{\mathcal{M}}{d}$. Equivalently, the functional storage bandwidth tradeoff reduces to a single point given by $(\alpha_{\text{MSMR}}, \beta_{\text{MSMR}}) = (\alpha_{\text{MBMR}}, \beta_{\text{MBMR}}) = (\frac{\mathcal{M}}{k}, \frac{\mathcal{M}}{d})$.

• **Case** $e < k$

Motivated by the previous case, the intuition is that, according to Lemmas 2.1, 2.2, and 2.3, given a scenario $\mathbf{u}$, one should form a new scenario which exhibits as many groups of size e as possible. Subsequently, one constructs a scenario $\mathbf{u}$ such that all its entries, except maybe one entry, are equal to e.

For a fixed β, we denote the cut corresponding to $\mathbf{u} = [\underbrace{e, \ldots, e}_{j \text{ times}}, r, \underbrace{e, \ldots, e}_{\eta - j \text{ times}}]$, as a function of α, by $C_j(\alpha), j = 0, \ldots, \eta$. As will be shown later in the proof of Theorem 2.2, a careful analysis of the behavior of the $\eta + 1$ different scenarios $C_j(\alpha), 0 \leq j \leq \eta$, is needed to determine the overall optimal scenario. We state the result in the following lemma, whose proof is relegated to Appendix 2.9.1.

Lemma 2.4. *Assume $e \nmid k$. There exists a real number $\alpha_c(\eta) \in [\frac{d}{e}\beta, \frac{d}{r}\beta]$ such that, for any $0 \leq j \leq \eta$,*

$$C_j(\alpha) \begin{cases} \geq C_0(\alpha), & \text{if } \alpha \leq \alpha_c(\eta), \\ \geq C_\eta(\alpha), & \text{if } \alpha \geq \alpha_c(\eta), \end{cases} \tag{2.9}$$

with

$$\alpha_c(\eta) = \frac{d + \eta r - \eta e}{r}\beta. \tag{2.10}$$

Proof of Theorem 2.2. Now that we have the necessary machinery, we proceed as follows: given any scenario $\mathbf{u}$, we keep combining and/or changing repair groups by means of successive applications of Lemma 2.2 and Lemma 2.3 on subsequences of $\mathbf{u}$ until we can no longer reduce the minimum cut. By Lemma 2.1 we reduced the overall minimum cut. The algorithm terminates because at each step, either the number of repair groups in $\mathbf{u}$ is reduced by one, or the number of repair groups of full size e is increased by one. As the number of repair groups is lower bounded by $\eta + 1$, and as the number of repair groups of full size e is upper bounded by η, the algorithm must terminate after a finite number of steps. It can be seen then that the above reduction procedure has a finite number of outcomes, given by

- $\mathbf{u} = [\underbrace{e, \ldots, e}_{\eta \text{ times}}]$ if $k = \eta e$,
- $\mathbf{u} = [\underbrace{e, \ldots, e}_{j\text{times}}, r, \underbrace{e, \ldots, e}_{\eta - j \text{ times}}]$ when $k = \eta e + r$, with $0 < r < e$ and $j \in \{0, \ldots, \eta\}$.

Therefore, if $e \mid k$, then the optimal scenario corresponds to considering exactly η repair groups. On the other hand, if $e \nmid k$, then, it is optimal to consider exactly $\eta+1$ repair groups. However, the optimal position of the repair group with r nodes needs to be determined. Then, using Lemma 2.4, the result in Theorem 2.2 follows. □

Example 2.1. *Let* $\mathbf{u} = [1, 3, 2, 3, 2]$ *with* $e = 3$*. Then, one can start by reducing the first three repair groups* $[1, 3, 2]$*. This leads to* $\mathbf{u} = [3, 3, 3, 2]$*. Another approach would be to consider the last three repair groups* $[2, 3, 2]$*. Reducing this vector leads to either* $\mathbf{u} = [1, 3, 3, 3, 1]$ *or* $\mathbf{u} = [1, 3, 1, 3, 3]$*. Reducing further* $\mathbf{u} = [1, 3, 3, 3, 1]$ *leads to* $\mathbf{u} = [2, 3, 3, 3]$ *or* $\mathbf{u} = [3, 3, 3, 2]$*. Reducing* $\mathbf{u} = [1, 3, 1, 3, 3]$ *leads to* $\mathbf{u} = [3, 2, 3, 3]$ *or* $\mathbf{u} = [2, 3, 3, 3]$*. It remains to compare*

the cuts given by $\mathbf{u} = [3, 3, 3, 2]$, $\mathbf{u} = [3, 3, 2, 3]$, $\mathbf{u} = [3, 2, 3, 3]$ *and* $\mathbf{u} = [2, 3, 3, 3]$. *Following Theorem 2.2, either* $\mathbf{u} = [2, 3, 3, 3]$ *or* $\mathbf{u} = [3, 3, 3, 2]$ *gives the lowest min-cut.*

2.2.5 Explicit expression of the tradeoff

Having characterized the optimal scenario generating the minimum cut in the last section, we are now ready to state the admissible storage-repair bandwidth region for the centralized multi-node repair problem, the proof of which is in Appendix 2.9.2.

Theorem 2.3. *For an* $(\mathcal{M}, n, k, d, e, \alpha, \beta)$ *storage system with total bandwidth* $\gamma = d\beta$, *there exists a threshold function* $\alpha^*(\mathcal{M}, n, k, d, e, \gamma)$ *such that for any* $\alpha \geq \alpha^*(\mathcal{M}, n, k, d, e, \gamma)$, *regenerating codes exist. For any* $\alpha < \alpha^*(\mathcal{M}, n, k, d, e, \gamma)$, *it is impossible to construct codes achieving the target parameters. The threshold function* $\alpha^*(\mathcal{M}, n, k, d, e, \gamma)$ *is defined as follows:*

- *if* $k \leq e$, *then:* $\alpha^* = \frac{\mathcal{M}}{k}$, $\gamma \in [\mathcal{M}, +\infty)$,
- *if* $k = \eta e, \eta \geq 2$, *then:*

$$\alpha^* = \begin{cases} \frac{\mathcal{M}}{k}, & \gamma \in [f_0(\eta - 1), +\infty), \\ \frac{\mathcal{M} - \gamma g_0(i)}{ie}, & \gamma \in [f_0(i-1), f_0(i)], i = \eta - 1, \dots 1, \end{cases}$$

- *if* $k = \eta e + r$ *with* $\eta \geq 1, 1 \leq r \leq e - 1$, *then:*

$$\alpha^* = \begin{cases} \frac{\mathcal{M}}{k}, & \gamma \in [f_r(\eta - 1), +\infty), \\ \frac{\mathcal{M} - \gamma g_r(i)}{r + ie}, & \gamma \in [f_r(i-1), f_r(i)], i = \eta - 1, \dots 1, \\ \frac{\mathcal{M} - \gamma g_r(0)}{r}, & \gamma \in [\frac{d\mathcal{M}}{(\eta+1)d - e\binom{\eta+1}{2}}, f_r(0)], \end{cases} \tag{2.11}$$

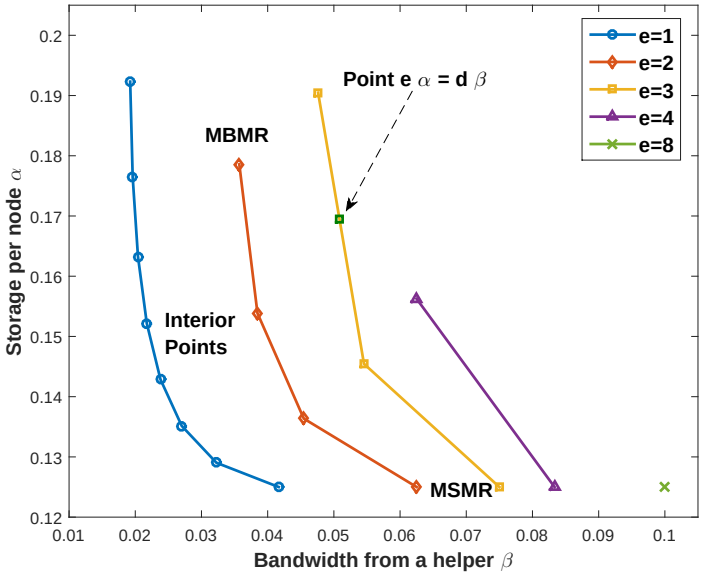


Figure 2.2: Multi-node functional repair tradeoff: $k = 8, d = 10, \mathcal{M} = 1, e \in \{1, 2, 3, 4, 8\}$. When $e \nmid k$, the point with $e\alpha = d\beta$ is not the MBMR point.

where

$$f_r(i) = \frac{2ed\mathcal{M}}{-k^2 - r^2 + e(k-r) + 2kd - e^2(i^2+i) - 2ier}, \qquad (2.12)$$
$$g_r(i) = \frac{(\eta - i)(-2r + e + 2d - \eta e - ei)}{2d}.$$

The functional repair tradeoff is illustrated in Figure 2.2 for multiple values of $e \in \{1, 2, 3, 4, 8\}$ and $k = 8, d = 10$, $\mathcal{M} = 1$.

Remark 2.1. *In the case of $e|k, e|d$, the following equality holds for all points on the tradeoff*

$$\mathcal{M} = \sum_{i=0}^{\eta-1} \min(e\alpha, (d - ie)\beta) \iff \frac{\mathcal{M}}{e} = \sum_{i=0}^{\eta-1} \min(\alpha, (\frac{d}{e} - i)\beta).$$

Therefore, the tradeoff between α and β is the same as the single erasure tradeoff of a system with reduced parameters given by $\frac{\mathcal{M}}{e}$, $\frac{k}{e} = \eta$ and $\frac{d}{e}$. The expression of the tradeoff in this case can be recovered from [14] with the appropriate parameters.

We next present the expressions of the two extreme points on the optimal tradeoff. We focus on the case $e < k$, as otherwise the optimal tradeoff reduces to a single point. The point corresponding to the minimum storage size is refereed to as minimum storage multi-node repair (MSMR) point, while the point corresponding to the minimum repair bandwidth is referred to as minimum bandwidth multi-node repair (MBMR) point.

MSMR. The MSMR point is the same irrespective of the relation between k and e, and it is given by

$$\alpha_{\text{MSMR}} = \frac{\mathcal{M}}{k}, \gamma_{\text{MSMR}} = \frac{\mathcal{M}}{k}\frac{ed}{d-k+e}. \tag{2.13}$$

MBMR. Interestingly, the MBMR point depends on whether e divides k or not.

- If $k = \eta e$, we obtain

$$\gamma_{\text{MBMR}} = \frac{2ed\mathcal{M}}{-k^2+ek+2kd} = \frac{d\mathcal{M}}{d\eta - e\binom{\eta}{2}}, \tag{2.14}$$

$$\alpha_{\text{MBMR}} = \frac{\gamma_{\text{MBMR}}}{e}. \tag{2.15}$$

The amount of information downloaded for repair is equal to the amount of information stored at the e replacement nodes. This property of the MBMR point is similar to the minimum bandwidth point in the single erasure case [14] and also the minimum bandwidth cooperative repair point [55].

- If $k = \eta e + r$, we obtain

$$\gamma_{\text{MBMR}} = \frac{2ed\mathcal{M}}{(k-r+e)(2d-k+r)} = \frac{d\mathcal{M}}{d(\eta+1) - e\binom{\eta+1}{2}}, \tag{2.16}$$

$$\alpha_{\text{MBMR}} = \gamma_{\text{MBMR}}\frac{d+\eta r - e\eta}{rd}. \tag{2.17}$$

This situation is novel for multiple erasures as the e nodes need to store more than the overall

downloaded information. This is an extra cost in order to achieve the low value of the repair bandwidth. Figure 2.2 illustrates this situation with $e = 3, k = 8$. However, later we will see that for both $e|k$ and $e \nmid k$, the total bandwidth at MBMR is equal to the entropy of the failed nodes (see Lemma 2.6 and Lemma 2.11):

$$H(W_E) = d\beta = \gamma,$$

where $E \subset [n]$ is any subset of nodes of size e and W_E is the information stored across the nodes in E.

Remark 2.2. *From the statement of Theorem 2.2, we note that if we only consider points between the MSMR and the MBMR points, then the scenario* $\mathbf{u} = [r, e, \ldots, e]$ *always generates the lowest cut. In fact, the scenario* $\mathbf{u} = [e, \ldots, e, r]$ *corresponds to points beyond the MBMR point, namely, points with* $\alpha \geq \alpha_{MBMR}, \beta = \beta_{MBMR}$.

Case study: centralized strategy v.s. separate strategy. We compare the centralized repair scheme repairing e nodes to a separate strategy repairing each of the e nodes separately using single erasure regenerating codes. We fix k, α and $\mathcal{M}$.

Case I: both strategies use d helpers. The separate strategy requires a total bandwidth given by $ed\beta_1$, while the centralized repair requires $d\beta_e$, where the subscript indicates the number of erasures repaired at a time. For simplicity, we assume that $e \mid k$. The case $e \nmid k$ can be treated in a similar way. For points on the multi-node repair tradeoff, we have

$$\mathcal{M} = \sum_{j=0}^{\eta-1} \min(e\alpha, (d - je)\beta_e).$$

Consider a point with the same $\alpha, k, d, \mathcal{M}$ on the single erasure tradeoff, we write

$$\begin{aligned}\mathcal{M} &= \sum_{j=0}^{\eta-1} \min(e\alpha, (d-je)\beta_e) = \sum_{j=0}^{k-1} \min(\alpha, (d-j)\beta_1) \\ &= \sum_{j=0}^{\eta-1}\sum_{i=0}^{e-1} \min(\alpha, (d-i-je)\beta_1) \\ &\leq \sum_{j=0}^{\eta-1} e\min(\alpha, (d-je)\beta_1) = \sum_{j=0}^{\eta-1} \min(e\alpha, (d-je)e\beta_1).\end{aligned}$$

It follows that $\beta_e \leq e\beta_1$ with equality if and only if $e = 1$. Therefore, for any storage capacity α, multi-node repair requires strictly less bandwidth than a separate strategy for the same number of helpers d.

Case II: multi-node repair uses $d - e + 1$ helpers, and separate repair uses d helpers. In this case, the original number of available nodes that can serve as helpers is assumed to be d, and $e \geq 1$ erasures occur within the available nodes. Then a separate strategy may require a smaller bandwidth for some values of α, as illustrated by Figure 2.3. However, as d is sufficiently large, we observe numerically that multi-node repair with $d - e + 1$ helpers performs better than a separate strategy for all values of α. Moreover, for the MSMR point, the separate repair bandwidth is $ed\beta_{1,\text{MSR}} = e\frac{\mathcal{M}}{k}\frac{d}{d-k+1}$, and centralized repair bandwidth is $(d-e+1)\beta_{e,\text{MSMR}} = \frac{\mathcal{M}}{k}\frac{e(d-e+1)}{d-k+1}$. It follows that a centralized repair is always better that a separate repair strategy, specifically, for $e > 2$,

$$\frac{(d-e+1)\beta_{e,\text{MSMR}}}{ed\beta_{1,\text{MSR}}} = \frac{d-(e-1)}{d} < 1.$$

2.3 Non-existence of exact MBMR regenerating codes

Recall that the MBMR point is defined as the minimum bandwidth point on the functional tradeoff. In this section, we explore the existence of linear exact MBMR regenerating codes

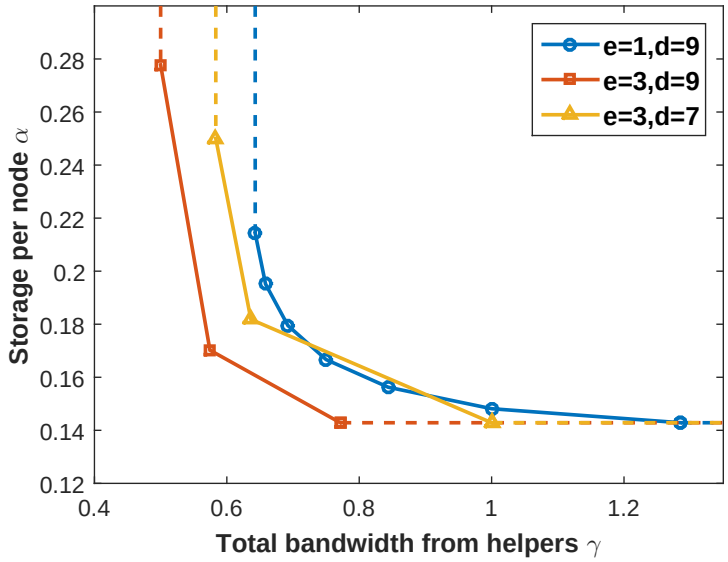


Figure 2.3: Centralized multi-node repair vs separate repair strategy: $k = 7, \mathcal{M} = 1$. Separate repair strategy uses $d = 9$ to repair 3 nodes successively while multi-node repair is plotted for $d = 7$ and $d = 9$.

for $1 < e < k$. Unlike the single erasure repair problem [60] and the cooperative repair problem [58], we prove that linear exact regenerating codes do not exist. Following [58, 60], we proceed by investigating subspace properties that linear exact MBMR codes should satisfy. Then, we prove that the derived properties over-constrain the system.

2.3.1 Subspace viewpoint

Linear exact regenerating codes can be analyzed from a viewpoint based on subspaces. A linear storage code is a code in which every stored symbol is a linear combination of the $\mathcal{M}$ symbols of the file. Let $\mathbf{f}$ denote an $\mathcal{M}$-dimensional vector containing the source symbols. Then, any symbol x can be represented by a vector $\mathbf{h}$ satisfying $x = \mathbf{f}^t\mathbf{h}$ such that $\mathbf{h} \in \mathbb{F}^{\mathcal{M}}$, $\mathbb{F}$ being the underlying finite field. The vectors $\mathbf{h}$ define the code. A node storing α symbols can be considered as storing α vectors. Node i stores $\mathbf{h}_1^{(i)} \dots \mathbf{h}_\alpha^{(i)}$. It is easy to see that linear operations performed on the stored symbols are equivalent to the same operations performed on the these vectors: $\sum \gamma_i \mathbf{f}^t \mathbf{h}_i = \mathbf{f}^t(\sum \gamma_i \mathbf{h}_i)$, $\gamma_i \in \mathbb{F}$. Thus, each node is said to

store a subspace of dimension at most α. We write W_A to denote the subspace stored by all nodes in the set A, $A \subseteq [n]$. For repair, each helper node passes β symbols. Equivalently, each node passes a subspace of dimension at most β. We denote the subspace passed by node j to repair a set R of e nodes by S_j^R. The subspace passed by a set of nodes A to repair a set R of e nodes is denoted by $S_A^R = \sum_{j \in A} S_j^R$, where the sum denotes the sum of subspaces.

Notation. The notation $\bigoplus_j X_j$ denotes the direct sum of subspaces $\{X_j\}$. For a general exact regenerating code, which can be nonlinear, we use by abuse of notation W_A, S_A^R to represent the random variables of the stored information in nodes A, and of the transmitted information from helpers A to failed nodes R. Properties that hold using entropic quantities for a general code do hold when considering linear codes. For instance, consider two sets A and B. Then, we note the following

$$\begin{aligned} H(W_A) &\to \dim(W_A), \\ H(W_A|W_B) &\to \dim(W_A) - \dim(W_A \cap W_B), \\ I(W_A, W_B) &\to \dim(W_A \cap W_B), \end{aligned}$$

where the symbol $\to$ means *translates to*. When results hold for general codes, we only prove for the entropy properties, and the proof for the subspace properties of linear codes is omitted. All results on entropic quantities are for general codes, and all results on subspaces are for linear codes. Moreover, all results in this section refer to properties of *optimal exact* multi-node repair codes with $k > e$, some of which are specific to MBMR codes and will be noted.

In this section, we assume that the codes are *symmetric*. Namely, the entropy (or subspace) properties do not depend on the indices of the nodes. Note that one can always construct a symmetric code from a non-symmetric code [63], hence our assumption does not lose

generality. We now start by proving some properties that exact regenerating codes, satisfying the optimal functional tradeoff, should satisfy. We note that the following property is also presented in [64, Lemma 4].

Lemma 2.5. *Let $B \subseteq [n]$ be a subset of nodes of size e, then for an arbitrary set of nodes A, such that $0 \leq |A| \leq d, B \cap A = \emptyset$,*

$$H(W_B|W_A) \leq H(W_B|S_A^B) \leq \min(e\alpha, (d-|A|)\beta).$$

Proof. If nodes B are erased, consider the case of having nodes A and nodes C as helper nodes, $|C| = d - |A|$. Then, the exact repair condition requires

$$\begin{aligned} 0 = H(W_B|S_A^B, S_C^B) &= H(W_B|S_A^B) - I(W_B, S_C^B|S_A^B) \\ &\geq H(W_B|S_A^B) - H(S_C^B) \\ &\geq H(W_B|S_A^B) - (d-|A|)\beta. \end{aligned}$$

Moreover, we have $H(W_B|S_A^B) \leq H(W_B) \leq |B|\alpha$, $H(W_B|W_A) \leq H(W_B|S_A^B)$, and the results follows. □

In the next two subsections, we focus on the cases where $e \mid k$ and $e \nmid k$, respectively.

2.3.2 Case $e \mid k$

Note that in this case since $e < k$, we have $k \geq 2e$. Recall from Theorem 2.2 that points on the optimal tradeoff satisfy $\mathcal{M} = \sum_{j=0}^{\eta-1} \min(e\alpha, (d-je)\beta)$. Points between and including MSMR and MBMR satisfy $\frac{d-k+e}{e}\beta \leq \alpha \leq \frac{d}{e}\beta$.

Lemma 2.6. *(Entropy of data stored): Consider points on the optimal tradeoff. For an arbitrary set L of storage nodes of size e, and a disjoint set A such that $|A| = em < k$ for*

some integer m,

$$H(W_L) = e\alpha, H(W_L|W_A) = \min(e\alpha, (d-em)\beta).$$

For linear codes,

$$\dim(W_L) = e\alpha, \dim(W_L) - \dim(W_L \cap W_A) = \min(e\alpha, (d-em)\beta).$$

Hence, the contents of any group of e nodes are independent. In particular, for a set A of nodes, $1 \leq |A| \leq e$, $H(W_A) = |A|\alpha$.

Proof. By reconstruction requirement, we write

$$\begin{aligned}\mathcal{M} = H(W_{[k]}) &= H(W_{[e]}) + \sum_{j=1}^{\eta-1} H(W_{\{ej+1,\ldots,e(j+1)\}}|W_{[je]}) \\ &\leq \min(e\alpha, d\beta) + \sum_{j=1}^{\eta-1} \min(e\alpha, (d-ej)\beta) = \mathcal{M},\end{aligned}$$

where the inequality follows from Lemma 2.5. Thus, all inequalities must be satisfied with equality. □

Corollary 2.1. *At the MBMR point, for any set L of size e and disjoint set A of size $|A| = em < k$, we have*

$$\dim(W_L \cap W_A) = em\beta.$$

Proof. By Lemma 2.6 and $e\alpha = d\beta$,

$$\dim(W_L) - \dim(W_L \cap W_A) = \min(e\alpha, (d-em)\beta) = (d-em)\beta.$$

Using the fact that $\dim(W_L) = e\alpha = d\beta$, we obtain the result. □

Lemma 2.7. *For any set E of size e, and a disjoint set A of size d, the MBMR point satisfies*

$$W_E = \bigoplus_{j \in A} S_j^E, \dim(S_j^E) = \beta.$$

Hence, the subspaces S_j^E and $S_{j'}^E$ are linearly independent. For every set $Q \subseteq A$, $\dim(S_Q^E) = |Q|\beta$. Moreover, each subspace S_j^E has to be in W_E, namely, $S_j^E \subseteq W_E$.

Proof. For exact repair, we need $W_E \subseteq \sum_j S_j^E$. Thus,

$$d\beta = e\alpha = \dim(W_E) \leq \dim(\sum_j S_j^E) \leq d\beta.$$

Thus, every inequality has to be satisfied with equality. □

Lemma 2.8. *At the MBMR point, for any set E of e nodes and any other disjoint set Q of size $|Q| \leq k - e$, we have*

$$S_Q^E = W_E \cap W_Q, \dim(W_E \cap W_Q) = \dim(S_Q^E) = |Q|\beta.$$

Proof. Consider Q nodes such that $|Q| \leq k - e$ helping in the repair of a set E of e nodes. Let J contains Q such that $|J| = k - e$. Denote $Q^c = J \backslash Q$. From Corollary 2.1, we have $\dim(W_E \cap W_J) = (k-e)\beta$. On the other hand, from Lemma 2.7, we have $\dim(S_J^E) = (k-e)\beta$ and $S_J^E \subseteq W_E$. Moreover, by definition, $S_J^E \subseteq W_J$. Thus, $S_J^E \subseteq W_E \cap W_J$. As the dimensions match, it follows that $S_J^E = W_E \cap W_J$. Note that $S_A^E \subseteq W_E \cap W_A$ holds for any subset A of size $|A| \leq d$. Now, we write

$$S_J^E = W_E \cap W_J = W_E \cap (W_Q + W_{Q^c}) \supseteq W_E \cap W_Q + W_E \cap W_{Q^c} \supseteq S_Q^E + S_{Q^c}^E = S_J^E.$$

This implies that all inclusion inequalities have to be satisfied with equality and the result

follows. □

The next lemma plays an important role in establishing the non-existence of exact MBMR codes. It only holds true when $e \geq 2$, which conforms with the existence of single erasure MBMR codes.

Lemma 2.9. *Consider the MBMR point. When $e \geq 2$, for any set of $e+2 \leq k$ nodes, labeled 1 through $e+2$, it holds that*

$$\dim(W_{e+2} \cap W_{[e+1]}) = \dim(W_{e+2} \cap W_{[e]}) = \beta.$$

Proof. We have

$$\begin{aligned}\dim(W_{[e+2]}) &= \dim(W_{[e]}) + \dim(W_{e+1} + W_{e+2}) - \dim(W_{[e]} \cap (W_{e+1} + W_{e+2})) \\ &= e\alpha + 2\alpha - 2\beta,\end{aligned}$$

where the second equality follows from Lemma 2.6, Lemma 2.8. On the other hand, we write

$$\begin{aligned}\dim(W_{[e+2]}) &= \dim(W_{[e]}) + \dim(W_{e+1}) - \dim(W_{e+1} \cap W_{[e]}) \\ &\quad + \dim(W_{e+2}) - \dim(W_{e+2} \cap W_{[e+1]}) \\ &= e\alpha + 2\alpha - \beta - \dim(W_{e+2} \cap W_{[e+1]}).\end{aligned}$$

The lemma follows from equating both equations. □

Theorem 2.4. *Exact linear regenerating MBMR codes do not exist when $2 \leq e < k$ and $e \mid k$.*

Proof. Assuming that there exists an exact-repair regenerating code, we consider the first e nodes. Then, these nodes store linearly independent vectors. We write, for $i = 1, \ldots, e$, $W_i =$

$\begin{bmatrix} V_{i1} & V_{i2} \end{bmatrix}$ where $V_{i,1}$ contains β linearly independent columns and $V_{i,2}$ contains the remaining $(\alpha - \beta)$ basis vectors for node i. Now, consider node $e+1$. We have $\dim(W_{e+1} \cap W_{[e]}) = \beta$ by Lemma 2.9. That means that node $e+1$ contains β columns, linearly dependent on the columns from the first e nodes. Since the first e nodes should be linearly independent, w.l.o.g, we can assume that the β dependent vectors of node $e+1$, denoted by $V_{e+1,1}$, is of the form

$$V_{e+1,1} = \sum_{i=1}^{e} V_{i,1}\mathbf{x}_i, \tag{2.18}$$

such that $\mathbf{x}_i \neq \mathbf{0}_{\beta\times 1}\ \forall i = 1, \ldots, e$. Now, consider node $e+2$. From Lemma 2.9, node $e+2$ contains $(\alpha - \beta)$ vectors linearly independent from vectors in nodes 1 through $e+1$. The remaining basis vectors of node $e+2$ (which are linearly independent of the $(\alpha-\beta)$ vectors) are denoted by $V_{e+2,1}$. Now, to repair any set of e nodes out of the set of first $e+1$ nodes, node $e+2$ can only pass $V_{e+2,1}$. Otherwise, Lemma 2.8 will be violated. Then, this implies that $V_{e+2,1} \subseteq W_J$, for all $J \subseteq \{1, \ldots, e+1\}$ such that $|J| = e$. Then, it can be seen that $V_{e+2,1}$ can only be of the same form in (2.18)

$$V_{e+2,1} = \sum_{i=1}^{e} V_{i,1}\mathbf{y}_i, \text{ such that } \mathbf{y}_i \neq \mathbf{0}_{\beta\times 1}\ \forall i = 1, \ldots, e.$$

Similar reasoning applies to node i for $i = e+3, \ldots, k+1$ to conclude that $V_{i,1}$ can be written as in (2.18).

Now, assume the first e nodes fail. Then, node i can only pass $V_{i,1}$ for $i = e+1, \ldots k+1$. We recall from Lemma 2.8 that $S_i^{[e]} = W_i \cap W_{[e]}$. The total number of vectors passed by these nodes is $(k-e+1)\beta \geq (e+1)\beta$. On the other hand, from (2.18), all $V_{i,1}$ are generated by $e\beta$ vectors. Thus, the set $\{V_{i,1}, i = e+1, \ldots, k+1\}$ must be linearly dependent, which contradicts the linear independence property of the passed subspaces passed for repair, as stated by Lemma 2.7. $\square$

2.3.3 Case $e \nmid k$

Recall that from the analysis of Theorem 2.2, for $k = \eta e + r$, $1 \leq r \leq e-1$, at the MBMR point, two scenarios generate the same minimum cut: $\mathbf{u}_1 = [r, e, \ldots, e]$ and $\mathbf{u}_2 = [e, \ldots, e, r]$. Equivalently, we have $\mathcal{M} = f(\mathbf{u}_1) = f(\mathbf{u}_2)$, where $f()$ is defined as in (2.4). Moreover, all points between and including MSMR and MBMR on the tradeoff satisfy

$$\begin{aligned} &\mathcal{M} = \min(r\alpha, d\beta) + \sum_{i=0}^{\eta-1} \min(e\alpha, (d - r - ie)\beta) = f(\mathbf{u}_1), \\ &\frac{d-k+e}{e}\beta \leq \alpha \leq \frac{d+\eta r-\eta e}{r}\beta. \end{aligned} \tag{2.19}$$

Properties satisfied by exact regenerating codes developed in the previous section extend to the case $e \nmid k$ with slight modifications. We state the properties without detailed proofs as the techniques are the same.

Lemma 2.10. *Consider points on the optimal tradeoff. For an arbitrary set R of storage nodes of size r, and a set A such that $|A| = je + r < k$ for some integer $j \leq \eta - 1$, for all exact-regenerating codes operating on the functional tradeoff, it holds that*

$$H(W_R) = r\alpha, H(W_E|W_A) = \min(e\alpha, (d - r - je)\beta).$$

For linear codes,

$$\dim(W_R) = r\alpha, \dim(W_E) - \dim(W_E \cap W_A) = \min(e\alpha, (d - r - je)\beta).$$

Proof. The result can be derived by proceeding as in Lemma 2.6 and using the fact that $\mathcal{M} = f(\mathbf{u}_1)$ from (2.19). □

Remark 2.3. *In the case of $e \nmid k$, a set of e nodes are no longer linearly independent. This is expected as $e\alpha > d\beta$. Instead, it can be seen from Lemma 2.10 that any set of r nodes are*

linearly independent.

Lemma 2.11. *For exact-regenerating codes operating at the MBMR point, given sets E, A, R and B such that $|E| = e$, E and A are disjoint, R and B are disjoint, $|A| = je$ with $j \leq \eta - 1$, $|R| = r$ and $|B| = \eta e$, it holds that*

$$H(W_E) = d\beta,$$
$$H(W_E|W_A) = (d - je)\beta,$$
$$H(W_R|W_B) = (d - \eta e)\beta.$$

For linear codes,

$$\dim(W_E) = d\beta,$$
$$\dim(W_E) - \dim(W_E \cap W_A) = (d - je)\beta,$$
$$\dim(W_R) - \dim(W_R \cap W_B) = (d - \eta e)\beta.$$

Proof. The result can be derived by proceeding as in Lemma 2.6 and using the fact that $\mathcal{M} = f(\mathbf{u}_2)$ and $e\alpha \geq d\beta, r\alpha \geq (d - ae)\beta$. □

It is easy to see that Lemma 2.7 and Lemma 2.8 hold true for the case $e \nmid k$, and for conciseness we do not repeat these lemmas. The following lemma is used to derive the contradiction in our non-achievability result.

Lemma 2.12. *Let $k = \eta e + r$, then exact linear MBMR point is not achievable when $d > k$. When $d = k$, for any set of $r + 1$ nodes, it holds that*

$$\dim(W_{r+1} \cap W_{[r]}) = \beta.$$

Proof. We have

$$d\beta = \dim(W_{[e]}) = \sum_{i=1}^{e} \big(\dim(W_i) - \dim(W_i \cap W_{[i-1]})\big) = e\alpha - \sum_{i=r+1}^{e} \dim(W_i \cap W_{[i-1]}),$$

where the last equality follows from the fact that the first r nodes are linearly independent. Thus, it follows that

$$\sum_{i=r+1}^{e} \dim(W_i \cap W_{[i-1]}) = e\alpha - d\beta = (e-r)(\alpha - \eta\beta), \tag{2.20}$$

where we used $\alpha = (d + r\eta - e\eta)\beta/r$ for MBMR point. Now we write

$$\begin{aligned}(e-r)(\alpha - \eta\beta) = \sum_{i=r+1}^{e} \dim(W_i \cap W_{[i-1]}) &\geq \sum_{i=r+1}^{e} \dim(W_i \cap W_{[r]}) \\ &= (e-r)\dim(W_{r+1} \cap W_{[r]}),\end{aligned}$$

where the last equality follows using symmetry. Then, it follows that

$$\dim(W_{r+1} \cap W_{[r]}) \leq \alpha - \eta\beta. \tag{2.21}$$

Combining (2.20) and (2.21), we obtain

$$\sum_{i=r+2}^{e} \dim(W_i \cap W_{[i-1]}) \geq (e-r-1)(\alpha - \eta\beta). \tag{2.22}$$

On the other hand, we have

$$\sum_{i=r+2}^{e} \dim(W_i \cap W_{[i-1]}) \leq \sum_{i=r+2}^{e} \dim(W_i \cap W_{\mathcal{E}_i}) = (e-r-1)\beta, \tag{2.23}$$

where $\mathcal{E}_i$ is a set of e nodes containing the first $i-1$ nodes and arbitrary $e-i+1$ nodes, excluding node i, and the equality follows from Lemma 2.8. Combining (2.22) and (2.23), it

follows

$$(e-r-1)(\alpha-\eta\beta) \leq \sum_{i=r+2}^{e} \dim(W_i \cap W_{[i-1]}) \leq (e-r-1)\beta. \tag{2.24}$$

It follows that $\alpha - \eta\beta = \frac{d-\eta e}{r}\beta \leq \beta$. The inequality holds only when $d = k$ and $\alpha - \eta\beta = \beta$. Indeed, when $d > k$, we have $\alpha - \eta\beta > \beta$. Therefore, we only consider the case $d = k$. Hence, it follows from (2.24) that

$$\sum_{i=r+2}^{e} \dim(W_i \cap W_{[i-1]}) = (e-r-1)\beta.$$

Using (2.20), we obtain $\dim(W_{r+1} \cap W_{[r]}) = \beta$. □

Theorem 2.5. *Exact linear regenerating MBMR codes do not exist when $e < k$ and $e \nmid k$.*

Proof. From Lemma 2.12, exact linear MBMR codes may only be feasible when $d = k$. Next we show that in fact such codes do not exist. Consider repair of the set of nodes E containing nodes 1 through e. Consider helper node i. As $\dim(W_i \cap W_{[r]}) = \dim(W_i \cap W_{[e]}) = \beta$, it follows that $W_i \cap W_{[r]} = W_i \cap W_{[e]} = S_i^E$. Then, each helper node sends vectors in the span of $W_{[r]}$. Thus, the span of all sub-spaces $S_i^{[e]}$ is included in the span of $W_{[r]}$: $\sum_i S_i^E \subseteq W_{[r]}$. This implies that $\dim(\sum_i S_i^E) \leq \dim(W_{[r]})$. Namely, we should have $d\beta \leq r\alpha$: this is a contradiction as $d\beta > r\alpha$. □

2.4 Infeasibility of the exact-repair interior points

In this section, we study the infeasibility of the interior points on the optimal *functional-repair* tradeoff for $e \mid k, e \mid d, 2e < k$, similarly to [50]. We note that all interior points satisfy $(d-k+e)\beta \leq e\alpha \leq d\beta$. This can be written as $(d' - \eta + 1)\beta \leq \alpha \leq d'\beta$, where $d' = \frac{d}{e}$ and $\eta = \frac{k}{e}$. This is similar to the single erasure case with reduced parameters. The

proof techniques in this section follow along similar lines as [50] and some of the proofs are relegated to the appendix.

Parameterization of the interior points. Let $\alpha = (d'-p)\beta-\theta$, namely $e\alpha = (d-ep)\beta-e\theta$ with $p \in \{0, 1, \ldots, \eta-1\}$, $\theta \in [0, \beta)$ such that $\theta = 0$ if $p = \eta - 1$. Points on the functional tradeoff satisfy

$$\mathcal{M} = e\sum_{i=0}^{\eta-1} \min(\alpha, (d'-i)\beta).$$

2.4.1 Properties of exact-repair codes

We present a set of properties that exact-repair codes, satisfying the optimal functional tradeoff, must satisfy.

Lemma 2.13. *For a set A of arbitrary nodes of size ej, a set L of nodes of size e such that $L \cap A = \emptyset$, we have*

$$I(W_L, W_A) = \begin{cases} 0, & j \leq p, \\ e((j-p)\beta - \theta), & p < j < \eta, \\ e\alpha, & j \geq \eta. \end{cases}$$

Proof. See Appendix 2.9.3. □

Corollary 2.2. *For an arbitrary set L of size e, and a disjoint set A such that $|A| = em < k$ for some integer m, we have*

$$H(W_L|S_A^L) = H(W_L|W_A) = \min(e\alpha, (d-em)\beta).$$

Proof. From Lemma 2.5, we have $H(W_L|S_A^L) \leq \min(e\alpha, (d-em)\beta)$. On the other hand,

from Lemma 2.6,

$$H(W_L|S_A^L) \geq H(W_L|W_A) = \min(e\alpha, (d-em)\beta).$$

Thus, $H(W_L|S_A^L) = H(W_L|W_A) = \min(e\alpha, (d-em)\beta)$. □

Lemma 2.14. *In the situation where node m is an arbitrary helper node assisting in the repair of a second set of arbitrary nodes L of size e, we have*

$$H(S_m^L) = \beta,$$

irrespective of the identity of the other $d-1$ helper nodes. Moreover, for set B of size $|B| \leq d-k+e$ with $B \cap L = \emptyset$, we have

$$H(S_B^L) = |B|\beta.$$

Proof. See Appendix 2.9.4. □

Helper node pooling. Consider a set F consisting of a collection of $f \leq d+e$ nodes (f is a multiple of e), and a subset R of the set F consisting of er' nodes, $r' \geq 1$. A helper node pooling scenario is a scenario where upon failure of any e nodes $L \subseteq R$, the d helper nodes include all the $f-e$ remaining nodes in F. The remaining $d-f+e$ helper nodes are fixed given L. Consider a subset of nodes $M \subseteq F \backslash R$. Partition the nodes in R into arbitrary but fixed sets $R_1, R_2, \ldots, R_{r'}$, each of size e. Denote by $S_M^R = (S_M^{R_1}, \ldots S_M^{R_{r'}})$ the collective transmitted information from helper nodes M to repair $R_1, \ldots, R_{r'}$, respectively.

Lemma 2.15. *In the helper node pooling scenario where* $\min(\eta, \frac{f}{e}) > p+2 \geq r'$*, for any set of e arbitrary nodes $M \subseteq F \backslash R$, we have*

$$H(S_M^R) \leq e(2\beta - \theta).$$

Proof. See Appendix 2.9.5. □

Lemma 2.16. *In the helper node scenario where* $\min\{\eta, \frac{f}{e}\} > p + 1 \geq r'$*, for an arbitrary set of* e *nodes* $M \subseteq F \backslash R$*, and an arbitrary pair of set of* e *nodes* L_1, L_2*, it must be that*

$$H(S_M^{L_1} | S_M^{L_2}) \leq e\theta,$$

and hence

$$H(S_M^R) \leq e(\beta + (r' - 1)\theta).$$

Proof. See Appendix 2.9.6. □

2.4.2 Non-existence proof

For interior points, $1 \leq p \leq \eta - 2$. First, we consider the interior points for which $e\alpha$ is a multiple of β. That is: $e\alpha = (d - ep)\beta, \theta = 0$.

Theorem 2.6. *Exact-repair codes do not exist for the interior points with* $\theta = 0$.

Proof. Consider a sub-network F consisting of $d + e$ nodes. The parameters satisfy the condition in Lemma 2.16. Note that by the regeneration property for any set of e nodes $L \subseteq F$, $H(W_L | S_{F-L}^L) = 0$. Moreover, for distinct $M, L_1, L_2 \subseteq F$, with $\theta = 0$, we have $H(S_M^{L_1} | S_M^{L_2}) = 0$. We partition the nodes in F into groups of size e, denoted $L_i, i = 1, 2, \ldots, d' + 1$. Then,

we write

$$\begin{aligned}\mathcal{M} \leq H(W_F) \leq H(S_{F-L_1}^{L_1}, \ldots, S_{F-L_{d'+1}}^{L_{d'+1}}) &= H(S_{L_1}^{F-L_1}, \ldots, S_{L_{d'+1}}^{F-L_{d'+1}}) \\ &\leq \sum_{i=1}^{d'+1} H(S_{L_i}^{F-L_i}) \\ &\leq \sum_{i=1}^{d'+1} e\beta = (d+e)\beta, \end{aligned} \tag{2.25}$$

where the inequality (2.25) follows from Lemma 2.16. On the other hand,

$$\begin{aligned}\mathcal{M} = \sum_{i=0}^{d'-1} \min(e\alpha, (d-ie)\beta) &= \sum_{i=0}^{d'-1} \min((d-ep)\beta, (d-ie)\beta) \\ &= 2(d-ep)\beta + \sum_{i=2}^{d'-1} \min((d-ep)\beta, (d-ie)\beta) \\ &\geq 2(d-ep)\beta + (\eta-2)e\beta \\ &\geq 2e\beta + (d-ep)\beta + (\eta-2)e\beta \\ &= (d-2e)\beta + (k-2e-ep)\beta \\ &\geq (d-2e)\beta, \end{aligned}$$

where we assume $1 \leq p \leq \eta - 2$ (non-MSMR point). Thus, $ep + 2e \leq k \leq d$. Both bounds are contradictory, thus proving the impossibility result in the case of $\theta = 0$. □

Theorem 2.7. *For any given values of $\mathcal{M}$, exact-repair regenerating codes do not exist for the parameters lying in the interior of the storage-bandwidth tradeoff when $\theta \neq 0$, except possibly for the case $p + 2 = \eta$ and $\theta \geq \frac{d-ep-e}{d-ep}\beta$.*

Proof. See Appendix 2.9.7. □

2.5 Outer bounds for linear exact-repair codes

In this section, we derive an outer bound on the storage-bandwidth tradeoff of linear exact-repair regenerating codes. The performance is evaluated under various parameter settings. We derive the bound by constructing a rank lower bound on the parity check matrices of linear centralized repair codes. This approach is inspired by [65], which fist applied it to the single erasure regeneration codes, and [59] which applied it to the case of cooperative regenerating codes. The following theorem describes the proposed outer bound.

Theorem 2.8. *Consider a linear exact repair regenerating code with parameters* $(\mathcal{M}, n, k, d, e, \alpha, \beta)$*, the file size* $\mathcal{M}$ *satisfies*

$$\mathcal{M} \leq \frac{s-1}{s+1}(d+e)\alpha + \frac{2}{s(s+1)}\sum_{h=1}^{g}\min(sl_h\alpha, hl_h\alpha, (d-k+\sum_{t=1}^{h}l_t)\beta), \tag{2.26}$$

where $\mathbf{l} = [l_1, \ldots, l_g] \in \mathcal{P}$*, given by* (2.5)*, and* s *is an arbitrary integer with* $1 \leq s \leq g$*.*

Notation. For a non-negative integer n, define $[n] \triangleq \{1, \ldots, n\}$. We use the concept of thick columns and thick rows. Let M be a matrix consisting of mn submatrices as

$$M = \begin{bmatrix} M_{11} & \cdots & M_{1n} \\ \vdots & \ddots & \vdots \\ M_{m1} & \cdots & M_{mn} \end{bmatrix}.$$

$S(M)$ denotes the column space of M. For sets $I \subseteq [m], J \subseteq [n]$, let M_{IJ} be the matrix constructed by collecting submatrices whose indices i and j belong to I and J, respectively. For $i \in [m], j \in [n]$, let $M_{i,[n]} = [M_{i1} \ldots M_{in}]$ be the i-th thick row and $M_{[m]j} = \begin{bmatrix} M_{ij}^t \ldots M_{mj}^t \end{bmatrix}^t$ be the j-th thick column.

2.6 Properties of linear exact repair codes

In this section, we define linear exact repair regenerating codes, and establish some of their properties.

An $(\mathcal{M}, n, k, d, e, \alpha, \beta)$ centralized repair regenerating code encodes a $(1 \times \mathcal{M})$ message vector $\mathbf{m}$ into a $(1 \times n\alpha)$ codeword vector $\mathbf{c}$. The first α symbols of $\mathbf{c}$ are stored in the first node, the next α symbols in the second node, and so on. Let $\mathbf{c}_i$ denotes the vector of symbols stored at node i, such that

$$\mathbf{c} = \Big[\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n\Big].$$

A *linear* exact repair regenerating code is an exact repair regenerating code where the encoding and decoding processes, performed during data collection and multi-node repair, are linear. Thus, a linear exact repair code can be regarded as an $(n\alpha, \mathcal{M})$-linear code, such that there exists an $(\mathcal{M} \times n\alpha)$ generator matrix G and an $((n\alpha - \mathcal{M}) \times n\alpha)$ parity check matrix H which satisfy

$$\mathbf{c} = \mathbf{m}G, GH^t = \mathbf{0},$$
$$\text{rank}(G) = \mathcal{M}, \text{rank}(H) = n\alpha - \mathcal{M}.$$

In the following, we restrict $n = d + e$, as any (n, k, d, e) regenerating code contains a $(d + e, k, d, e)$ regenerating code. We now derive a key lemma that will be used in the proof of Theorem 2.8, establishing conditions that must be satisfied by matrices G and H.

Lemma 2.17. *Consider an $(\mathcal{M}, n, k, d, e, \alpha, \beta)$-linear centralized exact repair regenerating code with $n = d + r$. The parity check matrix H satisfies the following two conditions.*

1. *The rank of a matrix constructed by collecting $n - k$ arbitrary thick columns is full*

$(n-k)\alpha$.

2. *For any index set* $R = \{i_1, \ldots, i_e\} \subset [n]$, *where* $i_1 < \ldots < i_e$ *and* $|R| = e$, *there exists an* $(e\alpha \times n\alpha)$ *matrix*

$$H_R = \begin{bmatrix} A^R_{i_1,1} & \cdots & A^R_{i_1,n} \\ \vdots & \ddots & \vdots \\ A^R_{i_e,1} & \cdots & A^R_{i_r,n} \end{bmatrix},$$

where $A^R_{i,j}$ *is of size* $(\alpha \times \alpha)$, *such that*

(a) $A^R_{RR} = I_{r\alpha}$,

(b) for $j \in D \triangleq [n]\backslash R$, *we have rank*$(A^R_{R,j}) \leq \beta$,

(c) $S(H^t_R) \subseteq S(H^t)$.

Proof. The proof of (1) is similar to [59, Proof of Lemma 1] and is omitted. It follows from the data reconstruction property. We now prove (2).

Consider a repair process where the e nodes in R are centrally repaired with the help of the d nodes in D. Node $j \in D$ sends a $(1 \times \beta)$ vector $\mathbf{s}_{Rj}$, which is a linear combination of elements of $\mathbf{c}_j$. That is, there exists an $(\alpha \times \beta)$ matrix Φ_{Rj} such that

$$\mathbf{s}_{Rj} = \mathbf{c}_j \Phi_{Rj}, j \in D.$$

According to the repair property, $\mathbf{c}_i$ can be reconstructed by linearly combining $d\beta$ symbols of $\{\mathbf{s}_{Rj}, j \in D\}$ as

$$\mathbf{c}_i = \sum_{j \in D} \mathbf{s}_{Rj} L^R_{i,j} = \sum_{j \in D} \mathbf{c}_j \Phi_{Rj} L^R_{i,j} \triangleq \sum_{j \in D} \mathbf{c}_j A^R_{i,j}, i \in R,$$

where $L_{i,j}^R$ are encoding matrices of size $(\beta \times \alpha)$. For $i \in R$, define

$$A_{i,j}^R = \begin{cases} I_\alpha, & \text{if } j = i, \\ -(\Phi_{Rj} L_{i,j}^R)^t, & \text{if } j \in D, \\ 0_{\alpha\times\alpha}, & \text{otherwise.} \end{cases}$$

Then, it follows that

$$\forall i \in R, \begin{bmatrix} A_{i,1}^R & A_{i,2}^R & \cdots & A_{i,n}^R \end{bmatrix} \mathbf{c}^t = 0 \implies H_R \mathbf{c}^t = 0.$$

Moreover, for $j \in D$,

$$\begin{aligned} (H_R)_{R,j}^t &= \begin{bmatrix} (A_{i_1,j}^R)^t & \cdots & (A_{i_R,j}^R)^t \end{bmatrix} \\ &= \begin{bmatrix} -\Phi_{Rj} L_{i_1,j}^R & \cdots & -\Phi_{Rj} L_{i_R,j}^R \end{bmatrix} \\ &= \Phi_{Rj} \begin{bmatrix} -L_{i_1,j}^R & \cdots & -L_{i_R,j}^R \end{bmatrix}. \end{aligned}$$

Thus,

$$\begin{aligned} \text{rank}((H_R)_{R,j}) &= \text{rank}(\Phi_{Rj} \begin{bmatrix} -L_{i_1,j}^R & \cdots & -L_{i_R,j}^R \end{bmatrix}) \\ &\leq \text{rank}(\Phi_{Rj}) \leq \beta. \end{aligned}$$

Finally, since $H_R \mathbf{c}^t = 0$ for every codeword $\mathbf{c}$, $S(H_R^t)$ is orthogonal to $S(G^t)$ and hence $S(H_R^t) \subseteq S(H^t)$. □

2.7 Outer bound for linear exact repair codes

In this section, we prove Theorem 2.8 via the following steps:

1. Choose an arbitrary vector $\mathbf{l} \in \mathcal{P}$, defined in (2.5).

2. Construct an $(n\alpha \times n\alpha)$ repair matrix, $H_{repair,\mathbf{l}}$, obtained using Lemma 2.17.

3. Find a lower bound of $\text{rank}(H_{repair,\mathbf{l}})$. This will also be a lower bound of $\text{rank}(H)$ as $S(H_R^t) \subseteq S(H^t)$.

4. The lower bound on $\text{rank}(H)$ is translated to an upper bound on $\mathcal{M}$, using the relation $\mathcal{M} = n\alpha - \text{rank}(H)$.

2.7.1 Construction of $H_{repair,\mathbf{l}}$

Consider a vector $\mathbf{l} \in \mathcal{P}$, and let $l_0 = n - k$. Then, $\sum_{h=0}^{g} l_i = n = d + e$. Consider the sets

$$R_h = R_h^{'} \cup N_h, R_h^{'} = [\sum_{t=0}^{h-1} l_t + 1, \sum_{t=0}^{h} l_t],$$

where $|R_h^{'}| = l_h$, and N_h is chosen arbitrarily to satisfy $N_h \subset [\sum_{t=0}^{h-1} l_t], |N_h| = r - l_h$. For each set R_h, which is of size e, by Lemma 2.17, there exists a matrix H_{R_h} of size $(r\alpha \times n\alpha)$. From this matrix, we collect the last l_h thick rows, in a matrix $A^{R_h}_{R_h^{'},[n]}$, of size $(l_h\alpha \times n\alpha)$.

Let $\tilde{H}$ be constructed by taking the first $(n-k)$ thick columns of H and $\tilde{H}^{\dagger}$ its left-inverse. $H_{repair,\mathbf{l}}$ is constructed as follows:

$$H^t_{repair,\mathbf{l}} = \left[(\tilde{H}^{\dagger}H)^t \quad (A^{R_1}_{R_1^{'},[n]})^t \quad \cdots \quad (A^{R_1}_{R_g^{'},[n]})^t \right].$$

The thick rows of $H_{repair,\mathbf{l}}$ have $l_0\alpha, l_1\alpha, \ldots, l_h\alpha$ rows, respectively. By grouping the columns

by the same pattern, $H_{repair,\mathbf{l}}$ consists of $(g+1)^2$ submatrices as

$$H_{repair,\mathbf{l}} = \begin{bmatrix} H_{0,0} & \cdots & H_{0,g} \\ \vdots & \ddots & \vdots \\ H_{g,0} & \cdots & H_{g,g} \end{bmatrix},$$

where $H_{h,t}$ is of size $(l_h\alpha \times l_t\alpha)$ and

$$\begin{bmatrix} H_{0,0} & H_{0,1} & \cdots & H_{0,g} \end{bmatrix} = \tilde{H}^{\dagger} H,$$
$$H_{h,t} = A^{R_h}_{R'_h,R'_t}, \text{ for } 1 \le h \le g, 0 \le t \le g.$$

Moreover, $H_{h,h}$ is the h-th diagonal submatrix with the size of $(l_h\alpha \times l_h\alpha)$. It can readily verified that $H_{0,0} = \tilde{H}^{\dagger}\tilde{H} = I_{n-k}\alpha$, and from Condition(2)-(a) in Lemma 2.17, it follows that

$$H_{h,h} = I_{l_h\alpha}, \text{for } 1 \le h \le g.$$

2.7.2 An alternative proof of the functional repair bound

Using the above machinery, we first derive a bound for the linear exact repair codes, that coincides with the functional repair cut-set bound. Define $\delta_0 = \text{rank}(H_{[0,g],0})$ and

$$\delta_h = \text{rank}(H_{[0,g],[0,h]}) - \text{rank}(H_{[0,g],[0,h-1]}), \text{ for } 1 \le h \le g.$$

Then, δ_h represents the increment of rank after the h-th thick column is added. Note that $\delta_0 = (n-k)\alpha$. For $1 \le h \le g$, looking at the h-th thick row, we have

$$\begin{aligned}\delta_h &\ge \mathrm{rank}(H_{h,[0,h]}) - \mathrm{rank}(H_{h,[0,h-1]}) \\ &\ge \mathrm{rank}(H_{h,h}) - \mathrm{rank}(H_{h,[0,h-1]}) \\ &= l_h\alpha - \mathrm{rank}(A^{R_h}_{R'_h,[\sum_{t=0}^{h-1} l_t]}).\end{aligned}$$

But $\mathrm{rank}(A^{R_h}_{R'_h,[\sum_{t=0}^{h-1} l_t]}) = \mathrm{rank}(A^{R_h}_{R'_h,L_h})$, where L_h is the set of thick columns that are non-zero. Thus, $L_h = [\sum_{t=0}^{h-1} l_t]\backslash N_h$. By Condition (2)-(b) in Lemma 2.17, we have

$$\mathrm{rank}(A^{R_h}_{R'_h,j}) \le \mathrm{rank}(A^{R_h}_{R_h,j}) \le \beta, \forall j \in [n]\backslash R_h.$$

Note that $L_h \subset [n]\backslash R_h$, and $|L_h| = \sum_{t=0}^{h-1} l_t - N_h = \sum_{t=0}^{h} l_t - r = d - k + \sum_{t=1}^{h} l_t$. Therefore, we have

$$\mathrm{rank}(A^{R_h}_{R'_h,L_h}) \le \sum_{j\in L_h} \mathrm{rank}(A^{R_h}_{R'_h,j}) \le \beta|L_h| = \beta(d-k+\sum_{t=1}^{h} l_t).$$

Thus, we have

$$\begin{aligned}\mathrm{rank}(H_{repair,\mathbf{l}}) &= \sum_{h=0}^{g} \delta_h = (n-k)\alpha + \sum_{h=1}^{g} \delta_h \\ &\ge (n-k)\alpha + \sum_{h=1}^{g} [l_h\alpha - (d-k+\sum_{t=1}^{h} l_t)]^+ \\ &= (n-k)\alpha + \sum_{h=1}^{g} [l_h\alpha - (d-\sum_{t=h+1}^{g} l_t)\beta]^+,\end{aligned}$$

as $\sum_{t=1}^{g} l_t = k$, where $[x]^+ \triangleq \max(0, x)$. Therefore, we have

$$\begin{aligned}
\mathcal{M} &= n\alpha - rank(H) \\
&\leq n\alpha - rank(H_{repair,\mathbf{l}}) \\
&= \sum_{h=1}^{g} l_h\alpha - \sum_{h=1}^{g}[l_h\alpha - (d - \sum_{t=h+1}^{g} l_t)\beta]^+ \\
&= \sum_{h=1}^{g} \min(l_h\alpha, (d - \sum_{t=h+1}^{g} l_t)\beta) \\
&= \sum_{h=1}^{g} \min(l_h'\alpha, (d - \sum_{t=1}^{h-1} l_t')\beta),
\end{aligned}$$

where $(l_1', l_2', \ldots, l_g') = (l_g, l_{g-1}, \ldots, l_1)$, recovering (2.3).

2.7.3 Derivation of Theorem 2.8

In this subsection, we obtain a different lower bound on $\text{rank}(H_{repair,\mathbf{l}})$ using the following theorem.

Theorem 2.9 ([59]). *Suppose a matrix M has n thick columns and n thick rows (i.e., M has n^2 submatrices). The number of columns (rows) in each thick column (thick row) does not have to be identical. $M_1, \ldots, M_n$ denote the n thick columns and $M_{i,j}$ $(i, j \in [n])$ denote the submatrices of M. If M satisfies the following conditions*

1. *For any $i \in [n]$, the thick column M_i has linearly independent columns,*
2. *$rank(M_{i,i}) = rank(M_i)$ for $i \in [n]$,*

then, for every integer $s \geq 1$, rank(M) is lower bounded by

$$\frac{s(s+1)}{2} rank(M) \geq$$
$$\sum_{i=1}^{n} \max(0, (s-i+1) rank(M_{i,i}), s\ rank(M_{i,i}) - T_i), \tag{2.27}$$

where

$$T_i = \begin{cases} 0, & \text{if } i = 1, \\ \sum_{j=1}^{i-1}\ rank(M_{i,j}), & \text{if } 2 \leq i \leq n. \end{cases}$$

Proof of Theorem 2.8: For a given vector $\mathbf{l}$, $H_{repair,\mathbf{l}}$ has $(g+1)^2$ submatrices. Moreover, the diagonal submatrices are identity matrices. Thus, $H_{repair,\mathbf{l}}$ satisfies conditions (1) and (2) of Theorem 2.9. Consider an integer $s \geq 1$. Using (2.27), we obtain

$$\frac{s(s+1)}{2} \text{rank}(H_{repair,\mathbf{l}}) \geq s(n-k)\alpha + \sum_{h=1}^{g} \max(0, (s-h)l_h\alpha, sl_h\alpha - T_h),$$

where $T_h = \sum_{t=0}^{h-1} \text{rank}(H_{h,t}) = \sum_{t=0}^{h-1} \text{rank}(A^{R_h}_{R'_h,R'_t})$. Note that

$$T_h \leq \sum_{j=0}^{\sum_{t=0}^{h-1} l_t} \text{rank}(A^{R_h}_{R'_h,j}) = \sum_{j \in L_h} \text{rank}(A^{R_h}_{R'_h,j}) \leq |L_h|\beta = (d-k+\sum_{t=1}^{h} l_t)\beta \triangleq \Delta_h.$$

Thus, we write

$$\begin{aligned} \text{rank}(H_{repair,\mathbf{l}}) &\geq \frac{2(n-k)\alpha}{s+1} + \frac{2}{s(s+1)} \sum_{h=1}^{g} \max(0, (s-h)l_h\alpha, sl_h\alpha - \Delta_h) \\ &\geq \frac{2(n-k)\alpha}{s+1} + \frac{2}{s(s+1)} \sum_{h=1}^{g} sl_h\alpha + \frac{2}{s(s+1)} \sum_{h=1}^{g} \max(-sl_h\alpha, -hl_h\alpha, -\Delta_h), \\ &= \frac{2n\alpha}{s+1} - \frac{2}{s(s+1)} \sum_{h=1}^{g} \min(sl_h\alpha, hl_h\alpha, \Delta_h). \end{aligned}$$

In terms of $\mathcal{M}$, we have $\mathcal{M} \leq n\alpha - rank(H_{repair,\mathbf{1}})$, which simplifies to (2.26) in Theorem 2.8. While (2.26) holds for $s \geq 1$, following [59, Remark 6], it is sufficient to consider $s \in [g]$.

Remark 2.4. *When $s = 1$, (2.26) coincides with the bound in (2.3). Theorem 2.8 is at least as tight as the functional repair bound*

2.8 Evaluation of the bound in Theorem 2.8 and discussion

We evaluate Theorem 2.8 under various parameter settings in Fig. 2.4. The bound improves upon the functional repair bound when d is close to k, and the improvement increases as k increases. In particular, for a $(k + e, k, k, e)$ system, we observe that the gap between the region of the exact repair bound and the best known achievable region from [3] decreases as k increases, for fixed e. We note however that Theorem 2.8 does not rule out the achievability of the minimum bandwidth point of the functional tradeoff, which is known from Theorem 2.4 and Theorem 2.5 to be not achievable under linear exact repair. As a result, an open problem is to tighten the bound of Theorem 2.8.

2.9 Conclusion

In this chapter, we studied the problem of centralized repair of multiple erasures in distributed storage systems. We explicitly characterized the optimal functional tradeoff between the repair bandwidth and the storage size per node. For instance, we obtained the expressions of the extreme points on the tradeoff, namely the minimum storage multi-node repair (MSMR) and the minimum bandwidth multi-node repair (MBMR) points. Furthermore, we proved that the functional MBMR point is not achievable for linear exact repair codes for $1 < e < k$. We also showed that the functional repair tradeoff is not achievable under exact

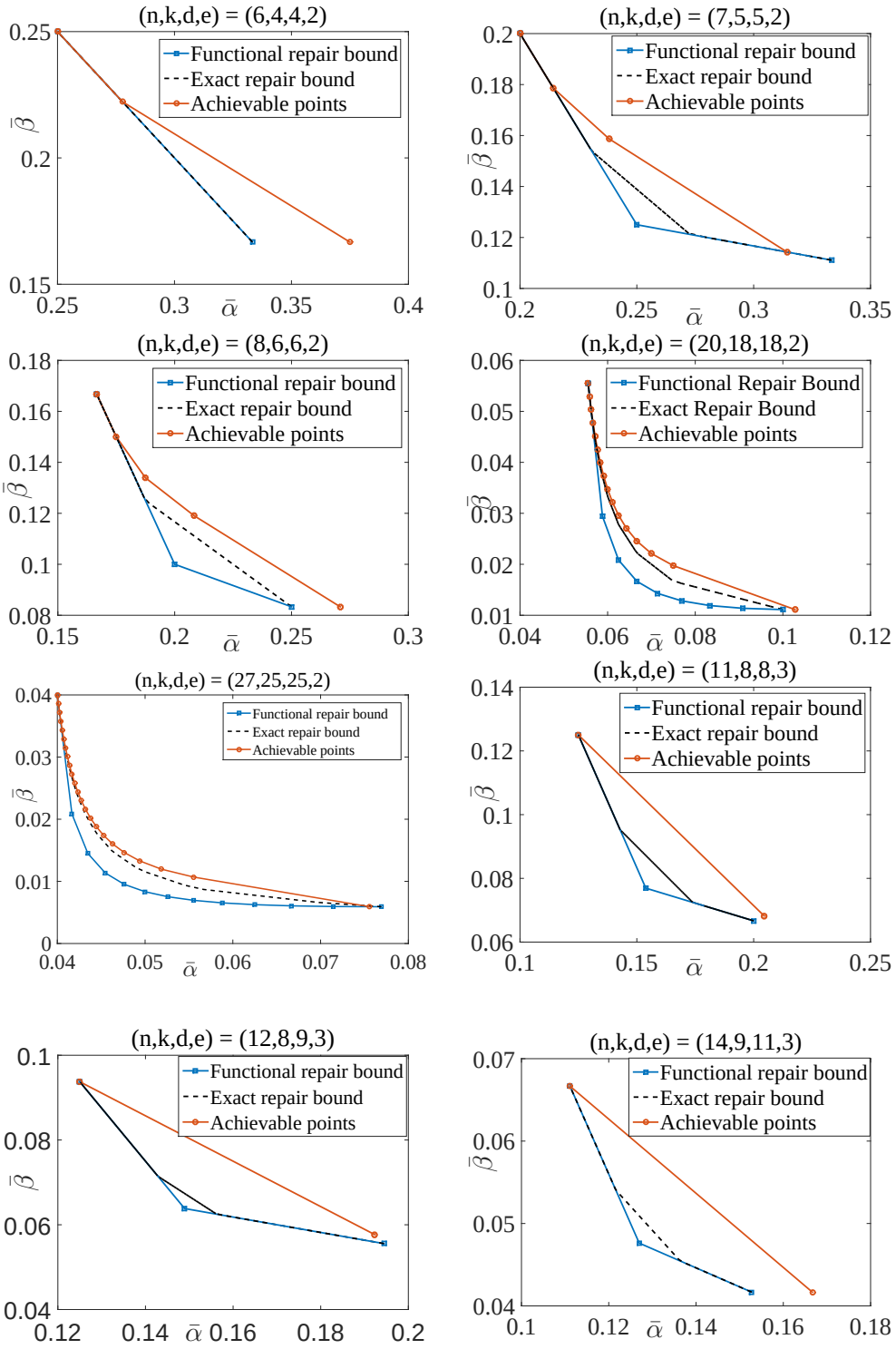


Figure 2.4: Exact repair bound of Theorem 2.8 vs. functional repair bound vs. achievable points from [3].

repair, except for maybe a small portion near the MSMR point for $e < k, e \mid k, e \mid d$. Finally, we presented outer bounds for linear exact repair codes.

Appendix

2.9.1 Proof of Lemma 2.4

We first state the following lemma which will be useful in the proof.

Lemma 2.18. *For fixed β, the scenario $\mathbf{u} = [e, \ldots, e, r]$ achieves the lowest final value of minimum cut:*

$$\lim_{\alpha \to +\infty} f(\mathbf{u}) \geq \lim_{\alpha \to +\infty} f([e, \ldots, e, r]), \forall \mathbf{u} \in \mathcal{P},$$

where $f(\mathbf{u})$ and $\mathcal{P}$ are defined in (2.4) and (2.5), respectively.

Proof. for a specific cut $\mathbf{u}$, we have

$$\begin{aligned}\lim_{\alpha \to +\infty} f(\mathbf{u}) = \sum_{i=1}^{g} (d - \sum_{j=1}^{i-1} u_j)\beta = d\beta g - \beta \sum_{i=1}^{g} \sum_{j=1}^{i-1} u_j = gd\beta - \beta \sum_{i=1}^{g-1} u_i (g - i) \\ = \beta(dg - g \sum_{i=1}^{g-1} u_i + \sum_{i=1}^{g-1} i u_i) = \beta((d-k)g + \sum_{i=1}^{g} i u_i).\end{aligned}$$

To obtain the smallest minimum cut value, we need to solve the following problem

$$\begin{aligned} \underset{\mathbf{u},g}{\text{minimize}} \quad & (d-k)g + \sum_{i=1}^{g} i u_i \\ \text{subject to} \quad & 1 \leq u_i \leq e, \\ & \sum_{i=1}^{g} u_i = k. \end{aligned} \tag{2.28}$$

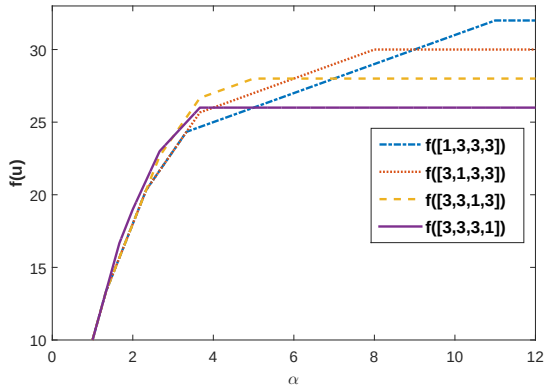


Figure 2.5: Values of the cut function for different vectors $\mathbf{u}$ for $k = 9, d = 10, \beta = 1$

It can be seen that the solution to (2.28) is given by $\mathbf{u} = [e, \ldots, e, r]$. □

We now study the different functions $C_j(\alpha)$ for $j = 0, \ldots, \eta$. An example of the different functions to be analyzed is given in Figure 2.5, with $k = 9, d = 10, \beta = 1$. It is observed that $\mathbf{u} = [1, 3, 3, 3]$ generates the lowest cut before some threshold $\alpha^* = 5$, after which the lowest cut is generated by $\mathbf{u} = [3, 3, 3, 1]$. In the following, by analyzing $C_j(\alpha)$ for $j = 0, \ldots, \eta$, we prove that the above observation holds true in general.

Case j=0: We have

$$\begin{aligned} C_0(\alpha) &= \min(r\alpha, d\beta) + \sum_{i=0}^{\eta-1} \min(e\alpha, (d - r - ie)\beta) \\ &= r \min(\alpha, \frac{d\beta}{r}) + \sum_{i=0}^{\eta-1} e \min(\alpha, \frac{(d - r - ie)\beta}{e}). \end{aligned}$$

$C_0(\alpha)$ is a piecewise linear function with breakpoints given by $\{\frac{d-r-(\eta-1)e}{e}\beta, \frac{d-r-(\eta-2)e}{e}\beta, \ldots, \frac{d-r}{e}\beta, \frac{d}{r}\beta\}$. C_0 increases from 0 at a slope of k. Its slope is then reduced by e by the successive breakpoints and then finally by r until it levels off.

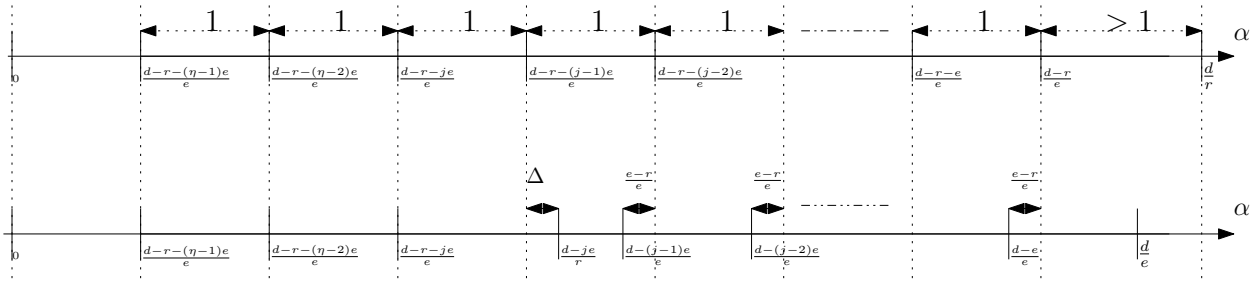


Figure 2.6: relative positions of the breakpoints of $C_0(\alpha)$ and $C_j(\alpha)$ (with $\beta = 1$)

.

Case $1 \leq j \leq \eta$**:** For each j, we have

$$\begin{aligned} C_j(\alpha) &= \sum_{i=0}^{j-1} \min(e\alpha, (d-ie)\beta) + \min(r\alpha, (d-je)\beta) + \sum_{i=j}^{\eta-1} \min(e\alpha, (d-r-ie)\beta) \\ &= \sum_{i=0}^{j-1} e \min(\alpha, \frac{(d-ie)\beta}{e}) + r\min(\alpha, \frac{(d-je)\beta}{r}) + \sum_{i=j}^{\eta-1} e\min(\alpha, \frac{(d-r-ie)\beta}{e}). \end{aligned}$$

$C_j(\alpha)$ is also piecewise-linear function with non-increasing successive slopes. Its breakpoints are given by $\{\frac{d-r-(\eta-1)e}{e}\beta, \ldots, \frac{d-r-je}{e}\beta, \frac{d-(j-1)e}{e}\beta, \ldots, \frac{d}{e}\beta\} \cup \{\frac{d-je}{r}\beta\}$. The exact relative position of the breakpoint $\frac{d-je}{r}\beta$ with respect to the other breakpoints of $C_j(\alpha)$ depends on the system's parameters. However, we give a lower bound on $\frac{d-je}{r}\beta$.

$$\begin{aligned} \frac{d-je}{r} - \frac{d-r-(j-1)e}{e} &= \frac{ed-rd-re+r^2-j(e^2-re)}{re} \\ &\geq \frac{(e-r)d-re+r^2-\eta(e^2-re)}{re} \\ &\geq \frac{(e-r)k-re+r^2-\eta(e^2-re)}{re} \\ &= 0, \end{aligned}$$

where the first inequality follows by noticing that the expression is decreasing in j and letting $j = \eta$, and the second inequality follows as the corresponding expression is increasing d.

Figure 2.6 illustrates the relative positions of all the breakpoints of $C_0(\alpha)$ and $C_j(\alpha), j \geq 1$, where for example $\frac{d-je}{r} \in [\frac{d-r-(j-1)e}{e}, \frac{d-r-(j-2)e}{e}]$. We denote by $C_j(\infty) = \lim_{\alpha \to +\infty} C_j(\alpha)$.

Lemma 2.19. *For $1 \leq j \leq \eta$, there exists a point $\alpha_c(j) \in [\frac{d}{e}, \frac{d}{r}]$ such that*

$$\begin{aligned} &C_0(\alpha_c(j)) = C_j(\alpha_c(j)), \\ &\quad C_0(\alpha) \leq C_j(\alpha) \qquad \textit{if } \alpha \leq \alpha_c(j), \\ &\quad C_0(\alpha) \geq C_j(\alpha) \qquad \textit{if } \alpha \geq \alpha_c(j), \\ &\quad C_j(\alpha) = C_j(\infty) \qquad \textit{if } \alpha \geq \alpha_c(j). \end{aligned} \tag{2.29}$$

Proof. W.l.o.g, assume $\beta = 1$. First, we note that

$$C_0(\alpha) = C_j(\alpha) = k\alpha \text{ for } \alpha \leq \frac{d - r - (j-1)e}{e}.$$

Next, we analyze the behavior of each of the functions $C_0(\alpha)$ and $C_j(\alpha)$ over the successive intervals $I_i \triangleq (\frac{d-r-ie}{e}, \frac{d-r-(i-1)e}{e}]$ for $i \in \{j-1, j-2, \ldots, 1\}$. Let $x_i = \frac{d-r-ie}{e}$ and define $s_j(I_i)$ as the slope of $C_j(\alpha)$ just before $\alpha = x_i$. Consider a given interval $I_i = (x_i, x_{i-1}]$, we have

- $C_0(\alpha)$ has no breakpoint inside I_i. Thus, $C_0(\alpha)$ increases by

$$C_0(x_{i-1}) - C_0(x_i) = s_0(I_i) - e.$$

- $C_j(\alpha)$ has either one or two breakpoints inside I_i.

  Case 1: $C_j(\alpha)$ has a single breakpoint inside I_i (at $\alpha = \frac{d-ie}{e}$), then, $C_j(\alpha)$ increases by

$$C_j(x_{i-1}) - C_j(x_i) = s_j(I_i)\frac{r}{e} + (s_j(I_i) - e)\frac{e-r}{e} = s_j(I_i) - e + r.$$

  Case 2: $C_j(\alpha)$ has two breakpoints inside I_i, namely at $\alpha = \frac{d-je}{r}$ and $\alpha = \frac{d-ie}{e}$. Let

$\Delta = \frac{d-je}{r} - \frac{d-r-ie}{e}$ (c.f. Figure 2.6). Assuming $\frac{d-je}{r} \leq \frac{d-ie}{e}$, then, $C_j(\alpha)$ increases by

$$\begin{aligned} C_j(x_{i-1}) - C_j(x_i) &= (s_j(I_i) - r)(1 - \Delta - \frac{e-r}{e}) + \frac{e-r}{e}(s_j(I_i) - r - e) + s_j(I_i)\Delta \\ &= s_j(I_i) - e + \Delta r. \end{aligned}$$

Assuming $\frac{d-je}{r} \geq \frac{d-ie}{e}$, then, $C_j(\alpha)$ increases by

$$\begin{aligned} C_j(x_{i-1}) - C_j(x_i) &= \frac{r}{e}s_j(I_i) + (k-e)(\Delta - \frac{r}{e}) + (s_j(I_i) - r - e)(1 - \Delta) \\ &= s_j(I_i) - e + \Delta r, \end{aligned}$$

which shows that the increase does not depend on the relative position of the two breakpoints.

Now that we have computed the increase increment of each C_j over I_i, we proceed to compare $C_0(\alpha)$ and $C_j(\alpha)$ for $1 \leq j \leq \eta$. We discuss two cases:

Case 1: Assume $\frac{d-je}{r} \in I_{j_0}$ for some $j_0 \in [1, j-1]$. j_0 may not exist, which will be discussed in the second case. Based on the above discussion, it can be seen that

$$C_j(\alpha) \geq C_0(\alpha), \text{ for } \alpha \leq x_{j_0}.$$

This can be seen by noticing that $\forall i < j_0, s_0(I_i) = s_j(I_i)$ and that

$$(C_j(x_{i-1}) - C_j(x_i)) - (C_0(x_{i-1}) - C_0(x_i)) = r \geq 0.$$

Over I_{j_0}, C_j also dominates C_0 at every point as $s_0(I_{j_0}) = s_j(I_{j_0})$ and

$$(C_j(x_{i-1}) - C_j(x_i)) - (C_0(x_{i-1}) - C_0(x_i)) = \Delta r \geq 0.$$

For $i > j_0$, we have $s_0(I_i) - s_j(I_i) = r$. Moreover, over each $I_i, i > j_0$, we have

$$(C_j(x_{i-1}) - C_j(x_i)) - (C_0(x_{i-1}) - C_0(x_i)) = (s_j(I_i) - e + r) - (s_0(I_i) - e) = 0.$$

Combining the last equation and the observation that $C_j(x_{j_0-1}) \geq C_j(x_{j_0-1})$, it follows that C_j continues to dominate C_0 over the successive intervals $I_i, i > j_0$. So far, we have shown that $C_j(\alpha) \geq C_0(\alpha),$ for $\alpha \leq \frac{d-r}{e}$.

For $\alpha \geq \frac{d-r}{e}$, we observe that C_j increases with a slope of e and levels off at $\frac{d}{e}$ while C_0 increases at smaller slope given by r and levels off at $\frac{d}{r} > \frac{d}{e}$. Moreover, we know from Lemma 2.18 that C_0 levels off at a higher value than that of C_j. Thus, there exists $\alpha_c(j) \in [\frac{d}{e}, \frac{d}{r}]$ that satisfies (2.29).

Case 2: Assume $\frac{d-r}{e} < \frac{d-je}{r} \leq \frac{d}{r}$, then, using similar arguments as in the first case, it follows that for $\alpha \leq \frac{d-r}{e}$, $C_j(\frac{d-r}{e}) \geq C_0(\frac{d-r}{e})$. At $\alpha = \frac{d-r}{e}$, $C_j(\alpha)$ has a slope of $r + e$, which is higher than that of C_0, given by r. Thus, the slope of C_j remains higher than than of C_0 until C_j levels off. Combining these observations with the fact that C_0 levels off at a higher value, it follows that both curves will intersect only once. Moreover, the intersection is at a point at which C_j has leveled off i.e., we have $\alpha_c(j) \geq \max(\frac{d}{e}, \frac{d-je}{r})$. Therefore, (2.29) holds also in this case. □

Using Lemma 2.19 and the fact that C_η achieves the smallest final value from Lemma 2.18, that is $C_\eta(\infty) \leq C_j(\infty)$, $j \in [0, \eta-1]$, it follows that (2.9) holds for any $j \in [0, \eta]$. Moreover, as $\alpha_c(\eta) \in [\frac{d}{e}, \frac{d}{r}]$, $\alpha_c(\eta)$ satisfies

$$r\alpha_c(\eta) + \sum_{i=0}^{\eta-1}(d - r - ie)\beta = (\eta + 1)\beta d - e\beta\frac{\eta^2 + \eta}{2},$$

which implies that

$$r\alpha_c(\eta) + \eta(d - r - \frac{e\eta}{2} + \frac{e}{2})\beta = (\eta+1)\beta d - e\beta\frac{\eta^2+\eta}{2}.$$

Simplifying the last equation yields (2.10).

2.9.2 Storage-bandwidth tradeoff expression

We start with the case $k = \eta e + r$. The optimization trade-off is

$$\begin{aligned} &\underset{\alpha\geq 0}{\text{minimize}} \quad \alpha \\ &\text{subject to} \quad C(\alpha) \geq \mathcal{M}. \end{aligned} \tag{2.30}$$

The constraint is a piece-wise linear function $C(\alpha)$ given by

$$C(\alpha) = \begin{cases} (\eta+1)\beta d - e\beta\eta(\eta+1)/2, \alpha \geq \alpha_c, \\ r\alpha + \sum_{j=0}^{\eta-1} b_j, \alpha \in [\frac{b_0}{e}, \alpha_c], \\ (r+ie)\alpha + \sum_{j=i}^{\eta-1} b_j, \alpha \in [\frac{b_i}{e}, \frac{b_{i-1}}{e}], i = 1, \ldots, \eta-1, \\ k\alpha, \alpha \leq \frac{b_{\eta-1}}{e}, \end{cases} \tag{2.31}$$

with $\alpha_c = \frac{d+\eta r - \eta e}{r}\beta, b_i = (d - r - ie)\beta$ and

$$\sum_{j=i}^{\eta-1} b_j = \beta(\eta - i)(d - r - \frac{e(\eta - 1 + i)}{2}) = \gamma\frac{(\eta-i)(-2r+e+2d-\eta e-ei)}{2d} \triangleq \gamma g_r(i), \tag{2.32}$$

such that

$$g_r(i) = \frac{(\eta - i)(-2r + e + 2d - \eta e - ei)}{2d}.$$

The expression $C(\alpha)$ increases from 0 to a maximum value given by $\beta((\eta+1)d - \binom{\eta+1}{2})$. To solve (2.30), we let $\alpha^* = C^{-1}(\mathcal{M})$ under the condition $\mathcal{M} \leq \beta((\eta+1)d - \binom{\eta+1}{2})$. Therefore, we obtain,

$$\alpha^* = \begin{cases} \frac{\mathcal{M}}{k}, & \mathcal{M} \in [0, \frac{kb_{\eta-1}}{e}] \\ \frac{\mathcal{M} - \sum_{j=i}^{\eta-1} b_j}{r+ie}, \mathcal{M} \in [(r+ie)\frac{b_i}{e} + \sum_{j=i}^{\eta-1} b_j, (r+ie)\frac{b_{i-1}}{e} + \sum_{j=i}^{\eta-1} b_j], & \\ \qquad\qquad \text{for } i = \eta - 1, \dots 1, & \\ \frac{\mathcal{M} - \sum_{j=0}^{\eta-1} b_j}{r}, \mathcal{M} \in [\frac{b_0 r}{e} + \sum_{j=0}^{\eta-1} b_j, r\alpha_c + \sum_{j=0}^{\eta-1} b_j], & \end{cases}$$

with

$$\begin{aligned} \frac{rb_i}{e} + ib_i + \sum_{j=i}^{\eta-1} b_j &= \frac{-\eta^2 e^2 + \eta e^2 - 2aer + 2dae - e^2 i^2 - e^2 i - 2eir - 2r^2 + 2dr)}{2de}\gamma \\ &= \frac{-k^2 - r^2 + e(k-r) + 2kd - e^2(i^2+i) - 2ier}{2ed}\gamma \\ &\triangleq \gamma \frac{\mathcal{M}}{f(i)}, \end{aligned}$$

such that

$$f_r(i) = \frac{2ed\mathcal{M}}{-k^2 - r^2 + e(k-r) + 2kd - e^2(i^2+i) - 2ier}.$$

Therefore, fixing $\mathcal{M}$ and varying γ, we write

$$\alpha^* = \begin{cases} \frac{\mathcal{M}}{k}, & \mathcal{M} \in [0, \frac{kg_r(\eta-1)\gamma}{e}], \\ \frac{\mathcal{M}-\gamma g_r(i)}{r+ie}, & \mathcal{M} \in [\frac{\gamma\mathcal{M}}{f_r(i)}, \frac{\gamma\mathcal{M}}{f_r(i-1)}], i = \eta-1, \dots 1, \\ \frac{\mathcal{M}-\gamma g_r(0)}{r}, & \mathcal{M} \in [\frac{\gamma\mathcal{M}}{f_r(0)}, (g_r(0) + \frac{d+ar-ae}{d})\gamma]. \end{cases} \tag{2.33}$$

As a function of γ, after simplifications, we obtain the expression of α^* as in Theorem 2.3. We note that there are η piece-wise linear portions on the curve. Moreover, the minimum bandwidth point γ_{MBMR} is given by

$$\gamma_{\text{MBMR}} = \frac{\mathcal{M}}{g_r(0) + \frac{d+ar-ae}{d}} = \frac{d\mathcal{M}}{d(\eta+1) - e\binom{\eta+1}{2}}. \tag{2.34}$$

The expression of α_{MBMR} is given by

$$\alpha_{\text{MBMR}} = \frac{\mathcal{M} - \gamma_{\text{MBMR}} g(0)}{r} = \gamma_{\text{MBMR}} \frac{d + \eta r - e\eta}{rd}.$$

In the case of $e \mid k$, we have $r = 0$. The expression of the tradeoff is obtained from (2.33) by setting $r = 0$ and eliminating the last line. We note that in this case, there are $\eta - 1$ piece-wise linear portions on the trade-off curve.

2.9.3 Proof of Lemma 2.13

Proof. First, we note that when $j \geq \eta$, $I(W_L, W_A) = H(W_L) - H(W_L|A) = H(W_L) = e\alpha$. In the following, we assume $j < \eta$. We write

$$\begin{aligned} I(W_L, W_A) &= H(W_L) - H(W_L|A) \\ &= e\alpha - \min(e\alpha, (d - je)\beta) \qquad (2.35) \\ &= e(\alpha - \min(\alpha, (d' - j)\beta)) \\ &= e(\alpha - (d' - j)\beta)^+ \\ &= e((j - p)\beta - \theta)^+, \end{aligned}$$

where we use the notation $(x)^+ \triangleq \max(x, 0)$. Here (2.35) follows from Lemma 2.6. □

2.9.4 Proof of Lemma 2.14

Proof. Partition the set of d helpers into A and B such that $|A| = k - e$ and $|B| = d - k + e$, such that $m \in B$. We have $H(W_L|S_A^L) = \min(e\alpha, (d - k + e)\beta) = (d - k + e)\beta$, as $e\alpha \geq (d - k + e)\beta$ for all points on the tradeoff. Moreover, exact repair requires $H(W_L|S_A^L, S_B^L) = 0$. Thus, $H(S_B^L) \geq (d - k + e)\beta$. This implies $H(S_B^L) = (d - k + e)\beta$. Moreover, it must hold that $H(S_m^L) = \beta$ in addition to S_m^L and $S_{m'}^L$ being independent if $m \neq m'$. Moreover, by choosing $M \subseteq B$, one obtains $H(S_M^L) = e\beta$. □

2.9.5 Proof of Lemma 2.15

Proof. If the statement holds true for some f, r', then it also holds true for all $f' \geq f$ and $r'' \leq r'$. Thus, for the proof, we only need to consider $F = R \cup M, |F| = f = e(p + 3), |R| = r'e = (p + 2)e, |M| = e$.

Consider repair of an arbitrary set of e nodes $L \subseteq R$, where the set of helpers include M and the $e(p+1)$ remaining nodes in R. Then, we write

$$\begin{aligned} I(S_M^L; W_R) &= I(S_M^L; W_L, W_{R-L}) \\ &= I(S_M^L; W_{R-L}) + I(S_M^L; W_L | W_{R-L}) \\ &\geq I(S_M^L; W_L | W_{R-L}) \\ &= H(W_L | W_{R-L}) - H(W_L | W_{R-L}, S_M^L) \\ &\geq H(W_L | W_{R-L}) - H(W_L | S_{R-L}^L, S_M^L) \\ &= \min(e\alpha, (d - e(p+1))\beta) - \min(e\alpha, (d - e(p+2))\beta) \qquad (2.36) \\ &= (d - e(p+1))\beta - (d - e(p+2))\beta = e\beta. \end{aligned}$$

Here (2.36) follows from Lemma 2.6 and Corollary 2.2. Then, we obtain

$$H(S_M^L | W_R) = H(S_M^L) - I(S_M^L; W_R) \leq e\beta - e\beta = 0.$$

Hence, $H(S_M^L | W_R) = 0$. Since L is arbitrary, it follows that $H(S_M^R | W_R) = 0$. It follows from Lemma 2.13 that

$$H(S_M^R) = I(S_M^R; W_R) \leq I(W_M; W_R) = e(2\beta - \theta).$$

Hence the proof is completed. □

2.9.6 Proof of Lemma 2.16

Proof. The set is R assumed to consist of $|R| = er' = e(p+1)$ nodes, and the set F is such that $F = R \cup \{M\}$, $|M| = e$. Similar to Lemma 2.15,

$$\begin{aligned} I(S_M^L; W_R) &\geq H(W_L|W_{R-L}) - H(W_L|S_{R-L}^L, S_M^L) \\ &= \min(e\alpha, (d-(r'-1)e)\beta) - \min(e\alpha, (d-r'e)\beta) \\ &= (d-pe)\beta - e\theta - (d-(p+1)e)\beta \\ &= e(\beta-\theta). \end{aligned}$$

Then, it must be that

$$H(S_M^L|W_R) = H(S_M^L) - I(S_M^L; W_R) \leq e\beta - e(\beta-\theta) = e\theta. \tag{2.37}$$

Note that the last inequality holds for any set $L \subseteq R$. Next, consider $L_1, L_2 \subseteq R$. For this, consider

$$\begin{aligned} H(S_M^{L_1}, S_M^{L_2}) &= I(W_R; S_M^{L_1}, S_M^{L_2}) + H(S_M^{L_1}, S_M^{L_2}|W_R) \\ &\leq I(W_R; W_M) + H(S_M^{L_1}, S_M^{L_2}|W_R) \\ &= I(W_R; W_M) + H(S_M^{L_1}|W_R) + H(S_M^{L_2}|W_R, S_M^{L_1}) \\ &\leq e(\beta-\theta) + e\theta + e\theta = e(\beta+\theta), \end{aligned} \tag{2.38}$$

where the last inequality follows from Lemma 2.13 and (2.37). Then, we have

$$H(S_M^{L_1}|S_M^{L_2}) = H(S_M^{L_1}, S_M^{L_2}) - H(S_M^{L_2}) = H(S_M^{L_1}, S_M^{L_2}) - e\beta \leq e(\beta+\theta) - e\beta = e\theta,$$

where the first equality follows from Lemma 2.14. Finally, partitioning the nodes in R into sets $R_1, R_2, \ldots, R_{r'}$ of size e, it follows

$$H(S_M^R) \leq H(S_M^{R_1}) + \sum_{i=2}^{r'} H(S_M^{R_i}|S_M^{R_{i-1}}) \leq e\beta + e(r'-1)\theta.$$

Thus the proof is completed. □

2.9.7 Proof of Theorem 2.7

Proof. Take a subnetwork F of $d+e$ nodes. Let $L, M \subseteq F$ be two disjoint groups of e nodes. Partition the $d-e$ remaining nodes into two sets, A of cardinality ep and B of cardinality $d-ep-e$. Exact repair requires

$$H(W_L|S_A^L, S_B^L, S_M^L) = 0, H(W_M|S_A^M, S_B^M, S_L^M) = 0.$$

It follows that

$$\begin{aligned} H(W_L, W_M|W_A, S_B^L, S_B^M, S_M^L) &= H(W_L|W_A, S_B^L, S_B^M, S_M^L) + H(W_M|W_L, W_A, S_B^L, S_B^M, S_M^L) \\ &= 0. \end{aligned}$$

Therefore, we have

$$\begin{aligned} H(S_B^L, S_B^M, S_M^L) &\geq H(W_L, W_M|W_A) \\ &= H(W_L|W_A) + H(W_M|W_A, W_L) \\ &= H(W_L) - I(W_L; W_A) + H(W_M) - I(W_M; W_A, W_L) \\ &= e\alpha - 0 + e\alpha - e(\beta - \theta) \qquad (2.39) \\ &= 2e\alpha - e\beta + e\theta \\ &= 2((d - ep)\beta - e\theta) - e\beta + e\theta \\ &= (2d - 2ep - e)\beta - e\theta. \qquad (2.40) \end{aligned}$$

Here (2.39) follows from Lemma 2.13. We note that the lower bound does not depend on whether d is a multiple of e. Next, we obtain an an upper bound on the same quantity.

Partition B into sets of size e, denoted by L_i. We will use $R = L \cup M, r' = 2$, in the helper node pooling.

case: $p + 2 < \eta$**:** In this case, the parameters satisfy the condition in Lemma 2.15.

$$\begin{aligned} H(S_B^L, S_B^M, S_M^L) &\leq \sum_{L_i \in B} H(S_{L_i}^L, S_{L_i}^M) + H(S_M^L) \\ &\leq \sum_{L_i \in B} e(2\beta - \theta) + e\beta \qquad (2.41) \\ &= (d - pe - e)(2\beta - \theta) + e\beta \\ &= (2d - 2ep - e)\beta - (d - ep - e)\theta, \qquad (2.42) \end{aligned}$$

where the inequality (2.41) is obtained using Lemma 2.14 and Lemma 2.15. Equations (2.40) and (2.42) are in contradiction if $d - ep - e > e \iff d > e(p + 2)$, which is true as $d \geq k = ae > (p + 2)e$.

case: $p{+}2 = \eta$**:** In this case, Lemma 2.16 is used to derive an upper bound on $H(S_B^L, S_B^M, S_M^L)$. Lemma 2.16 does not hold if $\eta = 2$. It holds for $\eta > 2 \iff k > 2e$. Thus, we consider $k > 2e$. We have

$$H(S_B^L, S_B^M, S_M^L) \leq \sum_{L_i \in B} H(S_{L_i}^L, S_{L_i}^M) + H(S_M^L) \leq \sum_{L_i \in B} e(\beta + \theta) + e\beta$$
$$= (d - ep)\beta + (d - ep - e)\theta. \tag{2.43}$$

Equations (2.40) and (2.43) are in contradiction when $\theta < \frac{d-ep-e}{d-ep}\beta$. □

Chapter 3

Exact-repair code constructions

3.1 Introduction

In the previous chapter, we introduced the problem of centralized repair of multiple failures. We distinguished between two repair modes: 1) functional repair, in which the replacement nodes may store different content than the replaced nodes, as opposed to 2) exact repair, where the replacement nodes should recover exactly the lost content. The functional repair tradeoff was solved through the concept of information flow graphs. The exact repair scenario is of paramount importance for most practical applications. Thus, understanding its fundamental bounds as well as designing multi-node repair codes are of interest. In the previous chapter, we derived outer bounds for exact-repair codes. In this chapter, we aim at designing exact-repair regenerating codes with minimum storage. Specifically, our target is to design (i) exact multi-node repair codes operating at the minimum storage point (MSMR point) of the functional repair tradeoff and (ii) Reed-Solomon codes that repair a single erasure with varying number of helpers. Constructions for other points of the tradeoff are presented in Chapter 4.

Related work

Constructing exact-repair regenerating codes for multiple failures has spawned a lot of interest in the research community. Several constructions with different characteristics have been proposed. Based on subspace interference alignment, the authors in [66] presented a maximum distance separable (MDS) code with asymptotic optimal repair for all $e \leq n-k$ and $k \leq d \leq n-e$ simultaneously. In [67], the authors presented an approach that enables single erasure MSR codes to recover from multiple failures simultaneously with near-optimal bandwidth. Based on simulations, [67] showed that their approach can provide efficient recovery of most of the failure patterns, but not all of them. In [64], the authors proved that Zigzag codes, which are MDS codes designed initially for repairing optimally single erasures [48], can also be used to optimally repair multiple erasures in a centralized manner. In [18], the authors independently proved that multiple failures can be repaired in Zigzag codes with optimal bandwidth. These constructions are suitable for high-rate codes, however, low-rate MSMR codes remains open.

Reed-Solomon (RS) codes [68] are widely used in practice, and thus repairing RS codes is of practical importance. The repair problem of RS codes has been investigated first in [69] for single erasure and in [70, 71, 72, 73, 74] for multiple erasures.

In [18], it is argued that cooperative regenerating codes can be used to construct centralized repair codes. The total bandwidth in this case is obtained by taking into account the bandwidth obtained from the helper nodes and disregarding the communication between the replacement nodes. Codes corresponding to the extreme points on the cooperative functional tradeoff have been developed [55]: minimum storage cooperative regenerating (MSCR) codes [56, 57, 55] and minimum bandwidth cooperative regenerating (MBCR) codes [58]. In particular, MSCR codes achieve the same performance as MSMR codes. In [75], the authors proved that the interference alignment MSR construction of [41], originally designed

for repairing any single node failure, can recover from multiple failures in a cooperative way. Specifically, it is shown that any set of systematic nodes, set of parity-check nodes, or pair of nodes can be repaired cooperatively with optimal bandwidth.

We note that most regenerating code constructions in the literature are designed for a fixed number of helpers d. Encoding and decoding is changed as the number of helpers d varies and consequently, the structure of the code, including the content of the nodes, changes. This aspect presents a considerable practical limitation as the number of helpers may vary due to several reasons, for example due to the unavailability of some nodes for maintenance or upgrade activities. As such, it is desirable to have regenerating codes with efficient repair for various number of helpers, simultaneously. The authors in [76] termed the above property as opportunistic repair and showed that for single-erasure functional repair regenerating codes, it is possible to construct codes that are simultaneously optimal for different numbers of helper nodes. The work in [77] studied the problem of designing MSR codes with efficient repair for various number of helpers, and this property is termed progressive engagement. Optimal constructions of MSR codes with flexible number of nodes appear in [73, 47, 78, 79].

Contributions of the chapter

The main contributions of this chapter are summarized as follows.

- When the number of erasures e satisfies $e \geq k$, k being the minimum number of nodes needed to reconstruct the entire data, the tradeoff reduces to a single point, for which we provide an explicit code construction.
- We formalize a construction for exact-repair MSMR codes. Given an instance of an exact linear MSR code, we present a framework to construct an instance of an exact linear MSMR code. We note here that [56] and [67] used a similar approach for MSCR codes and their numerical results, respectively.

- Based on this framework, we study the product-matrix (PM) MSR codes [60] and the interference alignment (IA) MSR construction in [41]. We prove the existence of PM and IA MSMR codes for any number of failures e, $e \leq n-k$ (Theorems 3.1, 3.2, 3.6). Moreover, for the IA code, we prove that the code can always efficiently recover from any set of $e \leq n-k$ node failures as long as the failed nodes are either all systematic nodes or all parity nodes (Theorem 3.3); for failures including both systematic and parity nodes, we derive explicit design conditions under which exact recovery is ensured, for some particular system parameters (Theorems 3.4, 3.5). We note here that unlike previous constructions, our codes are applicable when the code rate is low and they use a small subpacketization size of $\alpha = k-1$ or k.

- Finally, we investigate two approaches for repairing Reed-Solomon codes with varying number of helpers that allows repair with progressive engagement.

3.2 Construction when $k \leq e$

In the case of $k \leq e$, the optimal tradeoff reduces to a single point, so our MSMR construction in this section is also an MBMR code. The optimal parameters satisfy $\alpha = \frac{\mathcal{M}}{k}, \beta = \frac{\mathcal{M}}{d}$ and $\gamma = \mathcal{M}$. We note that the overall repair bandwidth $d\beta$ and the reconstruction bandwidth $k\alpha$ are the same. Therefore, one can achieve α and γ by dividing the data into k symbols and encoding them using an (n, k) MDS code (for example, a RS code). The repair can be done by downloading the full content of any k out of d helpers while not using $d-k$ helpers. Such repair is asymmetric in nature. We describe one alternative approach for achieving the repair with equal contribution from d helpers.

1. Divide the original file into kd symbols (that is $\mathcal{M} = kd$) and encode them using an (nd, kd) MDS code.

2. Store the encoded symbols at n nodes, such that each node stores $\alpha = d$ encoded symbols.

3. For reconstruction, from any k nodes, we obtain kd different symbols. By virtue of the MDS property, we can reconstruct the data.

4. For repair, each helper node transmits any $\beta = \frac{\mathcal{M}}{d} = k$ symbols. The replacement nodes receive dk different coded symbols, which are sufficient to reconstruct the whole data and thus regenerate the missing symbols.

Remark 3.1. *The above procedure works for a specific predetermined d. However, it can be generalized to support any value of d satisfying $k \leq d \leq n - e$. For instance, let $\delta = lcm(k, k+1, k+2, \ldots, n-e)$ (lcm denotes the least common multiple). Assume $\mathcal{M} = k\delta$. The file of size $\mathcal{M}$ is then encoded using an $(n\delta, k\delta)$ MDS code. Each node stores $\alpha = \frac{\mathcal{M}}{k} = \frac{k\delta}{k} = \delta$ coded symbols. To repair with d helpers, for any $k \leq d \leq n - e$, each node transmits any $\beta = \frac{\mathcal{M}}{d} = \frac{k\delta}{d}$ coded symbols for his node. Similarly, it can be seen that reconstruction is always feasible.*

3.3 Minimum storage codes framework

In the following sections, we discuss an explicit MSMR code construction method using existing MSR codes designed for single failures for $k > e$. We first describe the general framework, and then present two specific codes. We use the total bandwidth $\gamma = d\beta$ in this chapter. We denote the code parameters by $(n, k, d, e, \alpha, \gamma)$.

The framework described in this section has been developed in [67] for numerical simulations. We present it here in a formal and analytical way. Consider an instance of an exact linear (n, k, d, α, β) MSR code, where $\beta = \frac{\alpha}{d-k+1}$. Consider e nodes, indexed by $f_1, \ldots, f_e$, and other distinct $d - e + 1$ nodes, indexed by $h_1, \ldots, h_{d-e+1}$, such that $d - e + 1 \geq k$. Let

$\mathcal{H} = \{f_1, \ldots, f_e, h_1, \ldots, h_{d-e+1}\}$ and define $\mathcal{H}_{f_j} = \mathcal{H} \backslash \{f_j\}$. Consider the single-node repair algorithm corresponding to failed node f_j and helper nodes $\mathcal{H}_{f_j}$. We denote by $\mathbf{s}_{h,f_j}^{\mathcal{H}}$ the information sent by node h to repair node f_j, for helpers $h \in \mathcal{H}_{f_j}$. We drop the superscript $\mathcal{H}$ when it is clear from the context. The size of $\mathbf{s}_{h,f_j}$ is β symbols.

Now we construct an $(n, k, d-e+1, e, \alpha, e\beta)$ MSMR code. Upon failure of the e nodes $f_1, \ldots, f_e$, the centralized node carrying the repair connects to the set of $d-e+1$ helpers $h_1, \ldots, h_{d-e+1}$. Each helper node h_i transmits $e\beta$ symbols given by $\cup_{j=1}^{e} \{\mathbf{s}_{h_i,f_j}\}$. One can check that the parameters of an MSMR code in (2.13) are satisfied with equality.

The approach consists in using the underlying MSR repair procedure for each of the e failed nodes. Note that $\mathbf{s}_{h_i,f_j}$ can be obtained from the $d-e+1$ helpers, for $i \in [d-e+1]$. To this end, the MSR repair procedure requires $\mathbf{s}_{f_i,f_j}^{\mathcal{H}}$ for all $\{(i,j) : i,j \in [e], i \neq j\}$, which we treat as unknowns. Let $E_{i,j}(\cdot)$ denote the encoding function used to encode the information sent from node h_i to node f_j. Also, let $D_i(\cdot)$ denote the decoding function used by the MSR code to repair node f_i given information from d helpers. Then, we write

$$\begin{aligned} \mathbf{s}_{f_i,f_j}^{\mathcal{H}} &= E_{i,j}(\mathbf{w}_{f_i}) \\ &= E_{i,j}(D_i(\mathbf{s}_{h,f_i}^{\mathcal{H}}, h \in \mathcal{H}_{f_i})), \end{aligned} \tag{3.1}$$

where $\mathbf{w}_j$ denotes the content of node j, and $i, j \in [e], i \neq j$. Equation (3.1) generates $e(e-1)\beta$ linear equations in $e(e-1)\beta$ unknowns. Let $\mathbf{s}$ be a vector containing the unknowns $\mathbf{s}_{f_i,f_j}$. Then, we seek to form a system of linear equations as

$$A\mathbf{s} = \mathbf{b}, \tag{3.2}$$

where A is a known $(e(e-1)\beta \times e(e-1)\beta)$ matrix and b is a known $(e(e-1)\beta \times 1)$ vector.

If A is non-singular, one can thus recover $\mathbf{s}$. Then, the centralized node can recover the failed node $\mathbf{w}_{f_i}$ as $\mathbf{w}_{f_i} = D_i(\mathbf{s}_{h,f_i}, h \in \mathcal{H}_{f_i})$. We adopt the above framework throughout the section.

Remark 3.2. *While the described framework applies to codes with arbitrary rates, we focus in the sequel on low-rate codes. High-rate MSMR constructions have been presented in [80]. However, in the low-rate regime, our constructions perform better. For instance, for a target MSMR code with rate* $\frac{1}{2}$*, the construction in [80] yields a storage size* $\alpha = k^{2k-1}$*, while applying the above approach to IA codes [41] or to PM codes [81] results in a smaller storage size* $\alpha = k$ *and* $\alpha = k - 1$*, respectively.*

3.4 Product-matrix codes

In this section, we construct MSMR codes for any e erasures based on product-matrix (PM) codes [81]. The PM framework allows the design of MBR codes for any value of d and the design of MSR codes for $d \geq 2k - 2$. Moreover, the PM construction offers simple encoding and decoding and ensures optimal repair of all nodes. Product-matrix MSR codes are a family of scalar MSR codes, i.e., $\beta = 1$. We first focus on the case $d = 2k - 2$. Under this setup, $\alpha = d - k + 1 = k - 1$, $\mathcal{M} = k(k-1)$. The codeword is represented by an $(n \times \alpha)$ code matrix C such that its i-th row corresponds to the α symbols stored by the i-th node. The code matrix is given by

$$C = \Psi M, \Psi = \begin{bmatrix} \Phi & \Lambda\Phi \end{bmatrix}, M = \begin{bmatrix} S_1 \\ S_2 \end{bmatrix},$$

where Ψ is an $(n \times d)$ encoding matrix and M is a $(d \times \alpha)$ message matrix. S_1 and S_2 are $(\alpha \times \alpha)$ symmetric matrices constructed such that the $\binom{\alpha+1}{2}$ entries in the upper-triangular part of each of the two matrices are filled up by $\binom{\alpha+1}{2}$ distinct file symbols. Φ is an $(n \times \alpha)$

matrix and Λ is an $(n \times n)$ diagonal matrix. The elements of Ψ should satisfy:

1. any d rows of Ψ are linearly independent;
2. any α rows of Φ are linearly independent;
3. the n diagonal elements of Λ are distinct.

The above conditions may be met by choosing Ψ to be a Vandermonde matrix, in which case its i^{th} row is given by $\psi_i^t = \begin{bmatrix} 1 & \lambda_i & \cdots & \lambda_i^{d-1} \end{bmatrix}$. It follows that $\Lambda = \text{diag}\{\lambda_1^\alpha, \ldots, \lambda_n^\alpha\}$ for some coefficients $\lambda_1, \lambda_2, \ldots, \lambda_n$. In the following, we assume that Ψ is a Vandermonde matrix.

Repair of a single erasure in PM codes: The single erasure repair algorithm [81] is reviewed below. Let $\mathbf{w}_i^t$ denote the content stored at a failed node. Let ϕ_i^t be the i^{th} row of Φ. Then, $\mathbf{w}_i^t = \psi_i^t M = \begin{bmatrix} \phi_i^t & \lambda_i^\alpha \phi_i^t \end{bmatrix} M = \phi_i^t S_1 + \lambda_i^\alpha \phi_i^t S_2$. Let $\mathcal{H}_i = \{h_1, \ldots, h_d\}$ denote the set of d helpers. Each helper h transmits $s_{h,i} = \mathbf{w}_h^t \phi_i = \psi_h^t M \phi_i$ to the replacement node, who obtains $\Psi_{\mathcal{H}_i} M \phi_i$, where $\Psi_{\mathcal{H}_i}^t = \begin{bmatrix} \psi_{h_1} & \cdots & \psi_{h_d} \end{bmatrix}$.

Note that $\Psi_{\mathcal{H}_i}$ is invertible by construction. Thus, using the symmetry of S_1 and S_2, we obtain $(M\phi_i)^t = \begin{bmatrix} \phi_i^t S_1 & \phi_i^t S_2 \end{bmatrix}$. We can then reconstruct $\mathbf{w}_i^t = \phi_i^t S_1 + \lambda_i^\alpha \phi_i^t S_2$.

Repair of multiple erasures in PM codes: Given the symmetry of PM codes, we can assume w.l.o.g that nodes in $\mathcal{E} = \{1, \ldots, e\}$ have failed. Define $\mathcal{E}_i = \mathcal{E} \backslash \{i\}$. Let $\mathcal{H} = \{1, \ldots, d+1\}$. The centralized node connects to helper node $h \in \{e+1, \ldots, d+1\}$, and obtains $\{\psi_h^t M \phi_j, j \in \mathcal{E}\}$.

Let $\mathbf{s} = [s_{1,2}, s_{2,1}, \ldots, s_{1,e}, s_{e,1}, \ldots, s_{e-1,e}, s_{e,e-1}]^t$. Our goal is to express explicitly A and $\mathbf{b}$ as in (3.2).

Consider the repair of node $i \in \mathcal{E}$ by the set of helpers in $\mathcal{H}_i = \mathcal{H}\backslash\{i\}$. From the single-node repair, we write

$$\begin{aligned}
\mathbf{w}_i &= \begin{bmatrix} I_\alpha & \lambda_i^\alpha I_\alpha \end{bmatrix} \Psi_{\mathcal{H}_i}^{-1} \mathbf{s}_{\mathcal{H}_i}, \text{ such that} \\
\Psi_{\mathcal{H}_i}^t &= \begin{bmatrix} \psi_1 & \cdots & \psi_{i-1} & \psi_{i+1} & \cdots & \psi_{d+1} \end{bmatrix}, \\
\mathbf{s}_{\mathcal{H}_i}^t &= \begin{bmatrix} s_{1,i} & \cdots & s_{i-1,i} & s_{i+1,i} & \cdots & s_{d+1,i} \end{bmatrix}.
\end{aligned}$$

It follows that

$$\begin{aligned}
s_{i,j} &= \phi_j^t \mathbf{w}_i \\
&= \begin{bmatrix} \phi_j^t & \lambda_i^\alpha \phi_j^t \end{bmatrix} \Psi_{\mathcal{H}_i}^{-1} (\sum_{l \in \mathcal{H}_i} s_{l,i} \mathbf{e}_{l,i}) \\
&= \sum_{l \in \mathcal{E}_i} (\begin{bmatrix} \phi_j^t & \lambda_i^\alpha \phi_j^t \end{bmatrix} \Psi_{\mathcal{H}_i}^{-1} \mathbf{e}_{l,i}) s_{l,i} + \sum_{l=e+1}^{d+1} (\begin{bmatrix} \phi_j^t & \lambda_i^\alpha \phi_j^t \end{bmatrix} \Psi_{\mathcal{H}_i}^{-1} \mathbf{e}_{l,i}) s_{l,i}.
\end{aligned} \tag{3.3}$$

Here, for $l \in [d+1]\backslash\{i\}$, we use the column standard basis $\mathbf{e}_l$ and define

$$\mathbf{e}_{l,i} \triangleq \begin{cases} \mathbf{e}_l, & l < i, \\ \mathbf{e}_{l-1}, & l > i. \end{cases}$$

Note that the second term in (3.3) is known from the helpers. Moreover, to compute (3.3), one may use the inverse of Vandermonde's matrix formula [82]. Let $h \in \{1, \ldots, d\}$, we have

$$(\Psi_{\mathcal{H}_i}^{-1} \mathbf{e}_{l,i})_h = \frac{\gamma_h(l,i)}{\prod_{m \in \mathcal{H}_i \backslash \{l\}} (\lambda_l - \lambda_m)} = \frac{\gamma_h(l,i)}{\sum_{j=1}^d \gamma_j(l,i) \lambda_l^{j-1}}, \tag{3.4}$$

where the subscript h in $(\cdot)_h$ means the h-th entry, and

$$\gamma_h(l,i) = (-1)^{d-h} \sum_{m_1 < \ldots < m_{d-h} \in \mathcal{H}_i \backslash \{l\}} \lambda_{m_1} \ldots \lambda_{m_{d-h}}. \tag{3.5}$$

We obtain

$$\begin{bmatrix} \phi_j^t & \lambda_i^\alpha \phi_j^t \end{bmatrix} \Psi_{\mathcal{H}_i}^{-1} \mathbf{e}_{l,i} = \frac{\sum_{h=1}^{\alpha} (\gamma_h(l,i) + \lambda_i^\alpha \gamma_{h+\alpha}(l,i)) \lambda_j^{h-1}}{\sum_{h=1}^{d} \gamma_h(l,i) \lambda_l^{h-1}}. \tag{3.6}$$

Therefore, one can construct A and $\mathbf{b}$ in (3.2) as follows:

- The entries of $\mathbf{b}$ are indexed with (i,j), corresponding to $s_{i,j}$. The entry of $\mathbf{b}$ at index (i,j) is given by
  $\sum_{l=e+1}^{d+1} (\begin{bmatrix} \phi_j^t & \lambda_i^\alpha \phi_j^t \end{bmatrix} \Psi_{\mathcal{H}_i}^{-1} \mathbf{e}_{l,i}) s_{l,i}$.
- Index the $e(e-1)$ rows (and columns respectively) of A with (i,j). A has zero in all entries except: For every row in A indexed by (i,j):
  - the entry at column indexed by (i,j) is -1.
  - for $l \in \mathcal{E}_i$, the entry at column indexed by (l,i) is given by $\begin{bmatrix} \phi_j^t & \lambda_i^\alpha \phi_j^t \end{bmatrix} \Psi_{\mathcal{H}_i}^{-1} \mathbf{e}_{l,i}$ as in (3.6).

For clear presentation, we first prove the existence of product-matrix MSMR codes for 2 erasures, and then prove the result for general e.

Theorem 3.1. *There exists $(n, k, 2k-3, 2, k-1, 2)$ product-matrix MSMR codes, defined over a large enough finite field, such that any two erasures can be optimally repaired.*

Proof. In this case, the matrix A of (3.2) is given by

$$A = \begin{bmatrix} -1 & \begin{bmatrix} \phi_2^t & \lambda_1^\alpha \phi_2^t \end{bmatrix} \Psi_{\mathcal{H}_1}^{-1} \mathbf{e}_{2,1} \\ \begin{bmatrix} \phi_1^t & \lambda_2^\alpha \phi_1^t \end{bmatrix} \Psi_{\mathcal{H}_2}^{-1} \mathbf{e}_{1,2} & -1 \end{bmatrix}.$$

From (3.4), noting that $\mathcal{H}_1\backslash\{2\} = \mathcal{H}_2\backslash\{1\}$, we obtain the determinant of A to be

$$\begin{aligned}
|A| &= 1 - \begin{bmatrix}\phi_2^t & \lambda_1^\alpha\phi_2^t\end{bmatrix}\Psi_{\mathcal{H}_1}^{-1}\mathbf{e}_{2,1}\begin{bmatrix}\phi_1^t & \lambda_2^\alpha\phi_1^t\end{bmatrix}\Psi_{\mathcal{H}_2}^{-1}\mathbf{e}_{1,2} \\
&= 1 - \frac{(\sum_{h=1}^{\alpha}\lambda_2^{h-1}(\gamma_h(1,2)+\lambda_1^\alpha\gamma_{h+\alpha}(1,2)))}{\sum_{h=1}^{d}\lambda_2^{h-1}\gamma_h(1,2)\sum_{h=1}^{d}\lambda_1^{h-1}\gamma_h(1,2)}(\sum_{h=1}^{\alpha}\lambda_1^{h-1}(\gamma_h(1,2)+\lambda_2^\alpha\gamma_{h+\alpha}(1,2))) \\
&\triangleq 1 - \frac{N(\lambda_1,\ldots,\lambda_{d+1})}{D(\lambda_1,\ldots,\lambda_{d+1})}.
\end{aligned}$$

$|A|$ can be viewed as a rational function of $(\lambda_1,\ldots,\lambda_{d+1})$, as N and D are polynomials in $(\lambda_1,\ldots,\lambda_{d+1})$. We want to show that the following polynomial is not identically zero:

$$P(\lambda_1,\ldots,\lambda_{d+1}) \triangleq D(\lambda_1,\ldots,\lambda_{d+1})|A| = D(\lambda_1,\ldots,\lambda_{d+1}) - N(\lambda_1,\ldots,\lambda_{d+1}).$$

Let $y_\alpha = (-1)^\alpha\lambda_3\cdots\lambda_{\alpha+2}$, $y_{\alpha-1} = (-1)^{\alpha-1}\lambda_3\cdots\lambda_{\alpha+1}$. Then, it can be seen that P contains the term

$$y_\alpha y_{\alpha-1}(\lambda_1^{d-1}+\lambda_2^{d-1}-\lambda_1^\alpha\lambda_2^{\alpha-1}-\lambda_2^\alpha\lambda_1^{\alpha-1}),$$

which is not zero. Hence, $P(\lambda_1,\ldots,\lambda_{d+1})$ is a non-zero polynomial. The PM construction, when based on a Vandermonde matrix, requires $\lambda_i^\alpha \neq \lambda_j^\alpha$ [81], or equivalently, $g(\lambda_1,\lambda_2) \triangleq \lambda_i^\alpha - \lambda_j^\alpha \neq 0$. Let $Q(\lambda_1,\ldots,\lambda_n)$ denote the polynomial obtained by varying the set of helpers and failure patterns, taking the product of all corresponding polynomials P, and also multiplied by all g for all pairs of two nodes. Then, Q is not identically zero. By Combinatorial Nullstellensatz [83], we can find assignments of the variables $\{\lambda_1,\ldots,\lambda_n\}$ over a large enough finite field, such that the polynomial is not zero. Equivalently, we can guarantee the successful optimal repair of any two erasures among the n storage nodes. □

Theorem 3.2. *There exists $(n,k,2k-e-1,e,k-1,e)$ product-matrix MSMR codes, defined over a large enough finite field, such that any e erasures can be optimally repaired.*

Proof. We only consider $e > 2$, as the case of $e = 2$ is covered by Theorem 3.1. Entries in each column indexed by $s_{i,j}$ in A is either -1 or some other $(e-1)$ non-zero entries whose denominator is the same and given by $\prod_{m \in \mathcal{H} \setminus \{i\}} (\lambda_i - \lambda_m)$. We multiply this common denominator to all entries in the column $s_{i,j}$, for all pairs $i \neq j$. When λ_i's are chosen to be distinct, this does not change the singularity of A. Denote this transformed matrix by B. Using (3.6), the entry of B in row (i,j) and column (l,m) is a polynomial in $\lambda_1, \ldots, \lambda_{d+1}$:

$$B_{(i,j),(l,m)} = \begin{cases} -\sum_{h=1}^{d} \gamma_h(i,j)\lambda_i^{h-1}, & l = i, m = j, \\ \sum_{h=1}^{\alpha} \left(\gamma_h(l,i)\lambda_j^{h-1} + \gamma_{h+\alpha}(l,i)\lambda_i^{\alpha}\lambda_j^{h-1}\right), & m = i, \\ 0, & \text{otherwise.} \end{cases}$$

Notice that $e + \alpha - 1 \leq k - 1 + \alpha - 1 = d - 1$. Let $y = (-1)^{\alpha-1}\lambda_{e+1} \cdots \lambda_{e+\alpha-1}$, which is a term in $\gamma_{\alpha+1}(i,j)$ for all (i,j) by (3.5). We observe that there is a single term $\pm y\lambda_i^{\alpha}$ in the polynomial $B_{(i,j),(l,m)}$ for the non-zero entries of B.

Recall that the Leibniz formula for determinant of a $(m \times m)$ matrix B is given by $|B| = \sum_{\sigma} \text{sgn}(\sigma) \prod_{i} b_{\sigma(i),i}$,where σ is a permutation from the permutation group S_m, sgn is the sign function of permutations, and $b_{i,j}$ is the entry (i,j) of B.

Claim 1. The term $T = \prod_{i=1}^{e} (y\lambda_i^{\alpha})^{e-1}$ in $|B|$ has a non-zero coefficient.

Claim 1 implies that $|B|$ is not a zero polynomial. Then, proceeding as in the proof in Theorem 3.1, by Combinatorial Nullstellensatz [83], we can find assignments of the variables $\{\lambda_1, \ldots, \lambda_n\}$ over a large enough finite field, such that the code guarantees optimal repair of any set of e erasures.

Next, we prove Claim 1. Note that the term T can be created if and only if we take the single term $\pm y\lambda_i^{\alpha}$ in the non-zero entries of B (depending on the permutation σ). Therefore, it is easy to see that the coefficient of term T in $|B|$ is the determinant of the following

$(e(e-1) \times e(e-1))$ matrix C

$$C_{(i,j),(l,m)} = \begin{cases} -1, & l = i, m = j, \\ 1, & m = i, \\ 0, & \text{otherwise.} \end{cases}$$

One can verify that C is diagonalizable, and the eigenvalues satisfy:

- Eigenvalue $e-2$ has multiplicity 1, with the corresponding (right) eigenvector $(1, 1, \ldots, 1)^t$.
- Eigenvalue -2 has multiplicity $e-1$, with the corresponding eigenspace $\{(x_{1,2}, \ldots, x_{e,e-1})^t : x_{1,j} = x_{1,2}, \forall j \in [e]\backslash\{1\}, x_{i,j} = x_{i,1}, \forall i \in [e]\backslash\{1\}, \forall j \in [e]\backslash\{i\}, x_{1,2} + \sum_{i=2}^{e} x_{i,1} = 0\}$ of dimension $e-1$.
- Eigenvalue -1 has multiplicity $e(e-2)$, with the corresponding eigenspace $\{(x_{1,2}, \ldots, x_{e,e-1})^t : \sum_{1 \leq i \leq e, i \neq j} x_{i,j} = 0, \forall j \in [e]\}$ of dimension $e(e-2)$.

To ensure that $|C| \neq 0$, a sufficient condition is to require the finite field to have a characteristic such that the eigenvalues $\{e-2, -2, -1\}$ are non-zero and $|C| \neq 0$. Therefore, Claim 1 is proved and the theorem statement follows. □

Remark 3.3. *There exists product-matrix MSMR codes, defined over a large enough finite field, that simultaneously repair any $e \in [n-k]$ erasures with optimal bandwidth. Indeed, let $\widetilde{Q} = \prod_{e=2}^{n-k} Q_e$, where Q_e is the polynomial corresponding to the code constraints for e erasures such that matrix A in (3.2) is invertible. Recall that the reconstruction process for PM codes requires that $\alpha_i^\alpha - \alpha_j^\alpha \neq 0$ for $\alpha_i \neq \alpha_j$. Let $g(\lambda_1, \ldots, \lambda_n) = \prod_{1 \leq i < j \leq n} (\lambda_j^\alpha - \lambda_i^\alpha)$. Let $Q(\lambda_1, \ldots, \lambda_n) = g(\lambda_1, \ldots, \lambda_n)\widetilde{Q}(\lambda_1, \ldots, \lambda_n)$. By Theorem 3.1 and Theorem 3.2, Q is not zero and the result follows by Combinatorial Nullstellensatz.*

We note that the sufficient condition that matrix C is invertible in the proof of Theorem

3.2 is not necessary for the existence of PM codes. Indeed, as it will be shown in Example 3.1, we can construct PM codes with optimal multi-node repair property over finite fields of characteristic 2.

Example 3.1. *Consider the product-matrix code with* $n = 11, k = 6, d = 10$, $\alpha = 5$. *The code is defined over* $\mathbb{F}_{2^6}$ *with* $\Lambda = diag\{\lambda_1^\alpha, \ldots, \lambda_{11}^\alpha\}$ *and* $\lambda_i = g^{i-1}$ *with* g *being the generator of the multiplicative group of* $\mathbb{F}_{2^6}$. *Recall that with the above choice of* λ_i, *any field of size at least* $n\alpha = 55$ *is sufficient to meet the PM code requirements [81]. We first consider repair of* $e = 2$ *erasures. One can check that out of the* $\binom{11}{2} = 55$ *possible 2 failure patterns, 2 patterns are not recoverable according to* (3.2)*:* $\mathcal{E} \in \{\{1, 2\}, \{10, 11\}\}$. *Considering the same code structure, for* $e = 3$ *erasures, one observes that out of the* $\binom{11}{3} = 165$ *possible 3 failure patterns, 5 patterns are not recoverable:* $\mathcal{E} \in \{\{1, 2, 11\}, \{2, 3, 7\}, \{2, 4, 8\}, \{3, 4, 7\}, \{5, 9, 10\}\}$. *It is worth noting that a lazy repair strategy can be beneficial in the following way: if nodes 10 and 11 fail, i.e.,* $\mathcal{E} = \{10, 11\}$, *then, one can optimally repair any 3 erasures* $\mathcal{E} \in \{\{i, 10, 11\}, i \neq 10, i \neq 11\}$. *Finally, as suggested by Theorem 3.1 and Theorem 3.2, we find that increasing the underlying field size to* $\mathbb{F}_{2^8}$ *suffices to ensure optimal repair of all two and three erasure patterns in this scenario.*

Remark 3.4. *Following the code shortening procedure described in [81], we construct an* $(n, k, d-e+1, e, k-1, e)$ *product-matrix MSMR code* $\mathcal{C}$ *with optimal repair for any* $e \in [n-k]$ *erasures such that* $2k - 2 \leq d \leq n - 1$. *First, as described in Remark 3.3, we consider an* $(n + (d-2k+2), k + (d-2k+2), d + (d-2k+2) - e + 1, e, d - k + 1, \beta = 1)$ *product-matrix MSMR code* $\mathcal{C}'$ *in systematic form with varying* $e \in [n - k]$. *Note that the code* $\mathcal{C}'$ *exists because the parameters satisfy Theorem 3.2. The first* $(d - 2k + 2)$ *systematic nodes of* $\mathcal{C}'$ *are set to zeros. Then, the target code* $\mathcal{C}$ *is formed by deleting the first* $(d - 2k - 2)$ *rows in each code matrix of* $\mathcal{C}'$. *It can be seen that the repair procedure for* e *erasures in* $\mathcal{C}$ *can be done by invoking that of the original code* $\mathcal{C}'$, *which leads to the result.*

3.5 Interference alignment codes

In this section, we give explicit code coefficient conditions for optimal MSMR codes from IA codes [41] for $e = 2, 3, 4$ erasures, and for any $e \leq k$ erasures from only the systematic (or only the parity) nodes. Moreover, we show the existence of MSMR codes for any $e \leq k$ erasures.

The scalar MSR IA code construction is based on interference alignment techniques. The code is systematic and defined over a finite field $\mathbb{F}_q$ with optimal repair bandwidth for the case $\frac{k}{n} \leq \frac{1}{2}$ and $d \geq 2k-1$. We focus on the case $n = 2k, d = 2k-1, \beta = 1$. In this scenario, the storage size is $\alpha = d - k + 1 = k$.

Notation. For an invertible matrix B, we define its inverse transpose to be $B' \triangleq (B^{-1})^t$. The columns of $B^{'}$ constitute the dual basis of the column vectors of B. Recall that $B_{i,j}$ denotes the (i,j)-th element of matrix B. We use the following symbols to denote the transmission of information during repair operations.

- $s_{i,j}$: from systematic node i to parity node j.
- $r_{i,j}$: from systematic node i to systematic node j.
- $\bar{s}_{i,j}$: from parity node i to systematic node j.
- $\bar{r}_{i,j}$: from parity node i to parity node j.

The IA code is constructed as below. Consider k linearly independent vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$, $\mathbf{v}_i \in \mathbb{F}_q^k, i \in [k]$. Let

$$V = \Big[\mathbf{v}_1, \ldots, \mathbf{v}_k\Big], U = \kappa^{-1} V^{'} P, \tag{3.7}$$

where every submatrix of the $(k \times k)$ matrix P is invertible and κ is an arbitrary non-zero constant in $\mathbb{F}_q$ satisfying $\kappa^2 - 1 \neq 0$. Let $\mathbf{w}_l, l \in [k]$ denote the content of systematic node l and $\bar{\mathbf{w}}_i$ the content of parity node i , $i \in [k]$. Let $\mathbf{u}_i, \mathbf{v}_i, \mathbf{u}_i^{'}, \mathbf{v}_i^{'}$ be the i-th column of U, V, U', V', respectively. Then, by the construction in [41],

$$\bar{\mathbf{w}}_i^t = \sum_{j=1}^{k} \mathbf{w}_j^t G_j^{(i)}, G_j^{(i)} = \mathbf{u}_i \mathbf{v}_j^t + P_{j,i} I,$$

such that the matrix $G_j^{(i)}$ indicates the encoding submatrix for parity node i, associated with information unit j, and I is the identity matrix of size $(k \times k)$.

Repair of a systematic node: Assume that systematic node l fails. The general repair procedure is described in [41]. In this section, we explicitly develop the exact expression of $\mathbf{w}_l$ as it is needed later in repairing multiple erasures. Each systematic node $j \in [k] \backslash \{l\}$ transmits $r_{j,l} = \mathbf{w}_j^t \mathbf{v}_l^{'}$. Each parity node $i \in [k]$ transmits $\bar{s}_{i,l} = \bar{\mathbf{w}}_i^t \mathbf{v}_l^{'}$. Noting that $G_j^{(i)} \mathbf{v}_l^{'} = \mathbb{1}_{\{j=l\}} \mathbf{u}_i + P_{j,i} \mathbf{v}_l^{'}$, it follows that

$$\bar{s}_{i,l} = \mathbf{w}_l^t (\mathbf{u}_i + P_{l,i} \mathbf{v}_l^{'}) + \sum_{j \in [k] \backslash \{l\}} P_{j,i} r_{j,l}.$$

Canceling the interference from systematic nodes, and arranging the contributions of parity nodes in matrix form, we write

$$\begin{bmatrix} \bar{s}_{1,l} - \sum_{j \in [k] \backslash \{l\}} P_{j,1} r_{j,l} \\ \vdots \\ \bar{s}_{k,l} - \sum_{j \in [k] \backslash \{l\}} P_{j,k} r_{j,l} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1^t + P_{l,1} \mathbf{v}_l^{'t} \\ \vdots \\ \mathbf{u}_k^t + P_{l,k} \mathbf{v}_l^{'t} \end{bmatrix} \mathbf{w}_l = \frac{1}{\kappa} P^t (I + \kappa \mathbf{e}_l \mathbf{e}_l^t) V^{-1},$$

where the last equality is obtained by substituting U by its expression in (3.7). Using the Sherman-Morrison formula, for an invertible square matrix A of size $(k \times k)$ and vectors $\mathbf{u}, \mathbf{v}$

of length k,

$$(A+\mathbf{u}\mathbf{v}^t)^{-1} = A^{-1} - \frac{A^{-1}\mathbf{u}\mathbf{v}^t A^{-1}}{1+\mathbf{v}^t A^{-1}\mathbf{u}},$$

we obtain that $(\frac{1}{\kappa}P^t(I+\kappa\mathbf{e}_l\mathbf{e}_l^t)V^{-1})^{-1} = U^{'} - \frac{\kappa^2}{1+\kappa}V\mathbf{e}_l\mathbf{e}_l^t P^{'}$, it follows that

$$\mathbf{w}_l = (U^{'} - \frac{\kappa^2}{1+\kappa}V\mathbf{e}_l\mathbf{e}_l^t P^{'}) \begin{bmatrix} \bar{s}_{1,l} - \sum\limits_{j\in[k]\setminus\{l\}} P_{j,1}r_{j,l} \\ \vdots \\ \bar{s}_{k,l} - \sum\limits_{j\in[k]\setminus\{l\}} P_{j,k}r_{j,l} \end{bmatrix}. \tag{3.8}$$

Repair of a parity node: The repair of a parity node is optimally achieved through the duality property of IA codes resulting in a structure that is also conducive to interference alignment. Indeed, inverting the roles of parity and systematic nodes, it follows from [41] that

$$\mathbf{w}_i = \sum_{j=1}^{k} \bar{\mathbf{w}}_j^t G_j^{'(i)}; G_j^{'(i)} = \frac{1}{1-\kappa^2}(\mathbf{v}_i^{'}\mathbf{u}_j^{'t} - \kappa^2 P_{i,j}^{'} I).$$

Assume parity node l fails, then systematic node i transmits $s_{i,l} = \mathbf{w}_i^t\mathbf{u}_l$ and parity node j sends $\bar{r}_{j,l} = \bar{\mathbf{w}}_j^t\mathbf{u}_l$. Note that $G_j^{'(i)}\mathbf{u}_l = \frac{1}{1-\kappa^2}(-\kappa^2 u_j + \mathbb{1}_{\{j=l\}}\mathbf{v}_i^{'})$. It follows that

$$s_{i,l} = \frac{1}{1-\kappa^2}\bar{\mathbf{w}}_l^t(\mathbf{v}_i^{'} - \kappa^2 P_{i,l}^{'}\mathbf{u}_l) + \sum_{j\in[k]\setminus\{l\}} \frac{-\kappa^2}{1-\kappa^2} P_{i,j}^{'}\bar{r}_{j,l}.$$

Combining information from different helpers, we obtain after simplification

$$\begin{bmatrix} s_{1,l} + \frac{\kappa^2}{1-\kappa^2}\sum\limits_{j\in[k]\setminus\{l\}} P_{1,j}^{'}\bar{r}_{j,l} \\ \vdots \\ s_{k,l} + \frac{\kappa^2}{1-\kappa^2}\sum\limits_{j\in[k]\setminus\{l\}} P_{k,j}^{'}\bar{r}_{j,l} \end{bmatrix} = \frac{1}{1-\kappa^2}\begin{bmatrix} \mathbf{v}_1^{'t} - \kappa^2 P_{1,l}^{'}\mathbf{u}_l^t \\ \vdots \\ \mathbf{v}_k^{'t} - \kappa^2 P_{k,l}^{'}\mathbf{u}_l^t \end{bmatrix} \bar{\mathbf{w}}_l = \frac{\kappa}{1-\kappa^2} P^{'}(I-\kappa\mathbf{e}_l\mathbf{e}_l^t)U^t\bar{\mathbf{w}}_l,$$

where the last equality is obtained by replacing $V^{-1} = \kappa P^{'} U^{t}$. Inverting the system of equations and using the Sherman-Morrison formula, we obtain

$$\bar{\mathbf{w}}_l = ((1-\kappa^2)V + (1+\kappa)U^{'}\mathbf{e}_l\mathbf{e}_l^{t}P^{t}) \begin{bmatrix} s_{1,l} + \frac{\kappa^2}{1-\kappa^2} \sum\limits_{j\in[k]\setminus\{l\}} P^{'}_{1,j}\bar{r}_{j,l} \\ \vdots \\ s_{k,l} + \frac{\kappa^2}{1-\kappa^2} \sum\limits_{j\in[k]\setminus\{l\}} P^{'}_{k,j}\bar{r}_{j,l} \end{bmatrix}. \tag{3.9}$$

Repair of multiple erasures. The goal is to construct the system of linear equations as in (3.2). We need to derive the equations relating the information transferred across the failed systematic and parity nodes according to (3.1). Consider a systematic node $l \in [k]$ and a parity node $m \in [k]$, from (3.8), we write

$$\begin{aligned}
s_{l,m} &= \mathbf{u}_m^{t}\mathbf{w}_l \\
&= (\mathbf{u}_m^{t}U^{'} - \frac{\kappa^2}{1+\kappa}\mathbf{u}_m^{t}V\mathbf{e}_l\mathbf{e}_l^{t}P^{'}) \begin{bmatrix} \bar{s}_{1,l} - \sum\limits_{j\in[k]\setminus\{l\}} P_{j,1}r_{j,l} \\ \vdots \\ \bar{s}_{k,l} - \sum\limits_{j\in[k]\setminus\{l\}} P_{j,k}r_{j,l} \end{bmatrix} \\
&= (\mathbf{e}_m^{t} - \frac{\kappa}{1+\kappa}P_{l,m}\mathbf{e}_l^{t}P^{'}) \begin{bmatrix} \bar{s}_{1,l} - \sum\limits_{j\in[k]\setminus\{l\}} P_{j,1}r_{j,l} \\ \vdots \\ \bar{s}_{k,l} - \sum\limits_{j\in[k]\setminus\{l\}} P_{j,k}r_{j,l} \end{bmatrix} \qquad (3.10) \\
&= (\mathbf{e}_m^{t} - \frac{\kappa}{1+\kappa}P_{l,m}\mathbf{e}_l^{t}P^{'})(\sum_{j\in[k]} \bar{s}_{j,l}\mathbf{e}_j - \sum_{j\in[k]\setminus\{l\}} r_{j,l}P^{t}\mathbf{e}_j) \\
&= (1 - \frac{\kappa}{1+\kappa}P_{l,m}P^{'}_{l,m})\bar{s}_{m,l} - \sum_{j\in[k]\setminus\{m\}} (\frac{\kappa}{1+\kappa}P_{l,m}P^{'}_{l,j})\bar{s}_{j,l} - \sum_{j\in[k]\setminus\{l\}} P_{j,m}r_{j,l}. \qquad (3.11)
\end{aligned}$$

Here (3.10) is obtained by noting that $U^{t}V = \frac{1}{\kappa}P^{t}$, and (3.11) follows using $P^{'}P^{t} = I$. Similarly, consider two systematic nodes $l_1, l_2 \in [k], l_1 \neq l_2$, starting from (3.8) and noting

that $V'^{t}U' = \kappa P'$, we obtain after simplification

$$r_{l_1,l_2} = \mathbf{v}_{l_2}^{'t}\mathbf{w}_{l_1} = \sum_{j\in[k]}(\kappa P'_{l_2,j})\bar{s}_{j,l_1} - \kappa r_{l_2,l_1}. \tag{3.12}$$

Proceeding in a similar way, for a systematic node $l \in [k]$ and a parity node $m \in [k]$, starting from (3.9), we obtain

$$\bar{s}_{m,l} = \mathbf{v}_{l}^{'t}\bar{\mathbf{w}}_{m} = (1-\kappa^2+\kappa(1+\kappa)P'_{l,m}P_{l,m})s_{l,m} + \sum_{j\in[k]\setminus\{l\}}(\kappa(1+\kappa)P'_{l,m}P_{j,m})s_{j,m}$$
$$+ \sum_{j\in[k]}(\kappa^2 P'_{l,j})\bar{r}_{j,m}. \tag{3.13}$$

Finally, consider two parity nodes $m_1, m_2 \in [k], m_1 \neq m_2$, starting from (3.9), we obtain

$$\bar{r}_{m_1,m_2} = \mathbf{u}_{m_2}^{t}\bar{\mathbf{w}}_{m_1} = \sum_{j\in[k]}(\frac{1-\kappa^2}{\kappa}P_{j,m_2})s_{j,m_1} + \kappa\bar{r}_{m_2,m_1}. \tag{3.14}$$

The details of deriving (3.12), (3.13) and (3.14) can be found in Appendix 3.7.1. Equations (3.11), (3.12), (3.13) and (3.14) can thus be used to derive A and $\mathbf{b}$ as defined in (3.2).

In the following theorem, we show that the IA code already provides optimal repair for systematic (respectively parity) failures, without the need to modify the coding matrices.

Theorem 3.3. *In the interference alignment MSR code [41], it is possible to optimally repair any set of $e \leq k$ systematic (respectively parity) failures.*

Proof. Assume w.l.o.g that nodes $\{1,\ldots,e\}$ have failed. Let $\mathbf{s} = \left[r_{1,2}, r_{2,1}, \ldots, r_{e-1,e}, r_{e,e-1}\right]^t$.

Then, from (3.12), it follows that A is a block-diagonal matrix given by

$$A = \begin{bmatrix} 1 & \kappa & & & \\ \kappa & 1 & & & \\ & & \ddots & & \\ & & & 1 & \kappa \\ & & & \kappa & 1 \end{bmatrix}.$$

It follows that $|A| = (1-\kappa^2)^{\frac{e(e-1)}{2}} \neq 0$ as $\kappa^2 \neq 1$ by design. The same procedure applies to any set of e failures among parity nodes using equation (3.14). □

Theorem 3.4. *The interference alignment MSR code achieves optimal simultaneous repair of one systematic node l and one parity node m if $P_{l,m}(P^{-1})_{m,l} \neq 1$.*

Proof. Assume that systematic node l and parity node m failed. Let $\mathbf{s} = [s_{l,m}, \bar{s}_{m,l}]^t$. From (3.11), we obtain

$$s_{l,m} = (1 - \frac{\kappa}{1+\kappa} P_{l,m} P'_{l,m}) \bar{s}_{m,l} + c_1,$$

where c_1 is a known quantity independent of $\mathbf{s}$. Similarly, from (3.13), we obtain

$$\bar{s}_{m,l} = (1 - \kappa^2 + \kappa(1+\kappa) P_{l,m} P'_{l,m}) s_{m,l} + c_2,$$

where c_2 is a known quantity independent of $\mathbf{s}$. It follows that A, as defined in (3.2), is given by

$$A = \begin{bmatrix} -1 & 1 - \frac{\kappa}{1+\kappa} P_{l,m} P'_{l,m} \\ 1 - \kappa^2 + \kappa(1+\kappa) P_{l,m} P'_{l,m} & -1 \end{bmatrix}.$$

After simplification, we have $|A| \neq 0 \iff \kappa^2 (P_{l,m} P'_{l,m} - 1)^2 \neq 0 \iff P_{l,m}(P^{-1})_{m,l} \neq 1$, as

$\kappa \neq 0$. $\square$

Combining Theorems 3.3 and 3.4 we know that $(2k, k, 2k-2, 2, k, 2)$ MSMR codes for 2 erasures can be constructed through IA codes. We point out that Theorems 3.3 and 3.4 have been derived in [75] for cooperative repair, using a different technique. Recall that MSCR codes are in particular MSMR codes [18]. However, their technique cannot be extended to more than two node failures including systematic and parity nodes [75].

Theorem 3.5. *The interference alignment MSR code achieves optimal simultaneous repair of*

- *two systematic failures l_1, l_2 and one parity failure m if*

$$1 - P_{l_1,m}(P^{-1})_{m,l_1} - P_{l_2,m}(P^{-1})_{m,l_2} \neq 0,$$

- *one systematic failure l and two parity failures m_1, m_2 if*

$$1 - P_{l,m_1}(P^{-1})_{m_1,l} - P_{l,m_2}(P^{-1})_{m_2,l} \neq 0,$$

- *three systematic failures l_1, l_2, l_3 and one parity failure m if*

$$1 - P_{l_1,m}(P^{-1})_{m,l} - P_{l_2,m}(P^{-1})_{m,l_2} - P_{l_3,m}(P^{-1})_{m,l_3} \neq 0,$$

- *one systematic failure l and three parity failures m_1, m_2, m_3 if*

$$1 - P_{l,m_1}(P^{-1})_{m_1,l} - P_{l,m_2}(P^{-1})_{m_2,l} - P_{l,m_3}(P^{-1})_{m_3,l} \neq 0,$$

- *two systematic failures l_1, l_2 and two parity failures m_1, m_2 if*

$$
\begin{aligned}
&1 - P_{l_1,m_1}(P^{-1})_{m_1,l_1} - P_{l_1,m_2}(P^{-1})_{m_2,l_1} - P_{l_2,m_1}(P^{-1})_{m_1,l_2} \\
&- P_{l_2,m_2}(P^{-1})_{m_2,l_2} + P_{l_1,m_1}(P^{-1})_{m_1,l_1}P_{l_2,m_2}(P^{-1})_{m_2,l_2} \\
&+ P_{l_1,m_2}(P^{-1})_{m_2,l_1}P_{l_2,m_1}(P^{-1})_{m_1,l_2} - P_{l_1,m_1}(P^{-1})_{m_1,l_2}P_{l_2,m_2}(P^{-1})_{m_2,l_1} \\
&- P_{l_1,m_2}(P^{-1})_{m_2,l_2}P_{l_2,m_1}(P^{-1})_{m_1,l_1} \neq 0.
\end{aligned}
$$

Proof. The proof follows along similar lines as Theorem 3.4 by constructing A using (3.11), (3.12), (3.13) and (3.14). The explicit expression of $|A|$ can then be obtained for example by using the Symbolic Math Toolbox of MATLAB, from which the above conditions can be readily obtained (the MATLAB source code is available online*.) □

Combining Theorems 3.3 and 3.5 we know that $(2k, k, 2k-e, e, k, e)$ MSMR codes for $e = 3, 4$ erasures can be constructed through IA codes.

Remark 3.5. *Deriving an exact condition under which the recovery of multiple failures for large e is not straightforward. However, we suspect that the general formula is given by the following expression*

$$
|A| = k^{2sp}(1-k^2)^{\binom{s}{2}+\binom{p}{2}}\Big(1 - \sum_{\substack{L\subset S, J\subset P,\\ |L|=|J|,\\ |J|\leq \min(s,p)}} \sum_{\sigma\in\Pi_{L,J}} \sum_{\sigma'\in\Pi_{L,J}} sgn(\sigma)sgn(\sigma') \prod_{i\in L} P_{i,\sigma(i)} \prod_{j\in J} P'_{j,\sigma'(j)}\Big)^e, \tag{3.15}
$$

where $\Pi_{L,J}$ is the group of permutations between the two sets L and J (L and J are ordered in increasing order), and element of order h in L is mapped to element of order h in J) and $sgn(\sigma)$ refers to the sign of a permutation σ. For example, if $L = \{1, 2, 3\}$, $J = \{2, 3, 4\}$ and $\sigma(1) = 3, \sigma(2) = 4, \sigma(3) = 2$. Then, sgn(σ)=1. One can check that the formulas in Theorems 3.3, 3.4 and 3.5 satisfy (3.15). A general proof of (3.15) is still open.

* `https://github.com/Marwen-Zorgui/Centralized_repair_IA`

Example 3.2. *Consider the IA code with $n = 8, k = 4, d = 7$, $\alpha = 4, \beta = 1$. The code is defined over the finite field $\mathbb{F}_{2^5}$ and let g be the generator of its multiplicative group. Let P be a Vandermonde matrix given by*

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & g & g^2 & g^3 \\ 1 & g^2 & g^4 & g^6 \\ 1 & g^3 & g^6 & g^9 \end{bmatrix}.$$

Using Theorems 3.3, 3.4 and 3.5, one can check that any two, three and four erasures can be repaired optimally using our repair framework.

In the following theorem, we provide an existence proof of IA MSMR codes for multiple erasures.

Theorem 3.6. *There exists $(2k, k, 2k-e, e, k, e)$ interference alignment MSMR codes, defined over a large enough finite field, such that any $e \leq k$ erasures can be optimally repaired.*

Proof. From Theorem 3.3, we know that if the errors are all either systematic or parity nodes, then efficient repair is possible. Thus, we only need to analyze the case of a mixture of systematic and parity failures.

Consider $e \leq k$ failures consisting of q systematic nodes and p parties nodes, indexed by the sets $\mathcal{Q}$ and $\mathcal{P}$. W.lo.g, assume that $\mathcal{Q} = [q]$ and $\mathcal{P} = [p]$. Let $\mathbf{s}$ denote the vector of unknowns such that pairs $(r_{i,j}, r_{j,i}), (\bar{r}_{i,j}, \bar{r}_{j,i})$ and $(s_{i,j}, \bar{s}_{j,i})$ are grouped together. Using (3.11), (3.12), (3.13) and (3.14), we construct A as in (3.2). Denote the determinant of A as $F(\kappa, P_{i,j}, P'_{i,j}, i \in \mathcal{Q}, j \in \mathcal{P}) \triangleq |A|$. The rows and columns of A are indexed by $\{r_{i,j}, s_{i,j}, \bar{r}_{i,j}, \bar{s}_{i,j}\}$. Let $M_{i,j}$ denote the minor in A corresponding to $A_{i,j}$. Similarity, $N_{i,j}$ denotes the minor in P corresponding to $P_{i,j}$. As $P' = (P^{-1})^t$, $P'_{i,j} = \frac{(-1)^{i+j} N_{j,i}}{|P|}$, one concludes that F is a rational function in $(\kappa, P_{i,j}, i, j \in [k])$.

Claim 2. F is not identically zero for any $q, p \geq 0, q + p = e \leq k$.

If Claim 2 holds, then the theorem is proved due to the following argument. By symmetry among the systematic (respectively parity) nodes of IA codes, any e-erasure pattern corresponds to a non-zero rational function F. Recall from [41] that the reconstruction process requires that every submatrix of P is invertible. This can be translated into a polynomial constraint given by $g(P_{i,j}, i \in \mathcal{Q}, j \in \mathcal{P}) \neq 0$. Let $T \triangleq g \prod_{e \text{ erasures}} F$. Here the product is over all possible e erasures, and the rational function F depend on the erasure pattern. Then, it follows that T is a non-zero rational polynomial in $(\kappa, P_{i,j}, (i,j) \in [k] \times [n-k])$. By Combinatorial Nullstellensatz [83], we can find assignments of the variables $(\kappa, \{P_{i,j}\})$ over a large enough finite field, such that the code guarantees optimal recovery of any set of e erasures.

Next, we prove Claim 2. We assume first that $q \leq \frac{k}{2}$. Let

$$P_{i,j} = 0 \ \forall \ (i,j) \in \mathcal{Q} \times \mathcal{P}. \tag{3.16}$$

Note that one can always construct a (normalized) invertible matrix P satisfying (3.16), so we can assume $|P| = 1$. Thus F is a polynomial. We will show

$$F(\kappa, P_{i,j} = 0, P'_{i,j}, (i,j) \in \mathcal{Q} \times \mathcal{P}) = F(\kappa, P_{i,j} = 0, P'_{i,j} = 0, (i,j) \in \mathcal{Q} \times \mathcal{P}) \neq 0,$$

which implies

$$F(\kappa, P_{i,j}, P'_{i,j}, (i,j) \in \mathcal{Q} \times \mathcal{P}) \neq 0.$$

To this end, we first prove that $F(\kappa, P_{i,j} = 0, P'_{i,j}, (i,j) \in \mathcal{Q} \times \mathcal{P})$, viewed as a polynomial of $(\kappa, \{P'_{i,j}\})$, does not depend on $\{P'_{i,j}\}$. From (3.12) and (3.13), one can check that $P'_{i,j}$ appears in A at entries given by

- $A_{r_{l,i},\bar{s}_{j,l}}$ for $l \in \mathcal{Q}\backslash\{i\}$,
- $A_{\bar{s}_{m,i},\bar{r}_{j,m}}$ for $m \in \mathcal{P}\backslash\{j\}$.

For any $l \in \mathcal{Q}\backslash\{i\}$, consider the two columns in A indexed by $r_{l,i}$ and $r_{i,l}$. Both columns have non-zero entries only at rows indexed with $r_{l,i}$ and $r_{i,l}$. Then, after removing entries at row $r_{l,i}$, it follows that both columns become linearly dependent, as both columns are scalar multiples of the same standard basis vector. Thus, $M_{r_{l,i},\bar{s}_{j,l}} = 0$.

The example in (3.18) illustrates the case of two systematic failures, given by systematic nodes 1 and 2, and one parity failure, given parity node 1. In this case $s = \left[s_{1,1}, \bar{s}_{1,1}, r_{1,2}, r_{2,1}, s_{2,1}, \bar{s}_{1,2}\right]$. Setting $P_{i,j} = 0$ for all $i = 1, 2, j = 1$ and looking at the submatrix of A by removing row $r_{1,2}$ in (3.18), it can be seen that columns $r_{1,2}, r_{2,1}$ are dependent, hence its corresponding minor $M_{r_{1,2},\bar{s}_{2,1}} = 0$.

Similarly, for any $m \in \mathcal{P}\backslash\{j\}$, consider the two rows in A indexed by $\bar{r}_{j,m}$ and $\bar{r}_{m,j}$. Both rows have non-zero entries only at columns indexed with $\bar{r}_{j,m}$ and $\bar{r}_{m,j}$. Then, after removing entries at column $\bar{r}_{j,m}$, it follows that both rows become linearly dependent. Thus, $M_{\bar{s}_{m,i},\bar{r}_{j,m}} = 0$.

The minors in A of all terms corresponding to $P'_{i,j}$ are thus equal to zero. Therefore, w.l.o.g, one can assume that $P'_{i,j} = 0, \forall (i,j) \in \mathcal{Q} \times \mathcal{P}$. It follows that A is block-diagonal matrix such that

- Row/column pairs $(r_{i,j}, r_{j,i})$ correspond to $\begin{bmatrix} -1 & -\kappa \\ -\kappa & -1 \end{bmatrix}$,
- Row/column pairs $(\bar{r}_{i,j}, \bar{r}_{j,i})$ correspond to $\begin{bmatrix} -1 & \kappa \\ \kappa & -1 \end{bmatrix}$,

- Row/column pairs $(s_{i,j}, \bar{s}_{j,i})$ correspond to $\begin{bmatrix} -1 & 1 \\ 1-\kappa^2 & -1 \end{bmatrix}$,
- Other entries are 0.

Therefore, $|A| = \kappa^{2qp}(1-\kappa^2)^{\binom{q}{2}+\binom{p}{2}} \neq 0$, as $\kappa \neq 0$ and $\kappa^2 \neq 1$.

Assume now that $q > \frac{k}{2}$. Then, $p \leq \frac{k}{2}$. Proceeding similarly, one can show that if $P'_{ij} = 0, \forall (i,j) \in \mathcal{Q} \times \mathcal{P}$, then, all terms P_{ij} have no impact on $|A|$ and one obtains similarly $|A| = \kappa^{2rp}(1-\kappa^2)^{\binom{r}{2}+\binom{p}{2}}$.

$$A = \tag{3.17}$$

$$\begin{array}{c} \begin{matrix} s_{1,1} & \bar{s}_{1,1} & r_{1,2} & r_{2,1} & s_{2,1} & \bar{s}_{1,2} \end{matrix} \\ \begin{bmatrix} -1 & 1-\frac{\kappa P_{11}P'_{11}}{\kappa+1} & 0 & -P_{21} & 0 & 0 \\ 1-\kappa^2+\kappa(\kappa+1)P_{11}P'_{11} & -1 & 0 & 0 & \kappa(\kappa+1)P_{21}P'_{11} & 0 \\ 0 & \kappa P'_{21} & -1 & -\kappa & 0 & 0 \\ 0 & 0 & -\kappa & -1 & 0 & \kappa P'_{11} \\ 0 & 0 & -P_{11} & 0 & -1 & 1-\frac{\kappa P_{21}P'_{21}}{\kappa+1} \\ \kappa(\kappa+1)P_{11}P'_{21} & 0 & 0 & 0 & 1-\kappa^2+\kappa(\kappa+1)P_{21}P'_{21} & -1 \end{bmatrix} \end{array} \begin{matrix} s_{1,1} \\ \bar{s}_{1,1} \\ r_{1,2} \\ r_{2,1} \\ s_{2,1} \\ \bar{s}_{1,2} \end{matrix}$$

(3.18)

□

3.6 Progressive repair of Reed-Solomon codes

RS codes are widely used in practical distributed storage. For example, RS codes with parameters $(n = 14, k = 10)$ are reported to be used in Facebook clusters [11]. Moreover, as discussed in the literature, it is desirable to a have a code obstruction where the number of helpers involved in the repair process is flexible, while achieving efficient repair bandwidth

[77]. In this section, we study the repair of RS codes with various numbers of helpers. We first review the repair procedure of RS codes from [69] and show how one can use this procedure for repairing RS codes for various number of helpers. Our focus in this section is on repairing single erasures.

3.6.1 Repairing Reed-Solomon codes overview

Consider a finite field $\mathbb{F}$. $\mathbb{F}[X]$ denotes the set of polynomials with coefficients in $\mathbb{F}$.

Definition 3.1. *The Reed-Solomun code $RS(A,k)$ of dimension k over a finite field $\mathbb{F}$ with evaluation points $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq \mathbb{F}$ is the set*

$$RS(A,k) = \{f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n) : f \in \mathbb{F}[X] \text{ is a polynomial of degree at most } k-1\}.$$

The k coefficients of the polynomials in the definition of $RS(A,k)$ correspond to information symbols.

The work in [69] provides a characterization of (linear) exact repair schemes of RS codes. The repair problem is shown to be equivalent to a search of a set of polynomials satisfying certain rank constraints. We note that the original field $\mathbb{F}$ as a vector space over any subfield $\mathbb{B}$. For a set of points $Z \subseteq \mathbb{F}$, $\dim_{\mathbb{B}}(Z)$ is the dimension of the vector space generated by the elements in Z when viewed as elements of the vector space over $\mathbb{B}$.

A repair strategy over a subfield $\mathbb{B}$ is a repair scheme where each helper node transfers symbols from the subfield $\mathbb{B}$. The repair bandwidth b is the total number of symbols in $\mathbb{B}$ transmitted by all helper nodes. Using all the information collected from the helpers, a replacement node can exactly repair its stored symbol. The following theorem states the polynomial characterization of the linear exact repair of RS codes.

Theorem 3.7 ([69])**.** Let $\mathbb{B}$ be a subfield of $\mathbb{F}$, where the degree of $\mathbb{F}$ over $\mathbb{B}$ is t. Let

$A \subseteq \mathbb{F}$ be any set of evaluation points. The following statements are equivalent for the code $RS(A, k)$.

1. There is a linear repair scheme for $RS(A, k)$ over $\mathbb{B}$ with bandwidth b.
2. For each $\alpha^* \in A$, there is a set $\mathcal{P}(\alpha^*) \subseteq \mathbb{F}[X]$ of t polynomials of degree less than $d - k + 1$, so that

$$\dim_{\mathbb{B}}(\{p(\alpha^*) : p \in \mathcal{P}(\alpha^*)\}) = t,$$
$$\max_{\alpha^* \in A} \sum_{\alpha \in A \setminus \{\alpha^*\}} \dim_{\mathbb{B}}(\{p(\alpha) : p \in \mathcal{P}(\alpha^*)\}) \leq b.$$

Here α^* corresponds to the lost node, and $\mathcal{P}(\alpha^*)$ is the set of polynomials used during repair. The dimension conditions in the theorem states that the t evaluations of the polynomials $\mathcal{P}(\alpha^*)$ should have full rank when evaluated at the lost point α^*, and a low rank when evaluated at the helper points $\alpha \in A \setminus \{\alpha^*\}$.

Using the RS repair characterization in this theorem, one can repair any given node with a varying number of helpers (i.e., evaluation points) without having to change the code structure. For instance, as the number of helpers changes, the number of available evaluation points for repair changes and one can repair a lost node by specifying a new set of polynomials satisfying the constraints in Theorem 3.7. Then, the specified polynomials are used to instantiate the repair algorithm as in [68, Remark 5].

3.6.2 Trace polynomials approach

Trace polynomials have been shown in [69] to approach a lower bound for RS codes in case $A = \mathbb{F}$. One can use trace polynomials also for a number of evaluation points less than the field size. The construction proceeds as follows. First, one chooses a basis $\mathcal{E}$ for $\mathbb{F}$ over $\mathbb{B}$.

Let α^* denote the evaluation point at the lost node, then, ones chooses the set of polynomials given by

$$\mathcal{P}(\alpha^*) = \left\{ \frac{\mathrm{tr}_{\mathbb{F}/\mathbb{B}}(\zeta(X-\alpha^*))}{X-\alpha}; \zeta \in \mathcal{E} \right\},$$

with the field trace given by $\mathrm{tr}_{\mathbb{F}/\mathbb{B}}(\alpha) = \sum_{i=0}^{t-1} \alpha^{|\mathbb{B}|^i}$ and $|\mathbb{B}|$ is the size of $\mathbb{B}$. Each polynomial has degree $\frac{|\mathbb{F}|}{|\mathbb{B}|} - 1$. Thus, for a fixed $\mathbb{B}$ and d, to use trace polynomials, the condition $\frac{|\mathbb{F}|}{|\mathbb{B}|} - 1 < d + 1 - k$ should be satisifed. Equivalently, the number of helpers d has to satisfy $k + \frac{|\mathbb{F}|}{|\mathbb{B}|} \leq d + 1 \leq |\mathbb{F}|$, achieving a bandwidth (in bits) of [69]

$$b = \log_2(\mathbb{B}) \sum_{\alpha \neq \alpha^*} \dim_{\mathbb{B}} \{p(\alpha) : p \in \mathcal{P}(\alpha^*)\} = (d-1)\log_2(\mathbb{B}).$$

We note that increasing the number of available evaluations (i.e., d), allows us to repair over a smaller subfield $\mathbb{B}$, resulting in a lower bandwidth.

Figure 3.1 illustrates the performance achieved with trace polynomials for $k = 6$ and $\mathbb{F} = \mathrm{GF}(2^4)$. Figure 3.1 illustrates also a lower bound that was developed in reference [69] for RS codes and the performance obtained by a naive scheme. A decrease in the overall repair bandwidth is obtained each time the number of helpers d increases enough to allow the repair over a smaller subfield.

3.6.3 Search-based approach

From Theorem 3.7, the repair performance depends on the specific choice of polynomials. As the optimal choice of polynomials is not known when $d+1$ is less than the field size $|\mathbb{F}|$, we propose a computer-based search to find a good set of polynomials achieving low repair bandwidth. We recall that the degree of each of the t polynomials is less than $n-k$, with $n = d+1$. Then, the search space is of size $|\mathbb{F}|^{t(n-k)}$. To make the computer search feasible,

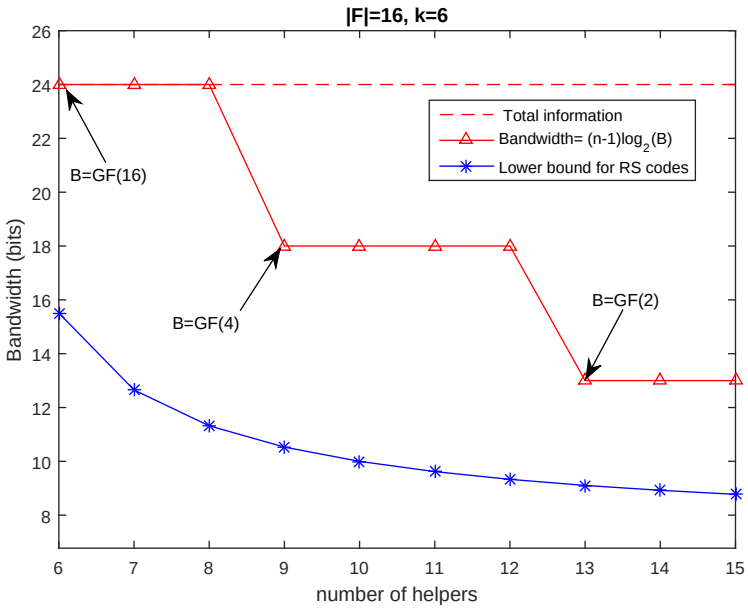


Figure 3.1: Bandwidth achieved with trace polynomials.

one can for instance reduce the search space by considering only polynomials of degree $n-k-1$ with roots from the set of evaluation points $A \setminus \{\alpha^*\}$ as in [69]. The computer search is computationally expensive as it involves rank operations and computations over a finite field. However, as for practical storage systems, the number of repair scenarios is limited, the search algorithm may be run ahead of time and the obtained polynomial coefficients may be stored for when they are required by the system.

Figure 3.2 illustrates the results obtained for a RS code with $k = 4$ and $\mathbb{F} = \mathrm{GF}(2^8)$. The results are shown for two sub-fields $\mathbb{B}$ which are $\mathbb{B} = \mathrm{GF}(2^2)$ and $\mathbb{B} = \mathrm{GF}(2^4)$, corresponding to $t = 2$ and $t = 4$, respectively. As expected, the results obtained using a larger subfield are no-better than using a smaller subfield. For instance, a set of polynomial that is feasible for a larger $\mathbb{B}$ provides a feasible set for a smaller subfield.

Remark 3.6. Assume one has a scheme for repairing a RS code defined over a field $\mathbb{F}$ with dimension t over a subfield $\mathbb{B}_1$. That is we have a set of t polynomials $\{p_1, \ldots, p_t\}$, each of

degree less than $n-k$ such that

$$\dim_{\mathbb{B}_1}(\{p_i(\alpha^*) : i = 1, \ldots, t\}) = t,$$
$$b \geq \sum_{\alpha \in A \setminus \{\alpha^*\}} \dim_{\mathbb{B}_1}(\{p_i(\alpha) :: i = 1, \ldots, t\}).$$

Now, we consider another subfield $\mathbb{B}_2$ that is a subfield of $\mathbb{B}_1$. This means that $\mathbb{B}_1$ can be seen as vector space over $\mathbb{B}_2$ with dimension r. Consider a basis $\{b_1, \ldots, b_r\}$ of $\mathbb{B}_1$ over $\mathbb{B}_2$. We note also that $\mathbb{F}$ is a vector space over $\mathbb{B}_2$ with dimension rt. Consider the set of rt polynomials $\{q_{ij}(X) = b_i p_j(X), i \leq r \text{ and } j \leq t\}$. Then, as we are multiplying by elements from $\mathbb{B}_1$, we have

$$\dim_{\mathbb{B}_2}\{q_{ij}(\alpha) : i = 1, \ldots, r, j = 1, \ldots, t\} \leq r \dim_{\mathbb{B}_1}\{p_j(\alpha) : j = 1, \ldots, t\}.$$

Moreover, the degree of each polynomial $q_{ij}(X)$ is at most $n-k$ and one can check that $\{q_{ij}(\alpha^*) = b_i p_j(\alpha^*), i \leq r \text{ and } j \leq t\}$ has dimension rt over $\mathbb{B}_2$. Finally, noting that $\log_2(|\mathbb{B}_1|) = r \log_2(|\mathbb{B}_2|)$, one can see that the overall bandwidth is at most the same bandwidth achieved over $\mathbb{B}_1$.

Remark 3.7. To compute the rank of a set of field elements over a given subfield, one can use the primitive normal polynomial over finite field [84] to generate a basis for rank computations. A normal basis of $\mathrm{GF}(q^n)$ over $\mathrm{GF}(q)$ is a basis of the form $\{\eta, \eta^q, \ldots, \eta^{q^{n-1}}\}$ for some $\eta \in \mathrm{GF}(q^n)$. Using a primitive normal polynomial $f(x)$ in the construction of the finite field, a root η of $f(x)$ generates a normal basis of $\mathrm{GF}(q^n)$ over $\mathrm{GF}(q^d)$ for all d divisor of n [84, Theorem 3.43]. We refer the reader to [85] for more details about normal basis over finite fields and to [84] for a list of primitive normal polynomials.

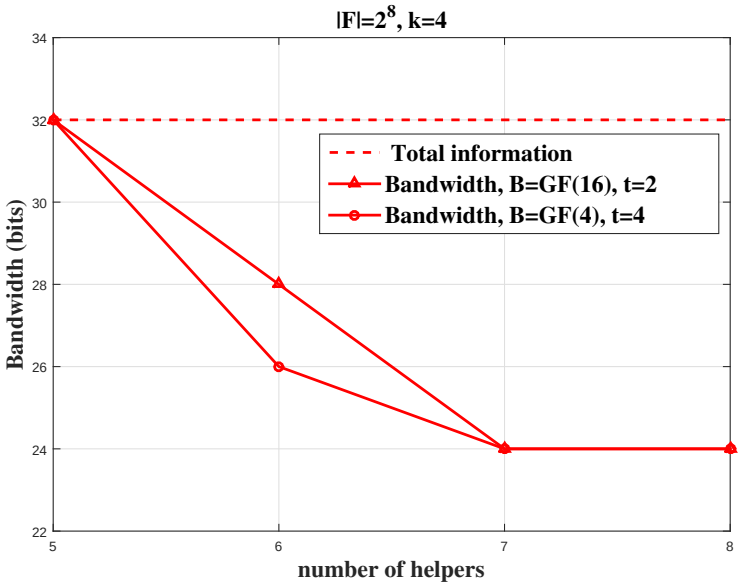


Figure 3.2: Bandwidth achieved with a restricted search.

3.7 Conclusion

In this chapter, We studied the problem of designing multi-node exact-repair regenerating codes. In the case of $e \geq k$, the functional tradeoff reduces to a single point, for which we provided a code construction achieving it. We described a general framework for converting single erasure MSR codes to MSMR codes. Then we applied the framework to product-matrix codes and interference alignment codes. Finally, we investigated the repair Reed-Solomon codes with varying number of helpers.

Appendix

3.7.1 Derivations of (3.12), (3.13) and (3.14) for IA codes

Consider two systematic nodes $l_1, l_2 \in [k], l_1 \neq l_2$, starting from (3.8) and noting that $V^{'t}U^{'} = \kappa P^{'}$, we obtain after simplification

$$r_{l_1,l_2} = \mathbf{v}_{l_2}^{'t}\mathbf{w}_{l_1} = \mathbf{v}_{l_2}^{'t}(U^{'} - \frac{\kappa^2}{1+\kappa}V\mathbf{e}_{l_1}\mathbf{e}_{l_1}^{t}P^{'}) \begin{bmatrix} \bar{s}_{1,l_1} - \sum\limits_{j\neq l_1} p_{j,1}r_{j,l_1} \\ \vdots \\ \bar{s}_{k,l_1} - \sum\limits_{j\neq l_1} p_{j,k}r_{j,l_1} \end{bmatrix}$$

$$= \kappa\mathbf{e}_{l_2}^{t}P^{'} \begin{bmatrix} \bar{s}_{1,l_1} - \sum\limits_{j\neq l_1} p_{j,1}r_{j,l_1} \\ \vdots \\ \bar{s}_{k,l_1} - \sum\limits_{j\neq l_1} p_{j,k}r_{j,l_1} \end{bmatrix} = \sum_{j\in[k]} (\kappa P_{l_2,j}^{'})\bar{s}_{j,l_1} - \kappa r_{l_2,l_1}.$$

Consider two parity nodes $m_1, m_2 \in [k], m_1 \neq m_2$, starting from (3.9), we obtain

$$\bar{r}_{m_1,m_2} = \mathbf{u}_{m_2}^{t}\bar{\mathbf{w}}_{m_1}$$

$$= \mathbf{u}_{m_2}^{t}((1-\kappa^2)V + (1+\kappa)U^{'}\mathbf{e}_{m_1}\mathbf{e}_{m_1}^{t}P^{t}) \begin{bmatrix} s_{1,m_1} + \frac{\kappa^2}{1-\kappa^2}\sum\limits_{j\neq m_1} P_{1,j}^{'}\bar{r}_{j,m_1} \\ \vdots \\ s_{k,m_1} + \frac{\kappa^2}{1-\kappa^2}\sum\limits_{j\neq m_1} P_{k,j}^{'}\bar{r}_{j,m_1} \end{bmatrix}$$

$$= \frac{1-\kappa^2}{\kappa}\mathbf{e}_{m_2}^{t}P^{t} \begin{bmatrix} s_{1,m_1} + \frac{\kappa^2}{1-\kappa^2}\sum\limits_{j\neq m_1} P_{1,j}^{'}\bar{r}_{j,m_1} \\ \vdots \\ s_{k,m_1} + \frac{\kappa^2}{1-\kappa^2}\sum\limits_{j\neq m_1} P_{k,j}^{'}\bar{r}_{j,m_1} \end{bmatrix} = \sum_{j\in[k]} (\frac{1-\kappa^2}{\kappa}P_{j,m_2})s_{j,m_1} + \kappa\bar{r}_{m_2,m_1}.$$

Proceeding in a similar way, for a systematic node $l \in [k]$ and a parity node $m \in [k]$, starting from (3.9), we obtain

$$
\begin{aligned}
\bar{s}_{m,l} &= \mathbf{v}_l^{'t} \bar{\mathbf{w}}_m \\
&= \mathbf{v}_l^{'t}((1-\kappa^2)V + (1+\kappa)U^{'}\mathbf{e}_m\mathbf{e}_m^{t}P^t) \begin{bmatrix} s_{1,m} + \frac{\kappa^2}{1-\kappa^2}\sum\limits_{j\neq m} P^{'}_{1,j}\bar{r}_{j,m} \\ \vdots \\ s_{k,m} + \frac{\kappa^2}{1-\kappa^2}\sum\limits_{j\neq m} P^{'}_{k,j}\bar{r}_{j,m} \end{bmatrix} \\
&= ((1-\kappa^2)\mathbf{e}_l^t + (1+\kappa)\kappa P^{'}_{l,m}\mathbf{e}_m^{t}P^t) \begin{bmatrix} s_{1,m} + \frac{\kappa^2}{1-\kappa^2}\sum\limits_{j\neq m} P^{'}_{1,j}\bar{r}_{j,m} \\ \vdots \\ s_{k,m} + \frac{\kappa^2}{1-\kappa^2}\sum\limits_{j\neq m} P^{'}_{k,j}\bar{r}_{j,m} \end{bmatrix} \\
&= (1-\kappa^2+\kappa(1+\kappa)P^{'}_{l,m}P_{l,m})s_{l,m} + \sum_{j\in[k]\setminus\{l\}} (\kappa(1+\kappa)P^{'}_{l,m}P_{j,m})s_{j,m} \\
&+ \sum_{j\in[k]} (\kappa^2 P^{'}_{l,j})\bar{r}_{j,m}.
\end{aligned}
$$

Chapter 4

Code constructions for interior points

4.1 Introduction

In Chapter 2, we studied the problem of centrally repairing multiple node failures in DSSs and introduced regenerating codes, parameterized with $(\mathcal{M}, n, k, d, e, \alpha, \beta)$, as a solution to the repair problem. We use $\bar{\alpha} = \frac{\alpha}{\mathcal{M}}, \bar{\beta} = \frac{\beta}{\mathcal{M}}$ to denote the normalized storage size and repair bandwidth, respectively. In particular, we characterized the functional repair tradeoff between α and β for multi-node recovery. Let $g = \lceil \frac{k}{e} \rceil, t = k - (g-1)e$. The normalized functional tradeoff can then be written as follows

$$\min(t\bar{\alpha}, d\bar{\beta}) + \sum_{p=0}^{g-2} \min(e\bar{\alpha}, (d-t-pe)\bar{\beta}) \geq 1, \tag{4.1}$$

Inequality (4.1) gives $g-1$ linear bounds:

$$(t+pe)\bar{\alpha} + \sum_{i=p}^{g-2} (d-t-ie)\bar{\beta} \geq 1, \quad p = 0, \ldots, g-2. \tag{4.2}$$

On the functional tradeoff, the MSMR and MBMR points are given by

$$(\bar{\alpha}_{MSMR}, \bar{\beta}_{MSMR}) = (\frac{1}{k}, \frac{e}{k(d-k+e)}), \tag{4.3}$$

$$(\bar{\alpha}_{MBMR}, \bar{\beta}_{MBMR}) = (\frac{2(d-k+gt)}{gt(2d-k+t)}, \frac{1}{dg-e\binom{g}{2}}). \tag{4.4}$$

In Chapter 3, we presented a framework for designing exact-repair MSMR codes. In this chapter, we go beyond the MSMR point and seek to construct codes operating at *interior* points. Interior points exhibit various tradeoffs between the storage per node and the bandwidth per helper, and can be of interest for storage system designers.

Definition 4.1. *We define interior points to be points that lie between the functional MSMR and MBMR points, given by (4.3) and (4.4), respectively. Precisely, an* $(\bar{\alpha}, \bar{\beta})$ *point is an interior point if it satisfies:* $\bar{\alpha} \geq \bar{\alpha}_{MSMR}, \bar{\beta} \geq \bar{\beta}_{MBMR}, (\bar{\alpha}, \bar{\beta}) \neq (\bar{\alpha}_{MSMR}, \bar{\beta}_{MSMR}), (\bar{\alpha}, \bar{\beta}) \neq (\bar{\alpha}_{MBMR}, \bar{\beta}_{MBMR})$.

Related work

For single erasure, several works have investigated construction of exact-repair codes operating at interior points [4, 86, 87, 88]. In particular, for a $(k+1, k, k, 1)$ system, the exact-repair tradeoff region, when restricted to linear codes [4, 87]. We recall that cooperative regenerating codes can be used to construct centralized repair codes. Of interest to us in this chapter are minimum bandwidth cooperative regenerating (MBCR) codes [58]. MBCR codes can be used as centralized repair codes, but do not correspond to MBMR codes. The MBCR point is given by

$$(\bar{\alpha}_{MBCR}, \bar{\beta}_{MBCR}) = (\frac{2d+e-1}{k(2d-k+e)}, \frac{2e}{k(2d-k+e)}). \tag{4.5}$$

An important family of MSMR codes is *universal* MSMR codes. Universal MSMR codes achieve simultaneously the optimal repair bandwidth for all $e \leq n-k, k \leq d \leq n-e$. Universal MSMR codes are the building blocks in our code constructions for interior points in this chapter. The interference alignment codes in [66] are asymptotically universal. In [78], the authors presented two families of universal MSMR codes. Given integers n and $n-k$, the first family of codes satisfy $\alpha = s^n$, where $s = lcm(1, 2, \ldots, n-k)$, lcm being the least common multiple, and they can be constructed over any base field $\mathbb{F}$ of size $|\mathbb{F}| \geq sn$. The second family of codes satisfy the optimal access property. That is, the repair can be accomplished by accessing the amount of data that equals the optimal β_{MSMR} in (4.3). The codes in the second family satisfy also $\alpha = s^n$, where $s = lcm(1, 2, \ldots, n-k)$, and they can be constructed over any base field $\mathbb{F}$ of size $|\mathbb{F}| \geq n+1$. The authors in [57] constructed universal MSCR codes (and thus MSMR codes), which can be constructed over a base field of size $|\mathbb{F}| \geq (n-k)n$, with $\alpha = \prod_{1 \leq e \leq n-d \leq n-k} ((e+s-1)(s-1)^{e-1})^{\binom{n}{e}}$, where $s = d-k+1$.

Contributions of the chapter

The main contributions of this chapter are summarized as follows.

- We show that exact-repair MBCR codes achieve an interior point, that lies near the MBMR point, when $k \equiv 1 \mod e$ (Theorem 4.1).
- We improve upon the layered construction presented in [87], which is concerned with single node repair, to construct a family of regenerating codes that is capable of repairing multiple failures (Construction 4.1 and Construction 4.2). To the best of our knowledge, this is the first known construction for multiple erasures in the interior points of the tradeoff. Moreover, our code possesses the universal repair property: it allows a varying number of failures and a varying number of helpers, such that $1 \leq e \leq n-k, k \leq d \leq n-e$.

- For the $(k+e, k, k, e)$ system, we first prove the optimality of a particular constructed code using the functional repair tradeoff (4.1) (Theorem 4.2); combining the achievable points via our construction and also the MBCR point, we then determine the best achievable region obtained by space-sharing among all known points.

- Finally, we study the adaptive repair problem of multiple erasures in MBR codes and present an MBR construction with optimal repair, simultaneously for varying numbers of helpers and varying numbers of erasures (Theorem 4.3).

4.2 MBCR codes as centralized repair codes

MBCR codes, given by (4.5), can be used as centralized multi-node repair regenerating codes [18]. In the case $e \mid k$, it is shown that MBCR codes achieve the MBMR bandwidth, i.e, $\gamma_{\text{MBCR}} = \gamma_{\text{MBMR}}$. In the case of $e \nmid k$, by imposing a certain entropy accumulation property on the entropy of any group of r nodes, [18] showed that the bandwidth achieved by MBCR codes is optimal. It is important to note here that, it can be checked that the entropy constraint condition results in $\gamma_{\text{MBCR}} > \gamma_{\text{MBMR}}$ for $e \nmid k$. Moreover, in both cases, it is not clear whether MBCR lies on the exact tradeoff of centralized repair. The next theorem determines the cases in which MBCR codes meet the centralized functional repair tradeoff (but does not correspond to the minimum bandwidth point on the functional repair curve). As a consequence, for such cases, MBCR codes meet the exact repair tradeoff as well.

Theorem 4.1. *Assume $1 < e \leq k$, then, minimum bandwidth cooperative regenerating codes meet the centralized functional repair tradeoff if and only if $k \equiv 1 \mod e$.*

Proof. We write $k = \eta e + r$, such that $\eta = \lfloor \frac{k}{e} \rfloor$ and $0 \leq r \leq e-1$. When $e \mid k$, from (4.4), it follows that $\gamma_{\text{MBMR}} = \gamma_{\text{MBCR}}$ and $\alpha_{\text{MBCR}} = \alpha_{\text{MBMR}} + \frac{\mathcal{M}(e-1)}{k(2d-k+e)}$. Thus, $\alpha_{\text{MBMR}} < \alpha_{\text{MBCR}}$. When $e \nmid k$, from (4.4) and the functional repair tradeoff expression in the previous

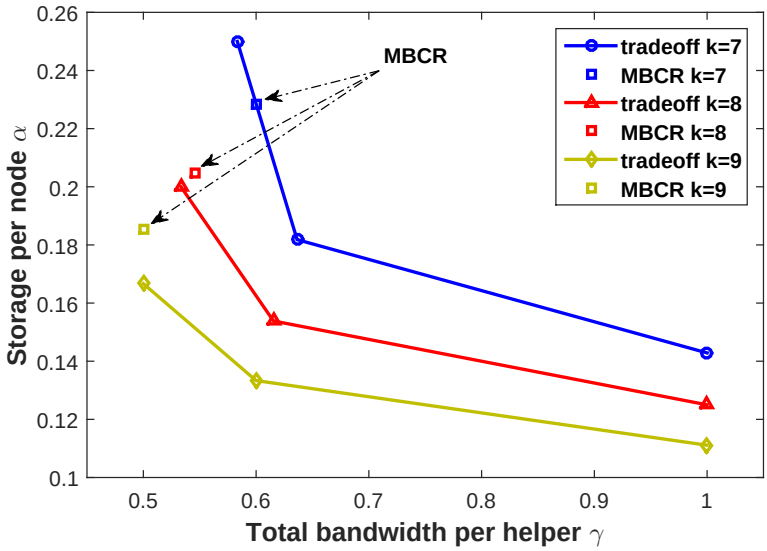


Figure 4.1: Optimal functional repair tradeoffs for fixed $e = 3$ and different $k \in \{7, 8, 9\}$. The MBCR point lies on the tradeoff only in the case of $k = 7$.

chapter, one can check that $\gamma_{\text{MBMR}} < \gamma_{\text{MBCR}} < f_r(0)$. It follows that the optimal storage size corresponding to γ_{MBCR} and achieving the centralized functional repair tradeoff is given by

$$\begin{aligned}\alpha^*(\gamma_{\text{MBCR}}) &= \frac{\mathcal{M} - \gamma_{\text{MBCR}} g_r(0)}{r} = \frac{\mathcal{M}(2d+e-r)}{k(2d-k+e)} \\ &= \alpha_{\text{MBCR}} - \frac{\mathcal{M}(r-1)}{k(2d-k+e)}.\end{aligned}$$

Therefore, $\alpha^*(\gamma_{\text{MBCR}}) \leq \alpha_{\text{MBCR}}$ with equality if and only if $r = 1$. □

Figure 4.1 illustrates the functional tradeoff for fixed $e = 3, \mathcal{M} = 1$ and multiple values of $k \in \{7, 8, 9\}$ such that $d = k$. As proved in Theorem 4.1, MBCR codes are optimal centralized repair codes only when $r = 1$, which corresponds to $k = 7$ in Figure 4.1. When $e \mid k$, MBCR codes achieve the same bandwidth as MBMR codes, but have a higher storage cost.

Remark 4.1. *Theorem 4.1 proves that, when* $k \equiv 1 \mod e$, *MBCR codes achieve an interior point on the functional tradeoff that lies near the MBMR point. We note that the existence*

of this exact-repair interior point does not contradict the infeasibility result of interior points, where we assume $e|k$.

4.3 First construction for interior points

4.3.1 Code construction framework

We first describe the code construction which is an improvement upon [87]. The construction is based on a collection of subsets of $[n]$, called a Steiner system. Information is first encoded within each subset, and then distributed among the n nodes. We recall the definition of Steiner systems.

Definition 4.2. *A Steiner system $S(t, r, n)$, $t \leq r \leq n$, is a collection of subsets of size r, included in $[n]$, such that any subset of $[n]$ of size t appears exactly once across all the subsets.*

The existence of Steiner systems is not known in general when $t < r < n$. But for large n, [89] proved the existence of Steiner systems as long as the parameters t, r, n satisfy certain divisibility conditions. When $t = r$, Steiner systems always exist, and the subsets in this case are all r-combinations of the set $[n]$. We call this case trivial Steiner systems. The family of $(\mathcal{M}, n, k, d, e, \alpha, \beta)$ codes we describe below is parameterized by t, m, r, for $e \leq m < r \leq n, t \leq r$, where

$$\mathcal{M} = N(r-m), N = \frac{\binom{n}{t}}{\binom{r}{t}}, \alpha = \frac{\binom{n-1}{t-1}}{\binom{r-1}{t-1}}, k = n - m. \tag{4.6}$$

Note here that N is the number of subsets in the Steiner system $S(t, r, n)$, and the integrality of N and α follows from properties of Steiner systems [90].

Construction 4.1. ***Precoding step:*** *We consider a Steiner system $S(t, r, n)$ and generate*

$N = \frac{\binom{n}{t}}{\binom{r}{t}}$ *subsets, also referred to as blocks, such that each block is indexed by a set* $J \in S(t, r, n)$. *Block* J *corresponds to* $r - m$ *information symbols over an alphabet of size* q, *which is then encoded using a universal MSMR code with length* r *and dimension* $r - m$ *over an alphabet of size* q. *The MSMR codeword symbols, called the repair group* J, *is comprised of* $\{c_{x,J} : x \in J\}$. *The total number of information symbols is* $\mathcal{M} = N(r - m)$, *each symbol being of size* $\log_2 q$ *bits.*

The code matrix: *The code structure can be described by a code matrix* C, *of size* $n \times N$. *The rows of* C *are indexed by integers in* $[n]$, *corresponding to the different storage nodes, and its columns are indexed by sets in* $S(t, r, n)$, *arranged in some arbitrary chosen order. We formally define* C *as*

$$C_{x,J} = \begin{cases} c_{x,J}, & \text{if } x \in J, \\ -, & \text{otherwise}, \end{cases} \tag{4.7}$$

where "$-$*" denotes an empty symbol. Node* $i \in [n]$ *stores all the non-empty symbols of row* i *in the code matrix* C. *It can be checked that the storage per node is given by* $\alpha = \frac{Nr}{n} = \frac{\binom{n-1}{t-1}}{\binom{r-1}{t-1}}$.

By abuse of notation, the terms block and repair group are used interchangeably. For each block, the universal MSMR code possesses the optimal repair bandwidth (4.3) for any number of erasures l, $1 \leq l \leq m$, and any number of helpers d, $r - m \leq d \leq r - l$. The requirement on the alphabet size q is dictated by the existence of a universal MSMR code. As discussed in the introduction, such MSMR codes are known to exist.

Example 4.1. *Consider a Steiner system* $S(t, r, n) = S(3, 4, 8)$. *So the number of blocks is*

$N = 14$ and each node number appears in $\alpha = \frac{rN}{n} = 7$ blocks. The 14 blocks are given by

$$J_1 = \{1,2,4,8\}, J_2 = \{2,3,5,8\}, J_3 = \{3,4,6,8\}, J_4 = \{4,5,7,8\}, J_5 = \{1,5,6,8\},$$
$$J_6 = \{2,6,7,8\}, J_7 = \{1,3,7,8\}, J_8 = \{3,5,6,7\}, J_9 = \{1,4,6,7\}, J_{10} = \{1,2,5,7\},$$
$$J_{11} = \{1,2,3,6\}, J_{12} = \{2,3,4,7\}, J_{13} = \{1,3,4,5\}, J_{14} = \{2,4,5,6\}.$$

The code matrix is given by (4.8).

$$C = \begin{bmatrix} c_{1,J_1} & - & - & - & c_{1,J_5} & - & c_{1,J_7} & - & c_{1,J_9} & c_{1,J_{10}} & c_{1,J_{11}} & - & c_{1,J_{13}} & - \\ c_{2,J_1} & c_{2,J_2} & - & - & - & c_{2,J_6} & - & - & - & c_{2,J_{10}} & c_{2,J_{11}} & c_{2,J_{12}} & - & c_{2,J_{14}} \\ - & c_{3,J_2} & c_{3,J_3} & - & - & - & c_{3,J_7} & c_{3,J_8} & - & - & c_{3,J_{11}} & c_{3,J_{12}} & c_{3,J_{13}} & - \\ c_{4,J_1} & - & c_{4,J_3} & c_{4,J_4} & - & - & - & - & c_{4,J_9} & - & - & c_{4,J_{12}} & c_{4,J_{13}} & c_{4,J_{14}} \\ - & c_{5,J_2} & - & c_{5,J_4} & c_{5,J_5} & - & - & c_{5,J_8} & - & c_{5,J_{10}} & - & - & c_{5,J_{13}} & c_{5,J_{14}} \\ - & - & c_{6,J_3} & - & c_{6,J_5} & c_{6,J_6} & - & c_{6,J_8} & c_{6,J_9} & - & c_{6,J_{11}} & - & - & c_{6,J_{14}} \\ - & - & - & c_{7,J_4} & - & c_{7,J_6} & c_{7,J_7} & c_{7,J_8} & c_{7,J_9} & c_{7,J_{10}} & - & c_{7,J_{12}} & - & - \\ c_{8,J_1} & c_{8,J_2} & c_{8,J_3} & c_{8,J_4} & c_{8,J_5} & c_{8,J_6} & c_{8,J_7} & - & - & - & - & - & - & - \end{bmatrix}. \tag{4.8}$$

Let $m = 2, e = 2, d = n - e = 6$. For each block, we use the EVENODD code in Table 1.1, so every symbol $c_{x,J} \in \mathbb{F}_2^2$, $q = 4$. Then we can repair nodes 1 and 2 simultaneously, by downloading

- *symbols c_{4,J_1}, c_{8,J_1} from nodes 4 and 8, respectively. These help repair symbols c_{1,J_1} and c_{2,J_1},*

- *symbols $c_{5,J_{10}}, c_{7,J_{10}}$ from nodes 5 and 7, respectively. These help repair symbols $c_{1,J_{10}}$ and $c_{2,J_{10}}$,*

- *symbols* $c_{3,J_{11}}, c_{6,J_{11}}$ *from nodes 3 and 6, respectively. These help repair symbols* $c_{1,J_{11}}$ *and* $c_{2,J_{11}}$,

- $\frac{1}{2}$ *symbol from each of the nodes* $5, 6$ *and* 8*, to repair* c_{1,J_5},

- $\frac{1}{2}$ *symbol from each of the nodes* $3, 7$ *and* 8*, to repair* c_{1,J_7},

- $\frac{1}{2}$ *symbol from each of the nodes* $4, 6$ *and* 7*, to repair* c_{1,J_9},

- $\frac{1}{2}$ *symbol from each of the nodes* $3, 4$ *and* 5*, to repair* $c_{1,J_{13}}$,

- *and similarly for node 2 to repair* $c_{2,J_2}, c_{2,J_6}, c_{2,J_{12}}$ *and* $c_{2,J_{14}}$.

In total, we download 18 *symbols. Each helper transmits* 3 *symbols.*

From the example above, we see that each repair group J tolerates the failure of m nodes. Therefore, the code C also tolerates the failure of up to any m nodes. Thus, it can be checked that for Construction 4.1 from any $k = n - m$ nodes, we can recover the data, which is the *reconstruction parameter*. Namely, the code can recover from any m failures.

Therefore, it is possible to repair any e failures, for a flexible number of failures such that $1 \leq e \leq m$. The number of helpers is flexible, and satisfies $k \leq d \leq n - e$. In other words, Construction 4.1 possesses the *universality* of repair for flexible e and d. The repair bandwidth is given in Propositions 4.1 and 4.2 for two different scenarios in the next two subsections.

4.3.2 Code construction with trivial Steiner systems

Proposition 4.1. *Using Construction 4.1 with* $t = r$*, it is possible to repair simultaneously any set of* $1 \leq e \leq m$ *nodes, using* $n - m \leq d \leq n - e$ *helpers, such that the contribution of*

each helper, denoted by $\beta_e(d)$, is given by

$$\beta_e(d) = \sum_{s=1}^{e} \binom{e}{s} \sum_{p=\max(s,r-d)}^{\min(n-d-e+s,r-1)} \binom{d-1}{r-p-1} \binom{n-d-e}{p-s} \frac{s}{m-p+s}. \tag{4.9}$$

Proof. In the repair procedure, any subset of missing symbols belonging to the same repair group is repaired via MSMR repair procedure, using *all* available helpers from the same group among the chosen helper nodes. Fixing the set of helper nodes, we argue that the repair is feasible. Indeed, let H be the set of d helpers. For each repair group J, we denote the set of remaining nodes in J as $J^{'}$. Using $|H \cup J^{'}| \leq n - e$ and $d \geq k = n - m$, it follows that

$$|J^{'} \cap H| = |H| + |J^{'}| - |H \cup J^{'}| \geq d + r - e - (n - e) = r + d - n \geq r - m.$$

Thus, for each repair group, we have enough information across the set of helpers to recover the missing components.

We now analyze the contribution of a single helper h: h helps in the simultaneous repair of s missing symbols of the same repair group, such that $1 \leq s \leq e$. For each size s, we count all possible cases in which the repair can be done through the help of $r-p$ coded symbols among all the d helpers, because the number of available coded symbols determines the contribution of each helper, as dictated by the MSMR repair bandwidth (4.3). It follows that, for the corresponding repair group, $r - p - 1$ can be chosen from the set of $d - 1$ helpers (helper h already belongs to the repair group by assumption), while the remaining $p - s$ elements of the repair group can be chosen from the remaining $n - e - d$ nodes. Figure 4.2 summarizes the repair situation for given parameters s and p. Summing over all repair contributions, and analyzing the limit cases of p for a given s, (4.9) follows. □

Example 4.2. *Consider a Steiner system $S(t, r, n) = S(4, 4, 5)$. So the number of blocks is $N = 5$ and each node stores $\alpha = \frac{rN}{n} = 5$ symbols. The 5 blocks are given by $J_1 =$*

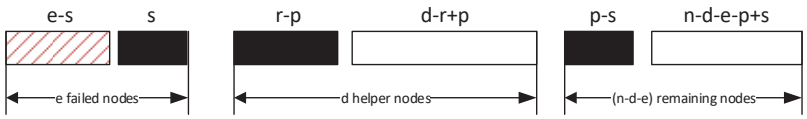


Figure 4.2: A repair situation associated with given parameters s and p.

$\{2,3,4,5\}, J_2 = \{1,3,4,5\}, J_3 = \{1,2,4,5\}, J_4 = \{1,2,3,5\}, J_5 = \{1,2,3,4\}$. *The code matrix is given by (4.10).*

$$C = \begin{bmatrix} - & c_{1,J_2} & c_{1,J_3} & c_{1,J_4} & c_{1,J_5} \\ c_{2,J_1} & - & c_{2,J_3} & c_{2,J_4} & c_{2,J_5} \\ c_{3,J_1} & c_{3,J_2} & - & c_{3,J_4} & c_{3,J_5} \\ c_{4,J_1} & c_{4,J_2} & c_{4,J_3} & - & c_{4,J_5} \\ c_{5,J_1} & c_{5,J_2} & c_{5,J_3} & c_{5,J_4} & - \end{bmatrix}. \tag{4.10}$$

Let $m = 2, e = 1, d = 3$. Then, $k = 2$. Consider the repair of node 1 with the helper nodes $2,3,4$. We download: 1 symbol from each of the nodes 3 and 4 to repair c_{1,J_2}; 1 symbol from each of the nodes 2 and 4 to repair c_{1,J_3}; 1 symbol from each of the nodes 2 and 3 to repair c_{1,J_4}; $\frac{1}{2}$ symbol from each of the nodes $2,3$ and 4 to repair c_{1,J_5}. Each helper transmits $\frac{5}{2}$ symbols, as given by (4.9). The first three cases correspond to $p = 2$, and the last case corresponds to $p = 1$.

Remark 4.2. *It can be seen that the repair procedure can benefit from the MSMR repair property in the case $n > k + 1$. In particular, the advantages of using MSMR codes in our construction over maximum distance separable (MDS) codes as in [87] are: 1) lower repair bandwidth, 2) symmetric repair among helper nodes, which obviates the need for the expensive procedure of duplicating the block design in [87], and 3) universality, meaning non-trivial repair strategies for multiple erasures, $1 \leq e \leq m$ with the help of varying number of helpers d, such that $n - m \leq d \leq n - e$. Figure 4.3 shows a comparison between the performance of the layered code in [87] and Construction 4.1, for an $(n, k, d, e) = (10, 7, 7, 1)$*

system. The MSMR repair property clearly helps reduce the bandwidth.

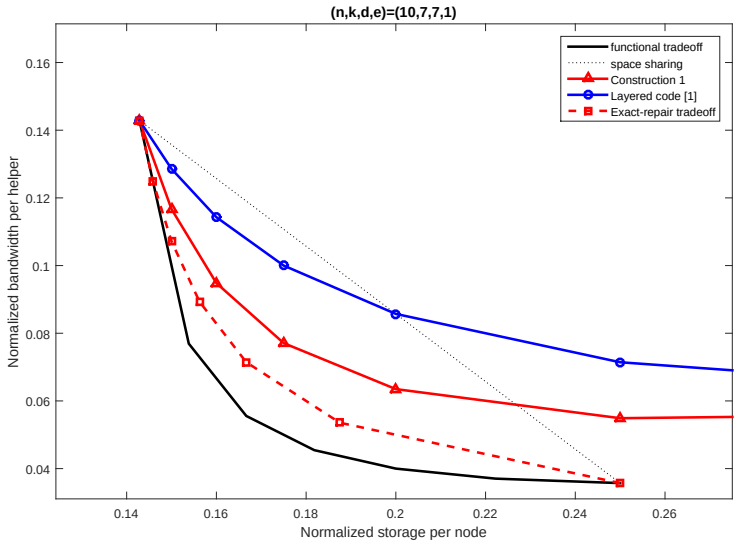


Figure 4.3: Using the MSMR repair property improves upon the layered code repair performance. The exact-repair tradeoff is achieved in [4], with lower bound derived in [5].

Example 4.3. *We note that for large n and $n-k$, the complexity of designing universal MSMR may lead to large subpacketization size α. However, for small n and $n-k$, universal MSMR can be competitive and may not entail additional overhead, compared to using MDS codes as in [87].*

Consider $(n,k,d) = (5,3,3), (r,m) = (4,2)$. We compare the performance achieved by [87] and Construction 4.1, considering the repair of a single failure.

- *The construction in [87, Figure 3] uses an MDS code with dimension 2 and length 4. Assume the MDS code is defined over a field of size 4, then, every codeword symbol of the MDS code corresponds to 2 bits. Then, the code generated in [87, Figure 3] achieves the following: $\mathcal{M} = 60$ bits and $\alpha = 24$ bits. In the repair process, a replacement node downloads 48 bits; each helper transmits 16 bits. The code achieves $(\bar{\alpha}, \bar{\beta}) = (\frac{2}{5}, \frac{4}{5})$.*

• *In Construction 4.1, assume we use the EVENODD code as the underlying MSR code. Then,* $\mathcal{M} = 20$ *bits,* $\alpha = 8$ *bits. In the repair process, a replacement node downloads* 15 *bits; each helper transmits* 5 *bits. The code achieves* $(\bar{\alpha}, \bar{\beta}) = (\frac{2}{5}, \frac{1}{4})$.

Therefore, for the same $\bar{\alpha} = \frac{2}{5}$*, Construction 4.1 achieves smaller subpacketization size and also strictly smaller normalized repair bandwidth* $\bar{\beta}$.

Remark 4.3. *The technique of using MSR codes as building blocks for outer code constructions has been used in the literature, for instance in constructing codes with local regeneration [91, 92] and exact-repair regenerating codes for single failure [88]. In particular, it can be checked that the construction in [88] uses copies of trivial Steiner systems, and achieves the same normalized points* $(\bar{\alpha}, \bar{\beta})$ *as Construction 4.1. However, the construction in [88] uses higher subpacketization size due to the necessity of code duplication in order to achieve symmetry across all nodes. Moreover, Construction 4.1 allows the use of general balanced incomplete block designs (c.f, Remark 4.6) in order to further reduce the subpacketization size. We note that determinant codes in [4] achieve the same normalized interior points as Construction 4.1 for an* $(k+1, k, k, 1)$ *system. Moreover, unlike Construction 4.1, determinant codes achieve the same set of normalized interior points for any number of nodes* $n \geq k+1$*, which is better than Construction 4.1. The authors in [86] provide a construction that improves upon [93] for certain parameter sets [86, Theorems 3.1, 3.2]. For a numerical example, for a* $(9, 7, 8, 1)$ *system, when* $\bar{\alpha} < 0.1538$*, Construction 4.1 outperforms [86], when* $\bar{\alpha} > 0.1538$*, [86] is better.*

Remark 4.4. *Using (4.3), it can be argued that* $\beta_e(d)$ *in (4.9) is decreasing in* d*. For fixed* n, k, m, e*, the minimum bandwidth per helper is then achieved with* $d = n - e$*, and is given by*

$$\beta_e(n-e) = \sum_{s=1}^{e} \binom{e}{s} \binom{n-e-1}{r-s-1} \frac{s}{m} = \frac{e}{m} \sum_{s=1}^{e} \binom{e-1}{s-1} \binom{n-e-1}{r-s-1} = \frac{e}{m} \binom{n-2}{r-2}, \quad (4.11)$$

where the last equality follows from Vandermonde's identity. Moreover, we obtain after

simplification

$$\mathcal{M} = (r-m)\binom{n}{r}, \bar{\alpha} = \frac{r}{n(r-2)}, \bar{\beta} = \frac{e}{m}\frac{r(r-1)}{n(n-1)(r-2)}. \tag{4.12}$$

We argue in the next example that for some parameters, one can use a regenerating code corresponding to an interior point instead of an MSMR code as the inner code per repair group, and achieve the same performance.

Example 4.4. *Consider the case* $(n,k,d,e) = (5,4,4,1)$. *Let* $r = t = 4, m = 1$ *in (4.6). The code matrix is given by (4.10). Thus, the code per column of* C *is of length* $r = 4$ *and its reconstruction parameter is* $r - m = 3$. *We use the interior code:* $(\bar{\alpha}_0, \bar{\beta}_0) = (\frac{3}{8}, \frac{1}{4})$ *per repair group. Let* F_0 *be the information size per column. Thus,* $\mathcal{M} = 5F_0$ *and* $\alpha = \frac{3F_0}{2}$. *It follows that* $\bar{\alpha} = \frac{3}{10}$. *To repair node 1, we download a total bandwidth of* $3F_0$. *Thus,* $\bar{\beta} = \frac{3}{20}$. *We obtain the achievable point* $(\bar{\alpha}, \bar{\beta}) = (\frac{3}{10}, \frac{3}{20})$. *The same point is equally achievable using Construction 4.1 with* $(t,r,n,m,e) = (3,3,5,1,1)$ *with an MSMR code as the interior code. This point is optimal on the exact-repair tradeoff of the* $(5,4,4,1)$ *system [93, 4], and is the optimal point next to the minimum bandwidth regenerating point.*

4.3.3 Code construction with general Steiner systems

In Proposition 4.1, we considered Construction 4.1 with Steiner systems such that $t = r$. We study next the use of a general Steiner system for the specific $(n, n-m, n-e, e)$ system, such that $1 \leq e \leq m, d = n - e$.

Proposition 4.2. *Construction 4.1 generates an* $(\mathcal{M}, n, n-m, n-e, e, \alpha, \beta)$ *code such, that*

during repair of $e \leq m$ nodes, each helper contributes β_e symbols and we have

$$\mathcal{M} = (r-m)\frac{\binom{n}{t}}{\binom{r}{t}}, \alpha = \frac{\binom{n-1}{t-1}}{\binom{r-1}{t-1}}, \beta_e = \frac{e}{m}\frac{\binom{n-2}{t-2}}{\binom{r-2}{t-2}}, \tag{4.13}$$

$$\bar{\alpha} = \frac{r}{n(r-2)}, \bar{\beta}_e = \frac{e}{m}\frac{r(r-1)}{n(n-1)(r-2)}. \tag{4.14}$$

Proof. We consider a Steiner system $S(t, r, n)$. Recall that $k = n-m$, which implies that the system can tolerate any $e \leq m$ failures. Let $\mathcal{E} = \{e_{i_1}, \ldots, e_{i_e}\}$ denote a set of e failed nodes. The set of helpers $\mathcal{H}$ consists of all the remaining nodes, $\mathcal{H} = [n]\backslash\mathcal{E}$. The repair proceeds by recovering the blocks which contain at least one failed node. Consider a helper $h \in \mathcal{H}$ and a failed node $i_j, j \in [e]$. By basic counting argument, nodes h and i_j share $\lambda_2 \triangleq \frac{\binom{n-2}{t-2}}{\binom{r-2}{t-2}}$ blocks. Let J be one of these λ_2 blocks, and denote by s the number of failed nodes within J. Note that $1 \leq s \leq e$. As the non-failed nodes within J are helpers, the contribution of node h in the recovery of block J follows from (4.3), and is given by $\frac{s}{(r-s)-(r-m)+s} = \frac{s}{m}$ symbols. That is, the average amount of information h contributes to the repair of i_j is $\frac{1}{m}$. Thus, node h contributes $\frac{\lambda_2}{m}$ symbols in the repair of node i_j across all blocks containing (h, i_j). Summing over all failed nodes, (4.13) follows. Using (4.6) and (4.13), (4.14) follows after simplification. □

The repair in Example 4.1 is an illustration of Proposition 4.2. Similar to Proposition 4.1, the repair bandwidth is identical among the helper nodes, and independent of the choice of the failed nodes and helpers. One can observe that for each helper, it transmits 1 symbol from 1 block, and two half symbols from 2 blocks. In particular, for helper h and failure nodes $1, 2$, the number of blocks containing any subset of $\{h, 1, 2\}$ is independent of the choice of h. Therefore, we observe this symmetry in the repair. In fact, such symmetry occurs when $t \geq e+1$ and is due to the fact that every subset of t appears exactly once in the Steiner system.

Remark 4.5. *We note here that $\bar{\alpha}, \bar{\beta}$ do not depend on t by (4.14). The advantage of using*

Steiner systems with smaller t, whenever they exist, is that they induce smaller α and β, for the same normalized parameters. Indeed, it can be shown that α, as given by (4.14), is strictly increasing in t. Therefore, to reduce the storage size per node, and therefore the repair bandwidth, it is advantageous to use a Steiner System with the smallest t, $t \leq r$. Moreover, when $d = n - e, t = r$, Proposition 4.1 and Proposition 4.2 give the same $\bar{\alpha}, \bar{\beta}$ (cf. Remark 4.4). Finally, the value of $\bar{\beta}$ for general Steiner systems and arbitrary $d > n - e$ is an open problem. We conjecture that the normalized bandwidth should be identical to that of Proposition 4.1.

Example 4.5. Consider a Steiner system $S(t, r, n) = S(2, 4, 13)$. Then, $N = 13, \alpha = 4$. The blocks are given by

$$J_1 = \{1, 2, 4, 10\}, J_2 = \{2, 3, 5, 11\}, J_3 = \{3, 4, 6, 12\}, J_4 = \{4, 5, 7, 13\}, J_5 = \{1, 5, 6, 8\},$$
$$J_6 = \{2, 6, 7, 9\}, J_7 = \{3, 7, 8, 10\}, J_8 = \{4, 8, 9, 11\}, J_9 = \{5, 9, 10, 12\},$$
$$J_{10} = \{6, 10, 11, 13\}, J_{11} = \{1, 7, 11, 12\}, J_{12} = \{2, 8, 12, 13\}, J_{13} = \{1, 3, 9, 13\}.$$

Let $m = 2, e = 2, d = n - e = 11$. We use the EVENODD code for each block. Nodes 1 and 2 can be repaired simultaneously by downloading

- symbols c_{4,J_1}, c_{10,J_1} from nodes 4 and 10, respectively. These will help repair symbols c_{1,J_1} and c_{2,J_1}.
- $\frac{1}{2}$ symbol from each of the nodes $5, 6$ and 8, to repair c_{1,J_5}.
- $\frac{1}{2}$ symbol from each of the nodes $7, 11$ and 12, to repair $c_{1,J_{11}}$.
- $\frac{1}{2}$ symbol from each of the nodes $3, 9$ and 13, to repair $c_{1,J_{13}}$.
- $\frac{1}{2}$ symbol from each of the nodes $3, 5$ and 11, to repair c_{2,J_2}.
- $\frac{1}{2}$ symbol from each of the nodes $6, 7$ and 9, to repair c_{2,J_6}.

- $\frac{1}{2}$ symbol from each of the nodes $8, 12$ and 13, to repair $c_{2,J_{12}}$.

While each helper transmits one symbol, the nature of the repair is different across helpers. For instance, node 4 transmits an entire symbol that contributes to the recovery of block J_1, while node 5 transmits two half symbols that contribute to the recovery of blocks J_2 and J_5, respectively. This scenario happens because $t \leq e$, which implies that some helpers may share a block with the set of e failed nodes, while other helpers may not. In all scenarios, the repair bandwidth is identical across helpers, as shown in Proposition 4.2.

Remark 4.6. *Construction 4.1 can be extended to general balanced incomplete block design (BIBD) systems. A BIBD system $BIBD(t, r, n, \lambda)$, $t \leq r \leq n$, is a collection of subsets of size r, included in $[n]$, such that any subset of $[n]$ of size t appears exactly λ times across all the subsets. In particular, a Steiner system is a BIBD system with $\lambda = 1$. When $t = r$, a BIBD system can simply be obtained by duplicating a Steiner system $S(t, r, n)$ λ times. Therefore, in this case we restrict our attention to Steiner systems as they provide smaller storage cost, for the same parameters. Moreover, it can be seen that the proof of Proposition 2 holds for any BIBD system, with the only difference being in the number of blocks any two nodes share. For instance, for a $BIBD(t, r, n, \lambda)$ system, any two nodes share $\lambda\frac{\binom{n-2}{t-2}}{\binom{r-2}{t-2}}$ blocks [90] and we obtain*

$$\mathcal{M} = \lambda(r-m)\frac{\binom{n}{t}}{\binom{r}{t}}, \alpha = \lambda\frac{\binom{n-1}{t-1}}{\binom{r-1}{t-1}}, \beta_e = \lambda\frac{e}{m}\frac{\binom{n-2}{t-2}}{\binom{r-2}{t-2}}, \tag{4.15}$$

$$\bar{\alpha} = \frac{r}{n(r-2)}, \bar{\beta}_e = \frac{e}{m}\frac{r(r-1)}{n(n-1)(r-2)}. \tag{4.16}$$

Therefore, when designing the code, for fixed parameters n, r, one can optimize over existing block designs and chooses the block design that results in the smallest $\alpha = \lambda\frac{\binom{n-1}{t-1}}{\binom{r-1}{t-1}}$. Recall that the trivial design always exists and satisfies $\lambda = 1, t = r$.

4.4 Analysis of the achievability for a $(k+e, k, k, e)$ system

In this section, we analyze the achievable region for an $(n, k, d, e) = (k+e, k, k, e)$ system by means of Construction 4.1, using a Steiner system $S(t, r, k+e)$ with the choice of $t \leq r$ as indicated in Remark 4.5.

Corollary 4.1. *Construction 4.1 with $m = e$ generates a set of achievable points for an $(\mathcal{M}, k+e, k, k, e, \alpha, \beta)$ system, such that*

$$F = (r-e)\frac{\binom{k+e}{t}}{\binom{r}{t}}, \alpha = \frac{\binom{k+e-1}{t-1}}{\binom{r-1}{t-1}}, \beta = \frac{\binom{k+e-2}{t-2}}{\binom{r-1}{t-1}}, \tag{4.17}$$

$$\bar{\alpha} = \frac{r}{(k+e)(r-e)}, \bar{\beta} = \frac{r(r-1)}{(k+e)(k+e-1)(r-e)}, \quad e+1 \leq r \leq k+e. \tag{4.18}$$

Proof. The proof follows from Proposition 4.2. □

4.4.1 Optimality of one achievable point

Theorem 4.2. *For the $(k+e, k, k, e)$ system, the point achieved in (4.18) for $r = k+e-1$ is an optimal interior point.*

Proof. From (4.17) when $r = k+e-1$, we obtain

$$(\bar{\alpha}, \bar{\beta}) = (\frac{k+e-1}{(k+e)(k-1)}, \frac{k+e-2}{(k+e)(k-1)}). \tag{4.19}$$

Substituting (4.19) in (4.2) and setting $p = g-2$, we obtain

$$(t+ge-2e)\bar{\alpha} + (k-t-ge+2e)\bar{\beta} = (k-e)\bar{\alpha} + e\bar{\beta} = 1.$$

Therefore, the above point lies on the functional repair lower bound and hence is optimal. It lies on the first segment of the bound near the MSMR point, and it is not the MSMR point nor the MBCR point, as indicated by (4.3) and (4.5). □

We note that the optimality of the point in Theorem 4.2 for the case of $(k+1, k, k, 1)$ was also shown in [87, Proposition 3]. Figure 4.4 illustrates the optimality of the point achieved by Theorem 4.2 for two different systems. The achievable point, corresponding to $(r, m) = (n-1, e)$, lies on the functional tradeoff, which proves also its optimality under exact repair.

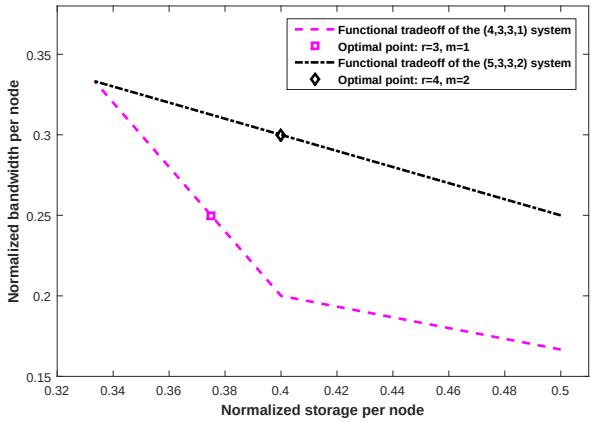


Figure 4.4: Functional tradeoff and optimal points achieved by Theorem 4.2.

4.4.2 Optimal extension property

From Theorem 4.2, Construction 4.1 gives us an optimal point for any $(k+e, k, k, e)$ system. Construction 4.1 also offers the following optimal extension property.

Proposition 4.3. *Consider a $(k+e, k, k, e)$ system and consider the optimal point achieved by Construction 4.1 in Theorem 4.2 with $t = r$, one can extend the system to a $(k+e+$*

$1, k, k, e+1)$ *system, operating at the optimal point of Theorem 4.2, by adding another node to the system and increasing the storage per node, while keeping the initial storage content.*

Proof. Let α_i, β_i, F_i, for $i = 1, 2$, refer to the parameters of the old and the new systems, respectively. Then, $\alpha_2 - \alpha_1 = 1, \beta_2 - \beta_1 = 1, F_2 - F_1 = k - 1$. Moreover, the number of blocks N is increased by 1. Let $k + e + 1$ be the index of the new node to be added. The new code is obtained by simply adding another block, whose set is $\{1, \ldots, k+e\}$, and adding to each of the old sets the element $(k + e + 1)$, and thus generating another coded symbol for the corresponding repair group. Note here that the dimension of the MSMR code in each block does not change. A key requirement is to assume the use of an MSMR code that can accommodate the addition of extra coded symbols, when needed. This can be done by choosing the number of symbols of the MSMR code to be as large as needed (this may result in an increase in the underlying field size). Each old node will store an extra symbol coming from the new repair group, while the new node stores the newly generated coded symbols from the old repair groups. □

Example 4.6. *We illustrate the process of extending a* $(4, 3, 3, 1)$ *system to a* $(5, 3, 3, 2)$ *system. Initially, each repair group is of size 3. The code blocks are given by*

$$J_1 = \{2, 3, 4\}, J_2 = \{1, 3, 4\}, J_3 = \{1, 2, 4\}, J_4 = \{1, 2, 3\}.$$

The code matrix is given by

$$C_1 = \begin{bmatrix} - & c_{1,J_2} & c_{1,J_3} & c_{1,J_4} \\ c_{2,J_1} & - & c_{2,J_3} & c_{2,J_4} \\ c_{3,J_1} & c_{3,J_2} & - & c_{3,J_4} \\ c_{4,J_1} & c_{4,J_2} & c_{4,J_3} & - \end{bmatrix}.$$

Adding node 5 to the system, we add another block $J_5 = \{1, 2, 3, 4\}$*, whose symbols will be*

distributed across the old nodes $\{1, 2, 3, 4\}$. *The old blocks become*

$$J_1 = \{2, 3, 4, 5\}, J_2 = \{1, 3, 4, 5\}, J_3 = \{1, 2, 4, 5\}, J_4 = \{1, 2, 3, 5\}.$$

The new node 5 *stores newly generated coded symbols of each of the old repair groups* $\{J_1, \ldots, J_4\}$. *The new code matrix is given by (4.10). The two points corresponding to this example are illustrated in Figure 4.4.*

The above property is useful for systems for which the fault tolerance may be deemed insufficient. Therefore, one can increase the fault tolerance of the system without sacrificing the optimality on the exact-repair tradeoff, or changing the existing data. We note also that by a successive application of Proposition 4.3, we can increase the fault tolerance of the system by any desirable factor.

4.4.3 Acheivability region for the $(k + e, k, k, e)$ system

In this subsection, we seek to determine the convex hull of the known achievable points for the $(k + e, k, k, e)$ system, which corresponds to the best known achievable points. The convex hull, denoted by $\mathcal{R}$, is the smallest convex set containing all known achievable points, obtained by all convex combinations (i.e., space-sharing) among the points achieved by Construction 4.1, described in (4.18), and also the MBCR point given by (4.5). Therefore, the objective is to determine which points are sufficient to describe $\mathcal{R}$. We refer to these points as *corner points* of $\mathcal{R}$.

Figure 4.5 presents the achievable points for a $(17, 14, 14, 3)$ system. The achievable points of (4.17) are parameterized by r, such that $e + 1 \leq r \leq e + k$. For each r, we denote the corresponding point as $(\bar{\alpha}_r, \bar{\beta}_r)$. As r decreases, the storage α_r increases. By abuse of notation, we refer to the point $(\bar{\alpha}_r, \bar{\beta}_r)$ as point r. We state some guiding observations for

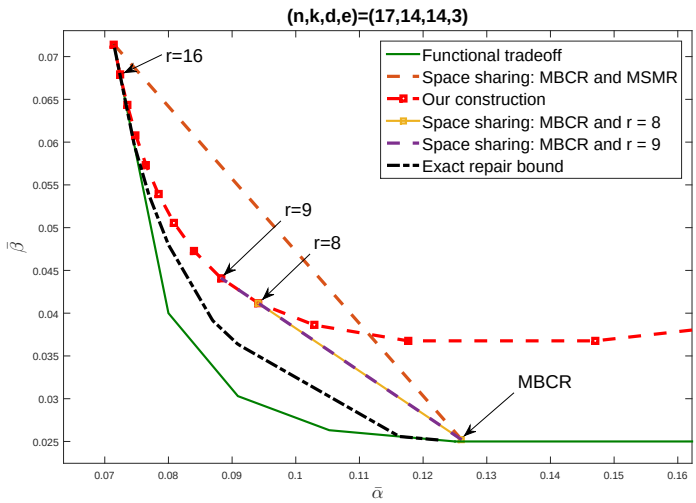


Figure 4.5: Achievable points by Construction 4.1 for a $(n, k, d, e) = (17, 14, 14, 3)$ system. The x-axis is $\bar{\alpha}$ and the y-axis is $\bar{\beta}$.

our subsequent analysis. First, one can eliminate some of the achievable points obtained by Construction 4.1. For instance, point $r = 5$, with $\bar{\alpha} = 0.1471$, achieves a similar bandwidth as its neighbor point $r = 6$, but a larger storage size. Points to the right of $\bar{\alpha} = 0.1471$, such that $r < 5$, can be also immediately eliminated, because they can be outperformed by space-sharing between the MBCR point and some interior point. Interestingly, we observe that point $r = 8$ lies exactly on the segment joining point $r = 9$ and the MBCR point. This means that, while point $r = 8$ is not outperformed by space-sharing, it is nonetheless not necessary for the description of $\mathcal{R}$, and thus it is not considered as a corner point. In the following, we show that the observations from Figure 4.5 can be generalized and we explicitly determine the corner points of $\mathcal{R}$, depending on the system parameters e and k. Our characterization of the corner points is presented in Propositions 4.4 and 4.5. In order to prove them, we first prove some facts about the corner points in Lemmas 4.1 to 4.4.

Lemma 4.1. *The achievable points in (4.17), with $r < 2e$, are not corner points in $\mathcal{R}$.*

Proof. From (4.17), it can be seen that $\bar{\alpha}(r)$, seen as a function of r, is decreasing. $\bar{\beta}(r)$ is a fractional function in r, with a pole at $r = e$. For $r > e$, $\bar{\beta}(r)$ is convex in r. It

can be shown that it decreases and then increases monotonically. Therefore, as $\bar{\alpha}(r)$ is decreasing, the points of interest are those for which $\bar{\beta}(r)$ increases. Moreover, by noticing that $\bar{\beta}(2e) = \bar{\beta}(2e-1)$, it follows that points with $r \leq 2e-1$ do not contribute to the acheivability region $(\bar{\alpha}, \bar{\beta})$ as they are outperformed by the point $r = 2e$ in terms of both, storage and bandwidth. □

Lemma 4.1 implies that it is sufficient to consider the range $2e \leq r \leq k+e$. We define the non-negative integer p such that $r = 2e + p$. We now show that the achievable points $r = 2e, \ldots, k+e$ can not be eliminated by space-sharing between themselves, when not considering the MBCR point.

Lemma 4.2. *The achievability region of the points $(\bar{\alpha}_r, \bar{\beta}_r), r = e+1, \ldots, k+e$ has points with $r \in \{2e, \ldots, k+e\}$ as corner points, when not considering the MBCR point.*

Proof. By virtue of Lemma 4.1, points with $e+1 \leq r < 2e$ can be eliminated. We consider the segment joining the points $(\bar{\alpha}_r, \bar{\beta}_r)$ and $(\bar{\alpha}_{r+1}, \bar{\beta}_{r+1})$. The slope of the segment, denoted $sl(r)$, is given by

$$sl(r) = \frac{\bar{\beta}_{r+1} - \bar{\beta}_r}{\bar{\alpha}_{r+1} - \bar{\alpha}_r} = \frac{-r(r-2e+1)}{e(k+e-1)}.$$

The slope $sl(r)$ is strictly decreasing in r for $r \geq 2e$. This means, for any three consecutive points $(\bar{\alpha}_{r+2}, \bar{\beta}_{r+2})$, $(\bar{\alpha}_{r+1}, \bar{\beta}_{r+1})$ and $(\bar{\alpha}_r, \bar{\beta}_r)$, the point $(\bar{\alpha}_{r+1}, \bar{\beta}_{r+1})$ lies below the segment joining the other two extreme points. Therefore, space-sharing between $(\bar{\alpha}_{r+2}, \bar{\beta}_{r+2})$ and $(\bar{\alpha}_r, \bar{\beta}_r)$ is suboptimal. □

Now, we analyze the achievability region when adjoining the MBCR point to the points in (4.17) with $2e \leq r \leq k+e$.

Lemma 4.3. *The MBCR point is a corner point for $\mathcal{R}$.*

Proof. Noting that $\bar{\alpha}_{\text{MBCR}} = \frac{2k+e-1}{(k+e)k} > \bar{\alpha}_{2e} = \frac{2}{(k+e)}$ and $\bar{\beta}_{\text{MBCR}} = \frac{2e}{(k+e)k} < \bar{\beta}_{2e} = \frac{2(2e-1)}{(k+e)(k+e-1)}$, along with Lemma 4.2 concludes the result. □

By Lemma 4.3, we only need to analyze whether space-sharing between the MBCR point and any other point r may outperform some of the other achievable points $r^{'}$.

Lemma 4.4. *If a point r, $r \geq 2e$, is not outperformed by space-sharing between the point $r+1$ and the MBCR point, then, all points r' such that $r' \geq r$, are corner points of the achievability region.*

Proof. The assumption of the lemma implies that the slope of the segment joining the points $(\bar{\alpha}_r, \bar{\beta}_r)$ and $(\bar{\alpha}_{\text{MBCR}}, \bar{\beta}_{\text{MBCR}})$ is smaller than the slope of the segment between $(\bar{\alpha}_{r+1}, \bar{\beta}_{r+1})$ and $(\bar{\alpha}_{\text{MBCR}}, \bar{\beta}_{\text{MBCR}})$. As from Lemma 4.2, the slope of the segment between $(\bar{\alpha}_r, \bar{\beta}_r)$ and $(\bar{\alpha}_{r+1}, \bar{\beta}_{r+1})$ is decreasing in r, it follows that no point $r' \geq r$ can be outperformed by space-sharing across any two other achievable points, including the MBCR point. □

Therefore, to determine the corner points of $\mathcal{R}$, we need to successively test for increasing values of p, such that $0 \leq p \leq k-e$, whether the point $r = 2e+p$ is outperformed by space-sharing of MBCR and point $r+1$. Let p^* denote the smallest p such that $r = 2e+p$ is not outperformed by space-sharing, it follows by Lemma 4.4 the following achievability region.

Proposition 4.4. *The achievability region $\mathcal{R}$ is given by the corner points*

$$\{(\bar{\alpha}_r, \bar{\beta}_r) : r \in \{r : r = 2e+p \text{ and } p^* \leq p \leq k-e\}\} \cup \{MBCR\}, \tag{4.20}$$

where $1 \leq p^* \leq k-e$, *and* p^* *is given by*

$$p^* = \left\lfloor \frac{e-k-2e^2+1+\sqrt{\Delta}}{2(e+k-1)} \right\rfloor + 1,$$

$$\Delta = (2e^2-e+k-1)^2 + 8(k+e-1)e(e-1)(k-e-1). \tag{4.21}$$

Proof. Consider $r = 2e+p, 0 \leq p \leq k-e-1$. We consider space-sharing between the MBCR point and the point $r+1$. We compute the normalized bandwidth, denoted by $\bar{\beta'}_r$, achieved by the considered space-sharing, at the intermediate point $\alpha = \alpha_r$, and then determine whether $\bar{\beta'}_r > \bar{\beta}_r$. Using (4.18) and (4.5), we obtain after simplification

$$\bar{\beta'}_r - \bar{\beta}_r = \frac{k(-2e^2+2e+p^2+p) - p(-2e^2+e+1) + 2e(e^2-1) + p^2(e-1)}{(e+k)(e+p)(e+k-1)(e^2+pe+k-p+kp-1)} \triangleq \frac{N_1(k)}{D} \tag{4.22}$$

$$= \frac{(k+e-1)p^2 + p(2e^2+k-e-1) + 2e(e-1)(e+1-k)}{(e+k)(e+p)(e+k-1)(e^2+pe+k-p+kp-1)} \triangleq \frac{N_2(p)}{D}. \tag{4.23}$$

We regard N_1 as a function of k, for fixed e and p, and N_2 as a function of p, for fixed e and k. In this proof, we are interested in analyzing N_2. We analyze N_1 in a later proof.

Clearly $D > 0$. Thus, $\text{sign}(\bar{\beta'}_r - \bar{\beta}_r) = \text{sign}(N_2(p))$. Therefore, it suffices to study the sign of $N_2(p)$. We note that $\bar{\beta'}_r - \bar{\beta}_r \leq 0$ implies that point $r = 2e+p$ can be eliminated by space-sharing and thus it is a not a corner point. $N_2(p)$ is a quadratic function in p. Let Δ denote the discriminant of $N_2(p)$. It can be checked that

$$\Delta = (2e^2-e+k-1)^2 + 8(k+e-1)e(e-1)(k-e-1) > 0.$$

Thus, there exists $p_{0,1}, p_{0,2}$ such that $N_2(p_{0,1}) = N_2(p_{0,2}) = 0$. As the leading coefficient of $N_2(p)$ is positive, and $N_2(0) = -2e(e-1)(k-e-1) \leq 0$, it follows that one solution, say $p_{0,1}$, is negative and the other solution $p_{0,2}$ is non-negative. That is, $p_{0,1} < 0$ and $p_{0,2} \geq 0$. Then, it follows that $\forall 0 \leq p \leq p_{0,2}, N_2(p) \leq 0$, which implies that the set $\{p : p \leq p_{0,2}\}$

can be eliminated. In particular, $p = 0$ is always eliminated. Let $p^* = \lfloor p_{0,2} \rfloor + 1$, as in (4.21). Thus, p^* outperforms space-sharing and so do all $p \geq p^*$. As $N_2(k-e-1) = (k-e)(k+e-1)(k-e-1) \geq 0$, it follows that $p_{0,2} \leq k-e-1$, and thus $p^* \leq k-e$. $\square$

Proposition 4.4 agrees with known particular cases. 1) When $e = 1$, we have $p^* = 1$ and the only eliminated point ($p = 0$) coincides with the MBCR point, in agreement with [87]. 2) The optimal point in Theorem 4.2 ($p = k-e-1$) is not a corner point for $k = e+1$, because of $p^* = k - e > p$ and Proposition 4.4. Indeed, the point with $p = k - e - 1$ lies exactly on the segment joining the MBCR and the MSMR point. 3) When $k > e+1$, the optimal point in Theorem 4.2 is a corner point, as $\bar{\beta}'_{k+e-1} - \bar{\beta}_{k+e-1} = \frac{(k-e)(k-e-1)}{k(k+e)(k-1)^2} > 0$.

While Proposition 4.4 describes exactly $\mathcal{R}$, it does not give insight into when a particular point $r = 2e+p$ is a corner point or not. We focus on the analysis of the sign of $N_1(k)$ in (4.22). $N_1(k)$ is linear in k. Depending on the sign of its the leading coefficient $-2e^2+2e+p^2+p$, there may exist an integer k_{th} such that when $k \geq k_{\text{th}}$ space-sharing enhances the achievability region (i.e., $N_1(k) \leq 0$) and does not enhance it when $k < k_{\text{th}}$. That is, a point with the same r may be a corner point for some $(k+e, k, k, e)$ systems and may be not a corner point for other systems, with higher reconstruction parameter k.

For example, for $e > 1$, let $p = e - 1$, we have $N_1(k) = e(1-e)(k-5e+1)$. It follows that, for systems with $k \geq 5e-1$, the point $r = 2e + (e-1) = 3e-1$ is outperformed by space-sharing. For systems with $2e-1 \leq k < 5e-1$, the point $r = 3e-1$ is a corner point.

The next proposition addresses the cases in which a particular point $r = 2e + p$ is a corner point, using a similar argument as the above example.

Proposition 4.5. *Consider the achievable point $r = 2e + p$, for fixed $(e, k), e > 1$. Let $p_{max} = \left\lfloor \frac{1}{2}(\sqrt{8e(e-1)-1} - 1) \right\rfloor$ and $k_{th}(p) = \left\lceil (1-e)\frac{\binom{p+1}{2}+2\binom{e+1}{2}+ep}{\binom{p+1}{2}-2\binom{e}{2}} \right\rceil$. Then, Table 4.1 specifies the scenarios in which $(\bar{\alpha}_r, \bar{\beta}_r)$ is a corner point in $\mathcal{R}$.*

$(\bar{\alpha}_r, \bar{\beta}_r)$	$k < k_{\text{th}}(p)$	$k \geq k_{\text{th}}(p)$
$p \leq p_{\max}$	✓	✗
$p > p_{\max}$	✓	

Table 4.1: Summary of cases for which $(\bar{\alpha}_r, \bar{\beta}_r)$ is a corner point in $\mathcal{R}$. The symbol ✓ means $(\bar{\alpha}_r, \bar{\beta}_r)$ is a corner point while the symbol ✗ denotes the other case.

Proof. We examine $N_1(k)$. First, we note that when $-2e^2 + 2e + p^2 + p > 0$, the point $r = 2e + p$ is a corner point for all systems. Indeed, as $N_1(e+1) = 2ep(e+p) > 0, p > 0$, we have $N_1(k) > 0, \forall k \geq e + 1, p > 0$. It follows that, for a fixed (e, p), we need to determine the sign of $-2e^2 + 2e + p^2 + p$. We have

$$-2e^2 + 2e + p^2 + p < 0 \iff p(p+1) < 2e(e+1) \iff \binom{p+1}{2} < 2\binom{e}{2}, \tag{4.24}$$

$$\iff p < \sqrt{2e^2 - 2e - \frac{1}{4}} - \frac{1}{2} = \frac{1}{2}(\sqrt{8e(e-1)-1} - 1). \tag{4.25}$$

We note that right hand side of (4.25) can not be an integer, as otherwise $\sqrt{8e(e-1)-1}$ should be an odd integer, implying $8e(e-1) - 1 \equiv 1 \mod 4$, which leads to a contradiction as $8e(e-1) - 1 \equiv 3 \mod 4$. This also implies that the slope of $N_1(k)$ cannot be 0, for $e > 0, \forall p \geq 0$. The maximum value of p satisfying (4.25) is given by

$$p_{\max} = \left\lfloor \frac{1}{2}(\sqrt{8e(e-1)-1} - 1) \right\rfloor. \tag{4.26}$$

Thus, a point $r = 2e + p, p > p_{\max}$ is a corner point for any $(k + e, k, k, e)$ system such that $r \leq k + e$. For each $0 \leq p \leq p_{\max}$, the point $r = 2e + p$ is a corner point if and only if $\text{sign}(\bar{\beta}'_r - \bar{\beta}_r) = \text{sign}(N_2(k)) > 0$. From (4.22), Let k_0 be the solution to the linear equation

$N_1(k) = 0$. Then, after simplification, we have

$$k_0 = \frac{(1-e)(2e^2+2ep+2e+p^2+p)}{-2e^2+2e+p^2+p} = (1-e)\frac{\binom{p+1}{2}+2\binom{e+1}{2}+ep}{\binom{p+1}{2}-2\binom{e}{2}}. \tag{4.27}$$

As $p \le p_{\max}$, we have $-2e^2+2e+p^2+p < 0$, which also implies that $k_0 > 0$. As $N_1(e+1) = 2ep(e+p)$, we have $k_{\text{th}} \ge e+1$, with equality iff $p = 0$. It can be checked from (4.27) that when $p = e-1$, $k_0 = 5e-1$. For $k \ge k_0$, point r is not a corner point. As k is an integer and k_0 is not necessarily an integer, it follows that $k \ge k_0 \iff k \ge \lceil k_0 \rceil \triangleq k_{\text{th}}$. □

Using Proposition 4.5, Corollary 4.2 follows.

Corollary 4.2. *For a $(k+e, k, k, e)$ system with $e \ge 2$, we have*

- *p^* in (4.21) can also be expressed as*

$$p^* = 1 + \max\{p : p \le p_{max} \text{ and } k \ge k_{th}(p)\} \tag{4.28}$$

$$= 1 + \max\left\{p : p \le \left\lfloor \frac{1}{2}(\sqrt{8e(e-1)-1}-1)\right\rfloor \right. \tag{4.29}$$

$$\left. \text{and } k \ge \left\lceil (1-e)\frac{\binom{p+1}{2}+2\binom{e+1}{2}+ep}{\binom{p+1}{2}-2\binom{e}{2}}\right\rceil\right\}. \tag{4.30}$$

- *The number of corner points in $\mathcal{R}$ is given by $n_c \triangleq |\{r : 2e+p^* \le r \le k+e\}| + 1 = k-e+2-p^*$.*

- *As a function of k, p^* levels out at $k = k_{th}(p_{max})$ and its final value is given by $1+p_{max}$.*

Example 4.7. *We consider the setting of Figure 4.5: $e = 3, k = 14$. We obtain $p_{max} = 2, p^* = 3$. This means the points r, for $6 \le r \le 2e+p^*-1 = 8$ are not corner points in $\mathcal{R}$ and the number of corner points is $n_c = 10$. This clearly matches the observations made in Figure 4.5.*

4.5 Second construction for interior points

In this section, we present another family of codes improved upon [87] that encapsulates Construction 4.1 as a special case.

Let G denote the $(N(r-m) \times n\alpha)$ generator matrix after vectorization of the code in (4.7), with $t = r$. Every node corresponds to a set of α columns of G. Different from Construction 4.1, we allow $k \leq n - m$, hence we may feed $F_c \triangleq (r-m)N$ dependent symbols to the generator matrix. Let T be $k\alpha$ columns of G corresponding to k out of the n nodes. Let $G|_T$ be the submatrix of G consisting of the columns of T. Then the rank of $G|_T$, denoted by $\rho_{k,m,r}$, is independent of the choice of the k nodes, and is given by [87]

$$\rho_{k,m,r} = \sum_{p=\max(1,r-(n-k))}^{\min(k,r)} \binom{k}{p}\binom{n-k}{r-p} \min(p, r-m). \tag{4.31}$$

The maximum amount of information that can be stored in the system, $\mathcal{M}$, is upper bounded by $\rho_{k,m,r}$, i.e., $\mathcal{M} \leq \rho_{k,m,r}$. For instance, when $m = n - k$, it can be checked that $\rho_{k,m,r} = (r-m)N = F_c$.

To generate the F_c dependent symbols, we add another layer of inner code to Construction 4.1. Moreover, the information symbols are assumed to be over $\mathbb{F}_q^\kappa$, for the finite field $\mathbb{F}_q$ and an appropriately chosen positive integer κ.

Construction 4.2. *For an (n, k, d, e) system, similarly to Construction 4.1, the code construction is parameterized by m, r, such that $e \leq m \leq n - k$ and $m + 1 \leq r \leq n$ (we assume $t = r$). For each pair (r, m), let $\mathcal{M}$ be given by (4.31), $\alpha = \binom{n-r}{r-1}$. First, the $\mathcal{M}$ information symbols $\{v_i\}_{i=1}^{\mathcal{M}}$, $v_i \in \mathbb{F}_q^\kappa$, are used to construct a linearized polynomial*

$$f(x) = \sum_{i=1}^{\mathcal{M}} v_i x^{q^{i-1}}. \tag{4.32}$$

The linearized polynomial is then evaluated at F_c elements of $\mathbb{F}_q^\kappa$ to obtain $\{f(\theta_i), 1 \leq i \leq F_c\}$, which when viewed as vectors over $\mathbb{F}_q$, are linearly independent. Finally, the evaluation points $\{f(\theta_i), 1 \leq i \leq F_c\}$ are fed to the encoder in Construction 4.1.

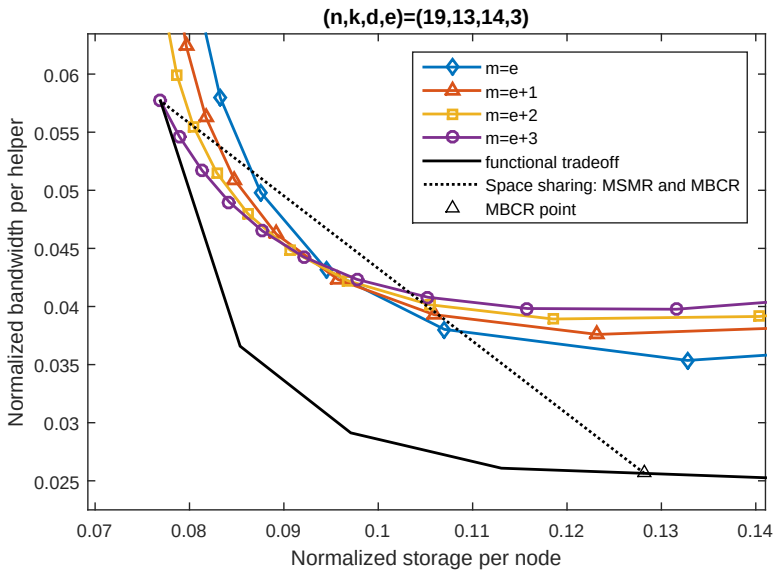
Repair: *The repair of e nodes is similar to Construction 4.1, and the contribution of each helper is given by (4.9).*

We note that the elements in Construction 4.1 are defined over an alphabet of size q, while the evaluation points are defined over $\mathbb{F}_q^\kappa$. This difference can be resolved by viewing $\{f(\theta_i), 1 \leq i \leq F_c\}$ as vectors over $\mathbb{F}_q$ and applying Construction 4.1 to each of their components. Similarly, the repair is carried out component-wise. The linearized polynomial evaluations are an instance of rank-metric codes. In [87, Proposition 5], it is shown that the use of rank-metric codes guarantees the reconstruction property of the regenerating code. Moreover, [87] shows that when $\kappa \geq F_c$, the symbols $\{f(\theta_i), 1 \leq i \leq F_c\}$ can be made independent over $\mathbb{F}_q$. In fact, rank metric codes may be replaced by other linear codes, as long as the reconstruction property is satisfied, so as to reduce the field size [87]. Furthermore, we note that when $m = n - k$, the use of rank-metric codes is not needed, and the code obtained is simply the code in Construction 4.1.

Remark 4.7. *Construction 4.2 generalizes the non-canonical construction in [87], which is designed for repairing single erasures. Moreover, the non-canonical construction in [87] is based on MDS codes, rather than MSMR codes, and its repair scheme is based on the naive repair scheme of MDS codes. Finally, non-canonical codes in [87] set $m = n - d$, while in Construction 4.2, m takes arbitrary values, such that $e \leq m \leq n - k$.*

Remark 4.8. *The repair process in Construction 4.2 does not take into account the dependency introduced by rank-metric codes among the $F_c = (r - m)N$ intermediate symbols. It may be possible to reduce further the repair bandwidth by leveraging such dependency.*

By varying m and r in Construction 4.2, we obtain various achievability points. Construction



[t]

Figure 4.6: Achievable points using Construction 4.2 for an $(n, k, d, e) = (19, 13, 14, 3)$ system. The x-axis is the normalized storage per node $\bar{\alpha}$ and the y-axis is the normalized bandwidth $\bar{\beta}$. When $m = n - k = e + 3$, the corresponding curve with circles coincides with Construction 4.1.

4.1 is a special case of Construction 4.2, corresponding to $m = n - k$. In particular, when $k = d, n = k + e$, Constructions 4.1 and 4.2 coincide as $m = n - k = e$. For other parameters, simulation shows that Construction 4.1 performs better close to MSMR while Construction 4.2 with $m = e$ performs better close to the MBCR. Figure 4.6 plots the achievable points by Construction 4.2 for an $(n, k, d, e) = (19, 13, 14, 3)$ system, for various values of $m, e \leq m \leq n - k$.

4.6 Adaptive multi-node repair for MBR codes

In this section, we study multi-node repair for MBR codes, allowing a varying number of helpers and a varying number of failures. In the first chapter, we proved that MBMR codes are not achievable for linear exact repair codes, when $2 \leq e < k$. When $e = 1$, exact MBMR codes are MBR codes and their existence is well established in the literature [81]. *Adaptive* regenerating codes possess the extra feature that the number of helpers involved in the repair

process can be adaptively selected, which provides the storage system with robustness to the network varying conditions [54, 76]. Adaptive MSR codes have been constructed in [80]. On the other hand, adaptive MBR codes have been investigated in [94], in which case optimal repair means that the total repair bandwidth for each number of helpers d is the lowest possible, and is given by $\gamma = \alpha, \forall d_{\min} \leq d \leq d_{\max}$ (assuming the storage per node contains no redundancy). Here $d_{\min}, d_{\max}$ are between k and $n-1$. It is shown in [94] that adaptive MBR codes, designed for arbitrary d, $d_{\min} \leq d \leq d_{\max}$, are equivalent to MBR codes that are designed for the worst-case number of helpers $d_{\min}$, and they satisfy optimal repair for arbitrary number of helpers $d_{\min} \leq d \leq d_{\max}$. Namely, adaptive MBR codes satisfy for any $d_{\min} \leq d \leq d_{\max}$,

$$\alpha = \frac{2d_{\min}\mathcal{M}}{-k^2 + k + 2kd_{\min}} = \frac{d_{\min}\mathcal{M}}{d_{\min}k - \binom{k}{2}}, d\beta = \alpha,$$

where the storage size α corresponds to the MBR code with $d_{\min}$ helpers.

A natural question of interest is whether there exists an MBR code that efficiently recover from varying number of failures simultaneously. In this section, we investigate the problem of repairing multiple failures in MBR codes under exact repair, for *varying number of helpers d and varying number of failures e*, such that $d_{\min} \leq d \leq d_{\max}$, $1 \leq e \leq k$, $e + d \leq n$. First, we derive a lower bound on the multi-node repair bandwidth for MBR codes, which applies to exact and functional codes. We assume that an MBR code is designed for d helpers, and we want to repair e failures. To emphasize the dependency on e, denote the total repair bandwidth by $\gamma_{\text{MBR}}(e)$.

Theorem 4.3. *Consider an (n, k, d, α, β) MBR regenerating code, the total repair bandwidth $\gamma_{MBR}(e)$ needed to repair any set of $1 \leq e \leq k$ nodes satisfies*

$$\gamma_{MBR}(e) \geq e\alpha - \binom{e}{2}\frac{\alpha}{d}. \tag{4.33}$$

Proof. Assume w.l.o.g that the first e nodes are to be repaired. From [95], at MBR point, for any set of A nodes of size $m < k$ and for $i \notin A$, we have $H(W_i|W_A) = (d-m)\beta$. Therefore,

$$H(W_{[e]}) = \sum_{i=1}^{e} H(H_i|W_{[i-1]}) = \sum_{i=1}^{e}(d-i+1)\beta = (ed - \binom{e}{2})\beta.$$

Noting that at the MBR, $\alpha = d\beta$, (4.33) follows. □

We now briefly describe a construction of adaptive MBR codes that simultaneously and efficiently repair single node failures, presented in [94]. Then, we show how to optimally repair multiple failures in this construction.

4.6.1 Adaptive single-failure MBR construction

The construction is based on product matrix codes [94, 81]. Let $\alpha = \prod_{d=d_{\min}}^{d_{\max}} d$. Define $z = \frac{\alpha}{d_{\min}}$ and construct the $(\alpha \times \alpha)$ data matrix M as

$$M = \begin{bmatrix} M_1 & O & \cdots & O \\ O & M_2 & \cdots & O \\ \vdots & & \ddots & \vdots \\ O & \cdots & O & M_z \end{bmatrix},$$

where O is a $(d_{\min} \times d_{\min})$ zero matrix and each of the submatrices M_i is filled with information symbols, and is symmetric and satisfies the structural properties of a product-matrix MBR code for parameters k and $d_{\min}$. For instance, M_i is given by

$$M_i = \begin{bmatrix} N_i & L_i \\ L_i^t & O' \end{bmatrix},$$

where N_i is a symmetric $(k \times k)$ matrix, L_i is $(k \times (d_{\min} - k))$ matrix, and O' is $(d_{\min} - k) \times (d_{\min} - k)$ zero matrix. let Ψ be an $(zn \times d_{\min})$ Vandermonde matrix, with rows denoted by ψ_j^t, for $1 \leq j \leq zn$. Then, storage node l is associated with

$$\mathbf{w}_l^t = \left[\psi_{(l-1)z+1}^t, \ldots, \psi_{lz}^t\right] M = \left[\psi_{(l-1)z+1}^t M_1, \ldots, \psi_{lz}^t M_z\right].$$

Single node repair. Denote the set of helpers by $\mathcal{H}$ such that $|\mathcal{H}| = d$ and $d_{\min} \leq d \leq d_{\max}$. Let Ω be an $(z \times z)$ matrix such that Ω^t is a Vandermonde matrix. Assume that node f fails. Let Ω_d be an $(\frac{\alpha}{d} \times z)$ matrix containing the first $\frac{\alpha}{d}$ rows of Ω. Moreover, let Φ_i be an $(\alpha \times z)$ matrix

$$\Phi_i = \begin{bmatrix} \psi_{(i-1)z+1} & & \\ & \ddots & \\ & & \psi_{iz} \end{bmatrix}.$$

Each helper node $i_j \in \mathcal{H}$ transmits $\mathbf{s}_{i_j,f}^t = \mathbf{w}_{i_j}^t \Phi_f \Omega_d^t$. After simplification, the replacement node obtains

$$\mathbf{w}_f^t \left[\Phi_{i_1} \Omega_d^t, \ldots, \Phi_{i_d} \Omega_d^t\right] = \mathbf{w}_f^t \Theta_{\mathcal{H}}.$$

Noting that $\Theta_{\mathcal{H}}$ is invertible [94], the replacement node can thus recover $\mathbf{w}_f^t$.

4.6.2 Adaptive multi-node repair in MBR codes

We state our result in the following theorem.

Theorem 4.4. *Adaptive single-failure MBR regenerating codes with storage per node α and arbitrary number of helpers $d_{\min} \leq d \leq d_{\max}$, presented in[94], can simultaneously and optimally repair e failures with d helpers, for all $d_{\min} \leq d \leq d_{\max}$, $1 \leq e \leq k$, $e + d \leq n$.*

Proof. Assume w.l.o.g that the first e nodes failed and d helpers are used, where $d_{\min} \leq d \leq d_{\max}$, $1 \leq e \leq k$, $e + d \leq n$. Denote the helpers by the set $\mathcal{H} = \{i_1, \ldots, i_d\}$. First, the repair of node 1 is done by contacting all the d helpers and downloading $\frac{\alpha}{d}$ symbols from each one of them, using the procedure described for single node repair. Node 2 is then repaired using only $d_{\min}$ helpers, comprising repaired node 1 and any other $d_{\min} - 1$ helpers in $\mathcal{H}$, such that each helper provides $\frac{\alpha}{d_{\min}} = z$ symbols. The same procedure is then applied repeatedly until recovering the last node e by contacting any $d_{\min} - e + 1$ helpers in $\mathcal{H}$ and using contributions from the $e - 1$ already repaired nodes. The overall repair bandwidth is given by

$$d\frac{\alpha}{d} + \sum_{i=1}^{e-1} z(d_{\min} - i) = e\alpha - \binom{e}{2}\frac{\alpha}{d_{\min}},$$

which matches the bound in (4.33), establishing the optimality of the repair procedure. □

Remark 4.9. *Repairing e failures in an $(n, k, d_{\min}, \alpha, \beta)$ MBR code separately requires a bandwidth of size $e\alpha$. However, simultaneously repairing e failures using $d \geq d_{\min}$ reduces the bandwidth by $\binom{e}{2}\frac{\alpha}{d_{\min}}$.*

Remark 4.10. *The repair procedure of multiple erasures in Theorem 4.4 is asymmetric. However, one can always duplicate the code a sufficient number of times to achieve a symmetric repair strategy (e.g. [63]).*

4.7 Conclusion

In this chapter, We studied the problem of designing multi-node exact-repair regenerating codes operating at interior points. We first described a construction, termed Construction 4.1, that achieves a new set of interior points. In particular, we proved the optimality of one point on the functional centralized repair tadeoff. Moreover, considering minimum bandwidth cooperative repair codes as centralized repair codes, we determined explicitly the best achievable region obtained by space-sharing among all known points, for the $(k + e, k, k, e)$

system. We then described another construction, Construction 4.2, that includes Construction 4.1 as a special case, and that generates various achievable points for a general (n, k, d, e) system. Finally, we studied the problem of designing MBR codes with multi-node repair capabilities.

Chapter 5

Polar codes for resistive memories

5.1 Introduction

In the previous chapters coding for storage systems are considered. The focus of this chapter is coding for storage devices, and in particular, for resistive memories (RRAMs).

Resistive memories (RRAMs) represent a promising memory technology with a high potential as an alternative for floating-gate-based nonvolatile memories. In RRAMs, the high/low cell resistance represents the stored bit. In order to retrieve the stored data, resistive sensing (reading) techniques are adopted. To achieve higher density, access devices such as transistors, diodes and selectors are removed. However, the main disadvantage of the selector-less (gate-less) crossbar-based memories is *sneak paths* which are undesired current paths and voltage drops limiting the readability of the array. In this paper, a parallel reading of an entire crossbar row [96] is adopted as shown in Figure 1.3. It eliminates the multi-path problem in single-cell reading [97], one of the causes of sneak paths. But, the inevitable wire resistances lead to undesired voltage drops, another type of sneak paths. These voltage drops are functions of the stored data and the wire resistance. At the expected feature size of $F = 5\ nm$ of RRAMs, the wire resistance per cell reaches as high as 90 Ω [21], leading to large voltage drops. We recall from Figure 1.4 that the sneak path effect results in cells of

different reliability levels. In this chapter, we propose error-correcting codes adapted to the crossbar array problem and evaluate their bit error rate (BER) performance under various scenarios.

Related work

Addressing the sneak path problem has attracted a lot of interest from both research and industry communities. Proposed solutions include hardware-based approaches, e.g., transistor gating [98], and/or techniques based on communication and coding theory [99, 100]. The focus of this paper is on the latter approach and our scheme is based on polar codes. Polar codes [101] are the first family of explicit error-correcting codes to provably achieve the capacity of binary symmetric channels, with a low-complexity encoding and successive cancellation decoding. For a code of length N, encoding/decoding has a complexity of $O(N \log N)$. For the above reasons, polar codes constitute an attractive error correction scheme.

Contributions of the chapter

Our contributions are summarized as follows:

- We study polar coding for channels with different reliability levels, termed non-stationary polar codes.
- We argue that the ordering of the channels is important and propose an ordering with a numerically competitive performance.
- We then apply the framework of non-stationary polar codes to the sneak path problem and demonstrate significant improvement in terms of bit error rate (BER). In partic-

ular, we discuss modeling the array cells as either binary symmetric channels (BSCs) and as binary asymmetric channels (BACs) and compare their BER performances.

- Finally, we propose a technique for biasing the number of high-resistance values in the crossbar through puncturing, and show by simulation that the proposed technique can help reduce the BER in certain scenarios.

Notation. A permutation π over the integers $\{0, \ldots, N-1\}$ is denoted as $\pi = [\pi(0), \ldots, \pi(N-1)]$, where $\pi(j)$ is the image of j under π. For a vector $\mathbf{x} = [x_0, \ldots, x_{N-1}]$, $\mathbf{x}_\pi$ denotes the vector $\mathbf{x}_\pi = [x_{\pi(0)}, \ldots, x_{\pi(N-1)}]$.

5.2 Non-stationary polar code construction

For a binary-input discrete memoryless channel (B-DMC), W, with output alphabet $\mathcal{Y}$, we denote its transition probabilities by $W(y|x), x \in \{0,1\}, y \in \mathcal{Y}$, and define the symmetric channel output probability as $W(y) = \frac{1}{2}W(y|0) + \frac{1}{2}W(y|1)$. Define the symmetric capacity $I(W)$ as

$$I(W) \triangleq \sum_{y \in \mathcal{Y}} \sum_{x \in \{0,1\}} \frac{1}{2} W(y|x) \log \frac{W(y|x)}{W(y)}. \tag{5.1}$$

Arikan's polar transformation [101] manufactures out of N independent copies of a given B-DMC channel, a second set of N *synthesized* binary-input channels. The channels show a polarization effect, namely, as N becomes large, the symmetric capacities of the synthesized channels tend towards 0 or 1 for all but a vanishing fraction.

In our framework, in contrast to the original polar codes, we consider the extension of polar codes to the setting where the underlying channels are of *varying reliability levels* (see Figure 5.1 and Figure 5.2). Using the terminology in [102], we refer to such polar codes as

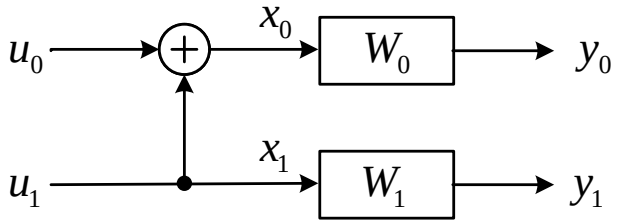


Figure 5.1: Basic channel polarization.

non-stationary polar codes.

Polar Transformation: Let $\mathbf{u} = [u_0, u_1, \ldots, u_{N-1}]$ and $\mathbf{x} = [x_0, x_1, \ldots, x_{N-1}]$ be the input and the output of a length-N polar code, respectively, with $N = 2^n$ for some integer n. The polar transformation is given by

$$\mathbf{x} = \mathbf{u}G, \quad G = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes n},$$

where the symbol $^{\otimes n}$ denotes the n-th Kronecker power operator.

When $n = 1, N = 2$, the basic polarization transformation is applied to two independent channels $W_0 : \{0,1\} \to \mathcal{Y}_0$ and $W_1 : \{0,1\} \to \mathcal{Y}_1$, resulting in two channels, $W' : \{0,1\} \to \mathcal{Y}_0 \times \mathcal{Y}_1$ and $W'' : \{0,1\} \to \mathcal{Y}_0 \times \mathcal{Y}_1 \times \{0,1\}$, given by

$$\begin{aligned} W'(y_0, y_1|u_0) &= \frac{1}{2} \sum_{u_1 \in \{0,1\}} W_0(y_0|u_0 \oplus u_1) W_1(y_1|u_1), \\ W''(y_0, y_1, u_0|u_1) &= \frac{1}{2} W_0(y_0|u_0 \oplus u_1) W_1(y_1|u_1), \end{aligned} \tag{5.2}$$

where $y_0 \in \mathcal{Y}_0, y_1 \in \mathcal{Y}_1$, and $u_0, u_1 \in \{0,1\}$. We denote this single-step transformation by $(W_0, W_1) \mapsto (W', W'')$.

Channel Polarization: The transformation preserves the average symmetric capacity, while exhibiting a polarization effect.

Lemma 5.1 ([102])**.** *Suppose* $(W_0, W_1) \mapsto (W', W'')$*. Then,*

$$\begin{aligned} &I(W') + I(W'') = I(W_0) + I(W_1), \\ &I(W') \leq I(W_i) \leq I(W''), i = 0, 1. \end{aligned} \tag{5.3}$$

The Bhattacharyya parameter of a binary discrete memory-less channel $W : \{0, 1\} \to \mathcal{Y}$, denoted by $Z(W)$, is a measure of the reliability of W, and has been used to bound the error probability of polar codes in [101]. The parameter $Z(W)$ is defined as

$$Z(W) \triangleq \sum_{y \in \mathcal{Y}} \sqrt{W(y|0)W(y|1)}.$$

Lemma 5.2. *Let* $W_0 : \{0, 1\} \to \mathcal{Y}_0$ *and* $W_1 : \{0, 1\} \to \mathcal{Y}_1$ *and* $(W_0, W_1) \mapsto (W', W'')$*.*
(i) The following relations hold:

$$Z(W') \leq Z(W_0) + Z(W_1) - Z(W_0)Z(W_1), \tag{5.4}$$

$$Z(W'') = Z(W_0)Z(W_1), \tag{5.5}$$

$$Z(W') \geq \sqrt{Z(W_0)^2 + Z(W_1)^2 - Z(W_0)^2 Z(W_1)^2}. \tag{5.6}$$

(ii) Equality (5.4) holds with equality if and only if W_0 *or* W_1 *is a binary erasure channel.*
(iii) Equality (5.6) holds with equality if and only if both W_0 *and* W_1 *are binary symmetric channels.*

In Lemma 5.2, parts (i) and (iii) are from [103, Lemma 2.15 and Lemma 2.16] and [104, Lemma 9], and part (ii) can be proved following similar lines as [101, Proposition 5]. Lemma 5.2 implies that reliability improves under a single-step channel transformation in

the sense that

$$Z(W') + Z(W'') \leq Z(W_0) + Z(W_1),$$

with equality if and only if W_0 or W_1 is a binary erasure channel (BEC).

The single-step transformation, as described in (5.2), can be generalized to the case of $N = 2^n$ binary, memoryless, but not necessarily stationary channels $\{W_i\}_{i=0}^{N-1}$. In particular, let $W_{m,i}$ denote the i-th bit channel after m levels of polarization of the sequence $\{W_i\}_{i=0}^{N-1}$, where $W_{0,i} \triangleq W_i, i = 0, \ldots, N-1$. Applying Arikan's polar transformation results in a collection of synthesized channels, such that, for any level $1 \leq l \leq n$, for $0 \leq j < 2^{l-1}, 0 \leq m < 2^{n-l}$, we have

$$(W_{l-1,2^l m+j}, W_{l-1,2^l m+2^{l-1}+j}) \mapsto (W_{l,2^l m+j}, W_{l,2^l m+2^{l-1}+j}). \tag{5.7}$$

In [102], it was shown that the fraction of non-polarized channels approach 0, i.e., for every $0 < a < b < 1$,

$$\liminf_{N\to\infty} \frac{1}{N} |0 \leq i < N : I(W_{n,i}) \in [a, b]\}| = 0.$$

The work in [104] proved that in the asymptotic regime where the blocklength N is large, the effective average symmetric capacity $\bar{I}(\{W_i\}_{i=0}^{\infty}) \triangleq \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} I(W_i)$ is achievable. Moreover, the polar coding scheme is constructed based on Arikan's channel polarization transformation in combination with certain permutations at each polarization level. However, it is not clear whether such permutation choices yield attractive performances for practical lengths. In this work, we are concerned with the performance of non-stationary polar codes in the finite blocklength regime.

Encoding and Decoding: After the polar transformation, one obtains N synthesized channels $\{W^{(i)} \triangleq W_{n,i}, 0 \leq i \leq N-1\}$. A polar code of dimension k transmits k information bits in the k synthesized channels with the highest $I(W^{(i)})$ (we denote the corresponding information set by $\mathcal{I}$), and $N-k$ arbitrary but fixed bits in the remaining $N-k$ synthesized channels (denoted by $\mathcal{F}$). Decoding of polar codes is carried out using successive cancellation decoding as in [101], taking into account the appropriate likelihood ratios of the original channels.

Remark 5.1. *The information set of a non-stationary polar code can be determined as long as $I(W^{(i)})$ are computed. However, their computational complexities are generally unmanageable. Instead, we propose that the reliability parameters of the synthesized channels can be approximated efficiently by $Z(W^{(i)})$ using (5.4) and (5.5) in Lemma 5.2. We call this method the Bhattacharyya bound approach. It is a generalization of the algorithm in [105], developed for regular polar codes. Let $Z_{n,i}$ denote the Bhattacharyya parameter of channel $W_{n,i}$. Then, we apply the following recursion: for level $1 \leq l \leq n$, for $0 \leq j < 2^{l-1}$ and $0 \leq m < 2^{n-l}$, we have*

$$\begin{aligned} Z_{l,2^l m+j} &= Z_{l-1,2^l m+j} + Z_{l-1,2^l m+2^{l-1}+j} - Z_{l-1,2^l m+j} Z_{l-1,2^l m+2^{l-1}+j}, \\ Z_{l,2^l m+2^{l-1}+j} &= Z_{l-1,2^l m+j} Z_{l-1,2^l m+2^{l-1}+j}, \end{aligned}$$

where $\{Z_{0,i}, 0 \leq i < N\}$ are the initial channel Bhattacharyya parameters. The indices of the lowest $N-k$ values in the set of N final stage values form the set $\mathcal{F}$. This algorithm is an evolution of the Bhattacharyya parameters of channels from right to left (see Figure 5.2), preferably applied in log-domain to avoid underflow.

Role of the Channel Ordering: The performance of polar codes depends on the synthesized channels of the information set $\sum_{i \in \mathcal{I}} I(W^{(i)})$ [101]. As illustrated in Figure 5.2, to construct a non-stationary polar code, we propose to apply a permutation π to the vector $\mathbf{x}$

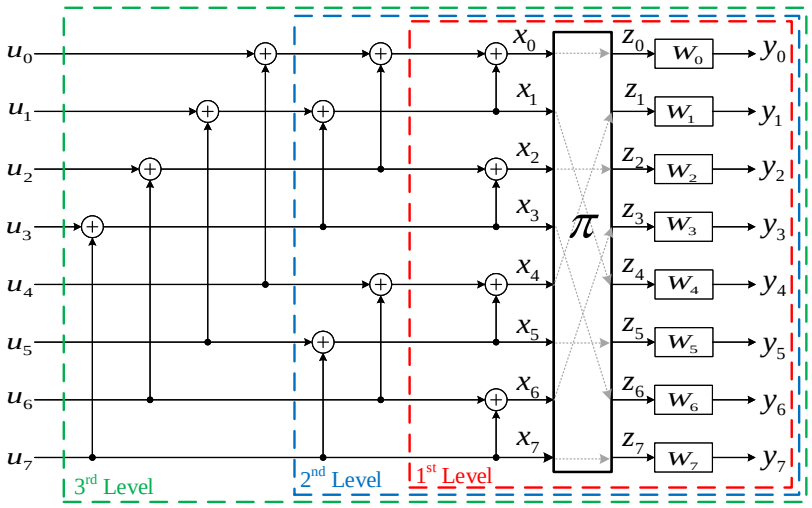


Figure 5.2: Polar encoding with $N = 8$ channels, with permutation $\pi = [0, 4, 2, 6, 1, 5, 3, 7]$.

in order to enhance the overall performance. Ideally, we want to find a permutation π^* such that for all permutation π,

$$\sum_{i \in \mathcal{I}_{\pi^*}} I(W_{\pi^*}^{(i)}) \geq \sum_{i \in \mathcal{I}_{\pi}} I(W_{\pi}^{(i)}), \tag{5.8}$$

where $I(W_{\pi}^{(i)})$ is the symmetric capacity of the i-th synthesized channel under permutation π and $\mathcal{I}_{\pi}$ is its information set.

Remark 5.2. *See Figure 5.3 for a schematic picture of the full encoder and decoder. The permutation π is defined such that $\mathbf{z}_{\pi} = \mathbf{x}$, or equivalently $\mathbf{z} = \mathbf{x}_{\pi^{-1}}$. Then, $z_{\pi(i)} = x_i$, implying that symbol x_i goes through channel $W_{\pi(i)}$, for $0 \leq i < N$. Correspondingly, a reverse permutation is required for the output of the channels. The polar decoder receives as input the vector $\mathbf{y}' = [y_{\pi(0)}, \ldots, y_{\pi(N-1)}] = \mathbf{y}_{\pi}$.*

Remark 5.3. *In general, the channel symmetric capacities can be given in any arbitrary order. In this case, to study the effect of permutations in a canonical way, we apply the composition permutation $\pi_{ord} \circ \pi$ to the output $\mathbf{x}$, where π_{ord} is a permutation that orders the channels in increasing order of symmetric capacities: that is $I(W_{\pi_{ord}(i)}) \leq I(W_{\pi_{ord}(j)})$, for $i < j$. Thus, the output of the polar encoder x_i is mapped to channel $W_{\pi_{ord} \circ \pi(i)}$, i.e., $x_i =$*

$z_{\pi_{ord} \circ \pi(i)}$.

We next show that some channel orderings are equivalent after polarization.

Definition 5.1. *Given N B-DMC channels $\{W_i\}_{i=0}^{N-1}$, an equivalence class of permutations over $\{0, \ldots, N-1\}$ consists of all channel permutations π such that the n-level polarization transformation of the channels $\{W_{\pi(i)}\}_{i=0}^{N-1}$ results in the same symmetric capacities.*

Lemma 5.3. *The number of permutation classes is upper bounded by $\frac{N!}{2^{N-1}}$.*

Proof. Consider the two single-step transformations $(W_0, W_1) \mapsto (W_1', W_1'')$ and $(W_1, W_0) \mapsto (W_2', W_2'')$. From (5.1) and (5.2), we have

$$\begin{aligned} I(W_1'') &= \sum_{y_0, y_1, u_0} \sum_{u_1} \frac{1}{4} W_0(y_0|u_0 \oplus u_1) W_1(y_1|u_1) \log \frac{\frac{1}{2} W_0(y_0|u_0 \oplus u_1) W_1(y_1|u_1)}{W_1''(y_0, y_1, u_0)} \\ &= \sum_{y_0, y_1, u_0} \sum_{u_1} \frac{1}{2} W_0(y_0|u_1) W_1(y_1|u_0 \oplus u_1) \log \frac{W_0(y_0|u_1) W_1(y_1|u_0 \oplus u_1)}{W_1''(y_0, y_1, u_0)}, \end{aligned} \tag{5.9}$$

where the second equality is obtained by a change of variable. On the other hand, we have

$$\begin{aligned} W_1''(y_0, y_1, u_0) &= \sum_{u_1 \in \{0,1\}} W_0(y_0|u_0 \oplus u_1) W_1(y_1|u_1) \\ &= \sum_{u_1 \in \{0,1\}} W_0(y_0|u_1) W_1(y_1|u_0 \oplus u_1) \\ &= W_2''(y_0, y_1, u_0). \end{aligned} \tag{5.10}$$

From (5.9) and (5.10), we obtain $I(W_1'') = I(W_2'')$. From (5.3), it follows $I(W_1') = I(W_2')$. Thus, the single-step transformation is symmetric in the sense that exchanging the order of W_0 and W_1 does not impact the symmetric capacities of the synthesized channels. It follows that for $N = 2^n$ channels, permuting channel W_{2i} and channel W_{2i+1}, for $0 \leq i \leq \frac{N}{2} - 1$ does not change the symmetric capacities of the synthesized channels obtained at the end of the polarization process. Extending the above reasoning to level 2, permuting

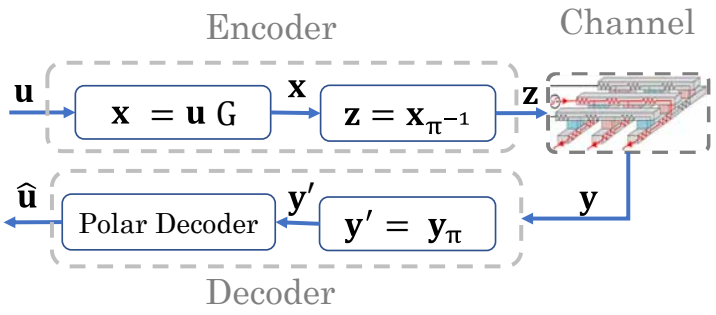


Figure 5.3: Full system model.

(W_{4i}, W_{4i+1}) with (W_{4i+2}, W_{4i+3}), $0 \leq i \leq \frac{N}{4} - 1$ results in the same symmetric capacities of the synthesized bit channels. At level 3, we can permute $(W_{8i}, W_{8i+1}, W_{8i+2}, W_{8i+3})$ with $(W_{8i+4}, W_{8i+5}, W_{8i+6}, W_{8i+7})$, $0 \leq i \leq \frac{N}{8} - 1$ without changing the symmetric capacities. Similar observation applies to all polarization levels l, for $1 \leq l \leq n$. Thus, the number of permutation classes is upper bounded by $\frac{N!}{2^{1+2+\ldots+\frac{N}{2}}} = \frac{N!}{2^{1+2+\ldots+2^{n-1}}} = \frac{N!}{2^{2^n-1}} = \frac{N!}{2^{N-1}}$. □

Remark 5.4. *Equivalent classes of permutations can be equally defined in terms of the Bhattacharyya parameters instead of symmetric capacities, and the statement of Lemma 5.3 holds.*

Example 5.1. *Figure 5.4 illustrates the polarization process for $N = 4$ channels consisting of two different binary erasures channels (BECs). Based on Lemma 5.3, it can be checked that in this example, the number of permutation classes is 2, as shown in Figure 5.4. For a target rate of $\frac{1}{2}$, the rightmost permutation in Figure 5.4 is advantageous as it corresponds to the highest sum-rate of the two best synthetic channels.*

Example 5.1 shows that in the finite blocklength regime, the ordering of the channels matters, depending on the code rate. For $N = 4$ BECs with different erasure probabilities, Proposition 5.1 determines the best ordering depending on the target rate.

Proposition 5.1. *Consider $N = 4$ BECs with parameters $\epsilon_i, 1 \leq i \leq 4$. Then, the permutation $\pi_{ord} \circ \pi$, where $\pi = [0, 3, 1, 2]$, satisfies (5.8).*

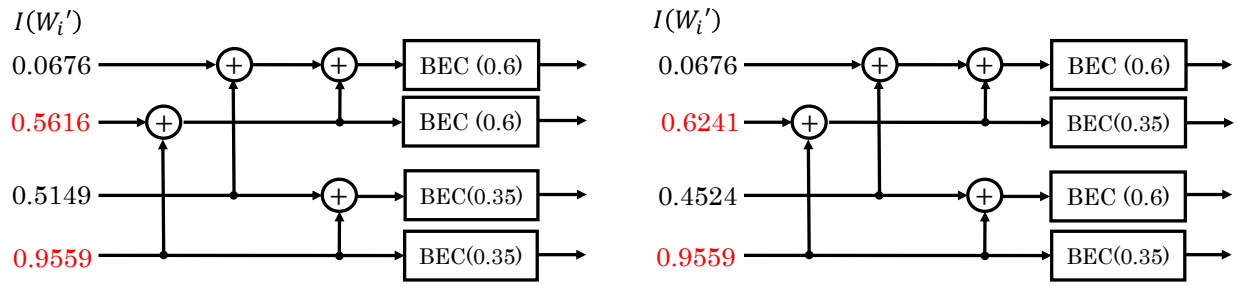


Figure 5.4: Two different ordering of the four BECs, with their synthetic channel rates. The permutation on the left is $\pi = [0, 1, 2, 3]$ and the permutation on the right is $\pi = [0, 2, 1, 3]$.

Proof. See Appendix 5.4.1. □

Extending the analysis in Proposition 5.1 is not tractable. In the following, we consider the bit-reversal permutation defined below. We note that such permutation was used in [106] in the context of rate-compatible punctured polar codes and was shown to yield attractive performance compared to other existing or random puncturing patterns. We explore the performance under bit-reversal permutation through various numerical examples.

Definition 5.2. *We define ψ to be the bit-reversal permutation. For each integer $i \in \{0, \ldots, 2^n - 1\}$, $\psi(i)$ is the integer obtained by reversing the binary representation of i. That is, let $i = \sum_{j=1}^{n} b_j 2^{j-1}$, then $\psi(i) = \sum_{j=1}^{n} b_j 2^{n-j}$. As an example, when $N = 8$, $\psi = [0, 4, 2, 6, 1, 5, 3, 7]$, as illustrated in Figure 5.2.*

From the proof of Proposition 5.1 and Lemma 5.3, when $\epsilon_0 = \epsilon_1$ or $\epsilon_2 = \epsilon_3$, then ψ satisfies (5.8) and is optimal.

Example 5.2. *We consider N binary symmetric channels (BSCs) with varying cross-over probabilities, linearly spaced and centered at value $p \in \{0.05, 0.065, 0.08, 0.095, 0.11\}$, with maximum deviation of 0.045. We consider a rate $\frac{1}{2}$ and a block size $N = 2^{10}$. We evaluate the performance of a regular polar code designed for the average BSC channel, with p_{avg} satisfying $N(1 - h(p_{avg})) = \sum_{i=0}^{N-1}(1 - h(p_i))$, where $h(\cdot)$ is the binary entropy function and p_i's are the cross-over probabilities of the channels. The default ordering of the channels is such*

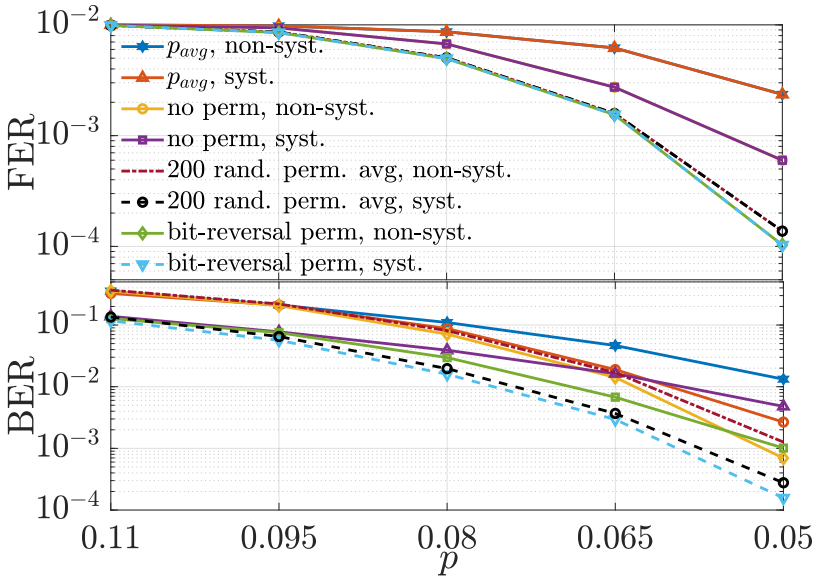


Figure 5.5: Performance evaluation for BSCs with linearly spaced cross-over probabilities. $N = 1024, k = 512$.

that the channels' Bhattacharyya parameters are ordered in a descending order (i.e., decreasing order of p_i's). We evaluate the performance with the default ordering (no permutation) and the performance with the bit-reversal permutation. We also evaluate the performance obtained by averaging 200 *random permutations. For each scenario, we evaluate the frame error rate (FER) and the bit error rate (BER) with non-systematic encoding and systematic encoding [107], for* 10^4 *runs. Figure 5.5 illustrates the results. Similar to [107], we observe that systematic encoding of non-stationary polar codes enhances the BER performance compared to non-systematic encoding, while keeping the FER unchanged. The regular polar code with p_{avg} exhibits the worst BER, while the non-stationary polar code under systematic encoding with $\pi = \psi$ performs the best. In particular, the latter scenario outperforms all 200 random permutations for all values of p.*

Based on the observation that bit-reversal permutation achieves competitive BER numerically, we choose to apply $\pi = \psi$ for our non-stationary polar codes, so as to mitigate the sneak path problem in crossbar arrays.

5.3 Application of the proposed polar codes to RRAMs

5.3.1 General framework

As outlined in the introduction, the crossbar array cells exhibit different reliability levels. For this reason, we propose the application of non-stationary polar codes to address the problem. We apply a two-step approach:

Step 1). We first estimate a single detection threshold for each wordline (row) to minimize the overall uncoded BER per word. The threshold for each wordline is estimated by generating large training data and then applying a good binary classifier. For instance, we observe that a logistic regression-based classifier gives superior performance in terms of accuracy and speed.

Step 2). Based on the estimated thresholds in Step 1), we model the read channels as BSCs or BACs. In particular, we estimate the cross-over probabilities of each cell in the array. Our simulation shows that the channels are indeed varying. We then apply non-stationary polar codes using the cell characterizations.

In [35], we proposed to encode each row separately. In this work, we focus on encoding the entire array. This can be suitable for applications such as archival data and image storage. Assuming the crossbar array size is $(N_1 \times N_2)$, then, the blocklength is $N = N_1 N_2$. The encoded output symbol $z_{in+j}, 0 \leq i < m, 0 \leq j < n$, is stored at the (i, j)-th entry in the crossbar array (i.e., we vectorize the array row by row). Instead of using high-level models for the sneak path problem, such as in [99, 100], we use a SPICE-like simulator that is built based on accurate modeling of the resistive crossbar array [21]. This numerical simulator offers a fast alternative to SPICE simulators while maintaining the same simulation accuracy. In our simulations, the high resistance state, representing 1, and the low resistance state,

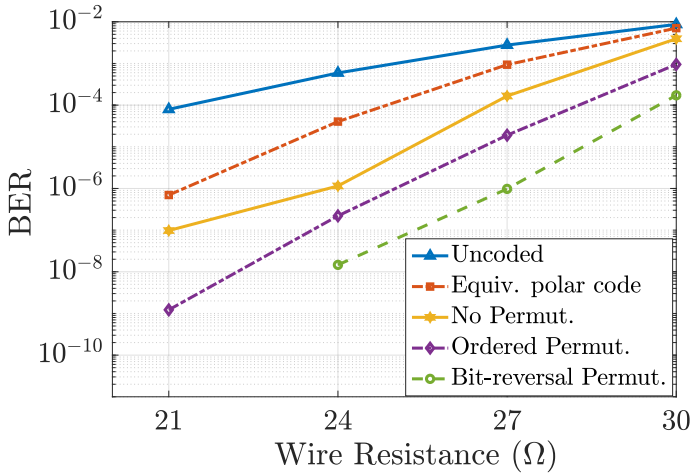


Figure 5.6: Performance evaluation for a (32×32) crossbar array, code rate 0.8.

representing 0, are set to $1M$ Ω and $1k$ Ω, respectively.

5.3.2 BSC and BAC channel modeling

First, the array cells are modeled as BSCs. The cross-over probability is found to match the BER of each cell.

Example 5.3. *In Figure 5.6, we simulate the BER performance of systematic polar codes for four cases: (i) equivalent regular polar codes correspond to BSCs with parameter p_{avg}, similar to Example 5.2, (ii) no permutation, (iii) permutation π_{ord}, and (iv) permutation $\pi = \pi_{ord} \circ \psi$, where π_{ord} is as defined in Remark 5.3. Clearly, the BER permutation under $\pi = \pi_{ord} \circ \psi$ outperforms the other permutations.*

Analyzing further the uncoded error distribution, as one may infer from Figure 1.5, we find that the conditional error distributions under 0's and 1's are different, i.e., $P(\text{error}|0) \neq P(\text{error}|1)$. Taking this observation into consideration, we model the crossbar array cells as binary asymmetric channels (BACs) and apply the non-stationary polar codes.

Example 5.4. *In Figure 5.7, we compare the systematic BER performance under both BSC and BAC modeling using the bit-reversal permutation. As expected, the BER performance*

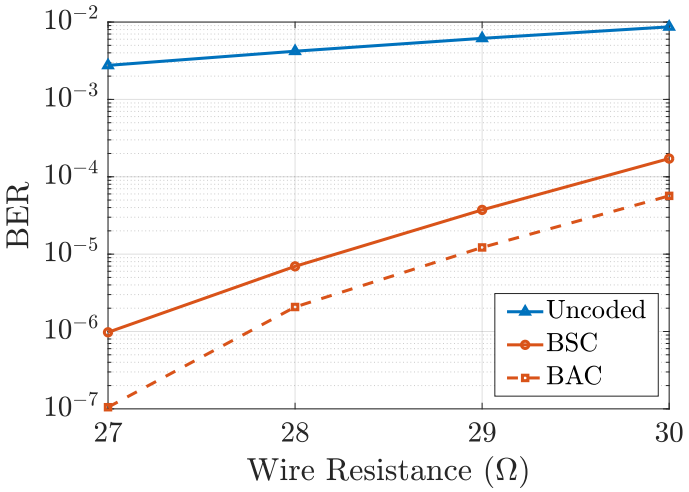


Figure 5.7: BER performance under BSC and BAC modeling for a (32×32) crossbar, code rate 0.8.

under the more accurate BAC model is better. The improvement is 3X – 9X for wire resistance between 27 Ω *and* 30 Ω.

5.3.3 Punctured polar codes

In this subsection, we propose a technique that can enhance the BER performance in some scenarios by biasing the fraction of high resistance cells. The sneak paths exist through the cells having low resistances, causing inter-cell interference [99]. Intuitively, having more high resistances in the array helps mitigate the sneak path problem. To leverage this intuition, we investigate the use of a (punctured) polar code of a shorter length, say $N - N_p$, while storing high resistances in the corresponding punctured N_p cells in the array. Clearly, there is a trade-off between two opposite factors: puncturing reduces the number of redundant codeword symbols, hence degrading the performance of polar codes, while high resistances decrease the sneak path effect resulting in fewer read errors. In the following, we investigate the application of the above approach to the crossbar array.

A *punctured* polar code is obtained from the N-length parent polar code using a binary puncturing vector $\mathbf{w} = [w_0, w_1, \ldots, w_{N-1}]$, where the zeros imply the punctured positions.

We note that the information set $\mathcal{I}$ should be recomputed as in Remark 5.1, when we consider puncturing.

Punctured polar codes have been investigated by many works, and several efficient puncturing patterns have been proposed in the literature [106]. In [106], the authors proposed an empirically good puncturing algorithm, termed quasi-uniform puncturing (QUP). QUP-polar codes were shown through simulations to outperform the performance of turbo codes in WCDMA (Wideband Code Division Multiple Access) or LTE (Long Term Evolution) wireless communication systems in the large range of code lengths. We adopt QUP as our puncturing pattern and we highlight its advantages below.

Definition 5.3 ([106])**.** *The QUP is described as follows:*

1. *Initialize the vector* $\mathbf{w}$ *as all ones, and then set the first* N_p *bits of* $\mathbf{w}$ *to zeros;*

2. *Perform bit-reversal permutation on the vector* $\mathbf{w}$ *and obtain the puncturing pattern.*

Example 5.5. *Let* $N = 8, N_p = 3$*. The initial puncturing vector is* $\mathbf{w} = [0, 0, 0, 1, 1, 1, 1, 1]$*. After bit-reversal permutation, the puncturing vector becomes* $\mathbf{w} = [0, 1, 0, 1, 0, 1, 1, 1]$*.*

By employing QUP in conjunction with the permutation $\pi_{\text{ord}} \circ \psi$, puncturing N_p positions corresponds to not using the N_p cells with the worst reliability levels for data storage. By placing high-resistance values (i.e., 1's) in the punctured locations, we increase the frequency of 1's in the codeword by $\frac{N_p}{2N}$.

Lemma 5.4. *For a punctured polar code length of* $N - N_p$*, the frequency of* 1*'s in the array is given by* $\frac{1}{2} + \frac{N_p}{2N}$*.*

Proof. Let i be chosen uniformly at random in $\{0, \ldots, N-1\}$, and let z_i be the corresponding codeword symbol. Let $\mathcal{B}_p$ be the set of punctured locations. The size of $\mathcal{B}_p$ is $N - N_p$. Then,

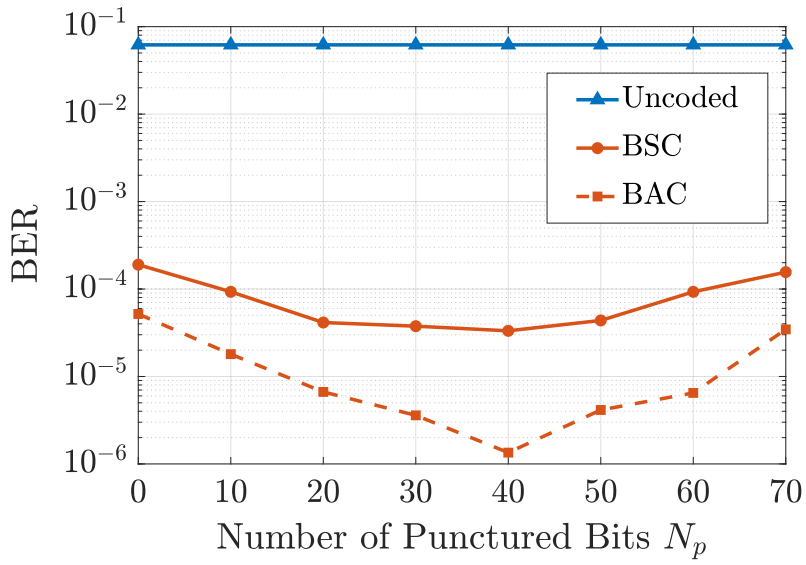


Figure 5.8: Punctured polar encoding over the (32×32) crossbar array for code rate 0.8.

we write

$$\begin{aligned} P(z_i = 1) &= P(i \in \mathcal{B}_p)P(z_i = 1 | i \in \mathcal{B}_p) \\ &\quad + P(i \notin \mathcal{B}_p)P(z_i = 1 | i \notin \mathcal{B}_p) \\ &= \frac{N_p}{N} + \frac{1}{2}\frac{N - N_p}{N} = \frac{1}{2} + \frac{N_p}{2N}. \end{aligned}$$

□

Example 5.6. *Figure 5.8 illustrates the systematic BER performance of QUP for a* (32×32) *array with* 35Ω *wire resistance,* $\pi = \pi_{ord} \circ \psi$*. We observe that the BER decreases as more bits* (N_p) *are punctured and as the frequency of* 1*'s increases. This gain is reversed for* $N_p > 40$ *and the BER increases as* N_p *and the codeword redundancy decrease. The BER is improved by a factor of* 5.7 *for a symmetric channel model and by a factor of* 38.5 *for an asymmetric channel model.*

5.3.4 Gaussian channel modeling

Instead of *hard decoding* as in the previous subsections, one can leverage the read value at the read output, and employ *soft decoding*. Towards that, we need a model of the read cells. However, the conditional read cell distributions vary, as a function of the bitline and the wordline indices. For simplicity, we model each conditional cell distribution as Gaussian with mean μ and variance σ^2, denoted as $\mathcal{N}(\mu, \sigma)$. To construct polar codes under the Gaussian modeling, we again use the algorithm in Remark 5.1, in conjunction with the following lemma.

Lemma 5.5. *The Bhattacharyya parameter of an (asymmetric) Gaussian channel, where* $W(y|0) \sim \mathcal{N}(\mu_0, \sigma_0)$*, and* $W(y|1) \sim \mathcal{N}(\mu_1, \sigma_1)$*, is given by*

$$Z = \int_{-\infty}^{\infty} \sqrt{W(y|0)W(y|1)}dy = \frac{\sqrt{2}\exp(-\frac{(\mu_0-\mu_1)^2}{4(\sigma_0^2+\sigma_1^2)})}{\sqrt{\frac{\sigma_0}{\sigma_1}+\frac{\sigma_1}{\sigma_0}}}. \tag{5.11}$$

Proof. The proof follows standard integration techniques and is omitted. □

Example 5.7. *Fig. 5.9 illustrates the systematic BER performance under Gaussian modeling for a* (32×32) *array with* $\pi = \pi_{ord} \circ \psi$*, and wire resistance* $\in \{30, \ldots, 35\}\Omega$ *(a higher range compared to Fig 5.6). Soft decoding clearly outperforms hard decoding (compare the BER for wire resistance of* 30 Ω*). Moreover, the same relative performance behavior between the permutations hold also in this case, with the bit-reversal permutation giving the best performance.*

5.4 Conclusion

In this paper, motivated by the sneak path problem in resistive memories, we studied polar coding over channels with different reliability levels. In particular, we argued that the

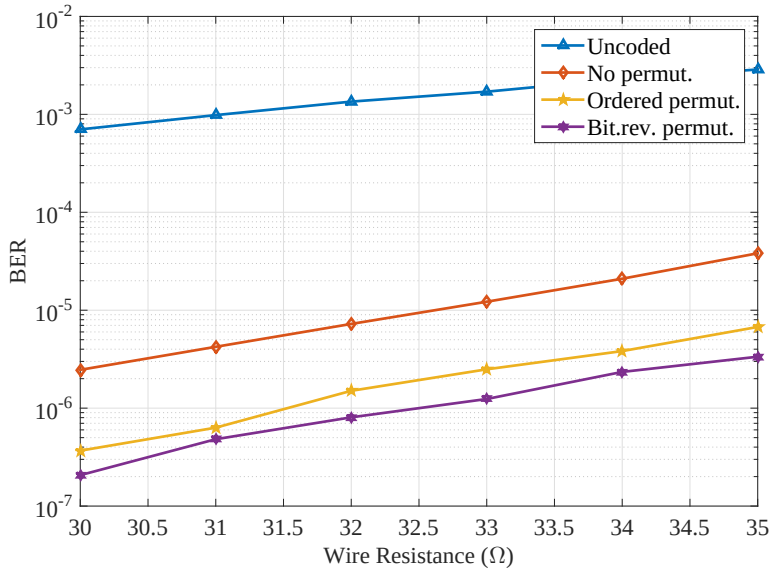


Figure 5.9: BER with Gaussian modeling, for a (32×32) crossbar, $k/n = 0.85$.

channels' ordering is important and proposed a channel ordering whose attractive performance was shown numerically. We then applied our framework to the sneak path problem in resistive memories. Simulation results on SPICE-like resistive crossbar simulator showed significant bit-error rate performance improvement, especially for low uncoded BER. Additionally, it is shown that biasing the frequency of high-resistance values in the array can mitigate the sneak path occurrences. Finally, using a more accurate binary asymmetric channel modeling, the BER is further reduced. We note that while in this work, we modeled each cell individually, the cost of such modeling can be amortized by using the same characterization over several crossbar arrays, which makes the model parameter costs justifiable from a practical perspective. Moreover, it is possible to cluster multiple cells together in a way to reduce the number of overall crossbar model parameters.

Appendix

5.4.1 Proof of proposition 5.1

Proof. Assume that $\epsilon_1 \geq \epsilon_2 \geq \epsilon_3 \geq \epsilon_4$. As for a BEC W, $Z(W) = 1 - I(W)$, we can equivalently study the Bhattacharyya parameters. By Lemma 5.3, it can be checked that we need only consider 3 different permutations.

- Case 1: $\pi_1 = [0, 1, 2, 3]$: using Lemma 5.2, we obtain

$$
\begin{aligned}
Z_{\pi_1}^{(0)} &= \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4 - \epsilon_1\epsilon_2 - \epsilon_3\epsilon_4 - (\epsilon_1 + \epsilon_2 - \epsilon_1\epsilon_2)(\epsilon_3 + \epsilon_4 - \epsilon_3\epsilon_4), \\
Z_{\pi_1}^{(1)} &= \epsilon_1\epsilon_2 + \epsilon_3\epsilon_4 - \epsilon_1\epsilon_2\epsilon_3\epsilon_4, \\
Z_{\pi_1}^{(2)} &= (\epsilon_1 + \epsilon_2 - \epsilon_1\epsilon_2)(\epsilon_3 + \epsilon_4 - \epsilon_3\epsilon_4), \\
Z_{\pi_1}^{(3)} &= \epsilon_1\epsilon_2\epsilon_3\epsilon_4.
\end{aligned}
$$

- Case 2: $\pi_2 = [0, 2, 1, 3]$: exchanging the roles of ϵ_1 and ϵ_2 in the previous case, we obtain

$$
\begin{bmatrix} Z_{\pi_2}^{(0)} \\ Z_{\pi_2}^{(1)} \\ Z_{\pi_2}^{(2)} \\ Z_{\pi_2}^{(3)} \end{bmatrix} = \begin{bmatrix} Z_{\pi_1}^{(0)} \\ \epsilon_1\epsilon_3 + \epsilon_2\epsilon_4 - \epsilon_1\epsilon_2\epsilon_3\epsilon_4 \\ (\epsilon_1 + \epsilon_3 - \epsilon_1\epsilon_3)(\epsilon_2 + \epsilon_4 - \epsilon_2\epsilon_4) \\ Z_{\pi_1}^{(3)} \end{bmatrix}.
$$

- Case 3: $\pi_3 = [0, 3, 1, 2]$: we obtain

$$
\begin{bmatrix} Z_{\pi_3}^{(0)} \\ Z_{\pi_3}^{(1)} \\ Z_{\pi_3}^{(2)} \\ Z_{\pi_3}^{(3)} \end{bmatrix} = \begin{bmatrix} Z_{\pi_1}^{(0)} \\ \epsilon_1\epsilon_4 + \epsilon_2\epsilon_3 - \epsilon_1\epsilon_2\epsilon_3\epsilon_4 \\ (\epsilon_1 + \epsilon_4 - \epsilon_1\epsilon_4)(\epsilon_2 + \epsilon_3 - \epsilon_2\epsilon_3) \\ Z_{\pi_1}^{(3)} \end{bmatrix}.
$$

Analysis: Note that $\sum_{i=0}^{3} Z_{\pi_1}^{(i)} = \sum_{i=0}^{3} Z_{\pi_2}^{(i)} = \sum_{i=0}^{3} Z_{\pi_3}^{(i)}$. Thus, $Z_{\pi_1}^{(1)} + Z_{\pi_1}^{(2)} = Z_{\pi_2}^{(1)} + Z_{\pi_2}^{(2)} = Z_{\pi_3}^{(1)} + Z_{\pi_3}^{(2)}$. Under π_2, it follows by Lemma 5.2 that $Z_{\pi_2}^{(0)} > Z_{\pi_2}^{(2)}$ and $Z_{\pi_2}^{(1)} > Z_{\pi_2}^{(3)}$. Moreover, we have

$$\begin{aligned}
Z_{\pi_2}^{(0)} - Z_{\pi_2}^{(1)} &= (1-\epsilon_1)(1-\epsilon_2)(\epsilon_3+\epsilon_4) + (\epsilon_1-\epsilon_3)(\epsilon_4-\epsilon_2) + (1-\epsilon_3)(1-\epsilon_4)(\epsilon_1+\epsilon_2) \geq 0, \\
Z_{\pi_2}^{(2)} - Z_{\pi_2}^{(3)} &= \epsilon_1\epsilon_3 + \epsilon_2\epsilon_4 - 2\epsilon_1\epsilon_2\epsilon_3\epsilon_4 \geq 0.
\end{aligned}$$

It follows that $Z_{\pi_2}^{(3)} \leq \min(Z_{\pi_2}^{(2)}, Z_{\pi_2}^{(1)}) \leq Z_{\pi_2}^{(0)}$. Similarly, one obtains

$$\begin{aligned}
Z_{\pi_1}^{(3)} &\leq \min(Z_{\pi_1}^{(2)}, Z_{\pi_1}^{(1)}) \leq Z_{\pi_1}^{(0)}, \\
Z_{\pi_3}^{(3)} &\leq \min(Z_{\pi_3}^{(2)}, Z_{\pi_3}^{(1)}) \leq Z_{\pi_3}^{(0)}.
\end{aligned}$$

Given the exact expressions of the Bhattacharyya parameters, we can determine the permutation satisfying (5.8), depending on the size of $\mathcal{I}$, i.e., depending on the code rate. Clearly, except for the rate $\frac{1}{2}$, all permutations satisfy (5.8). For example, if $R = 1/4$, we choose the same best channel corresponding to $Z_{\pi_1}^{(3)}$ for all permutations. Thus, we only need to analyze the rate $\frac{1}{2}, i.e., |\mathcal{A}| = 2$. We first compare permutations π_2 and π_3.

$$\begin{aligned}
Z_{\pi_2}^{(1)} - Z_{\pi_3}^{(1)} &= Z_{\pi_3}^{(2)} - Z_{\pi_2}^{(2)} = (\epsilon_1-\epsilon_2)(\epsilon_3-\epsilon_4) \geq 0, \\
Z_{\pi_3}^{(1)} - Z_{\pi_2}^{(2)} &= Z_{\pi_2}^{(1)} - Z_{\pi_3}^{(2)} \\
&= \epsilon_1\epsilon_2\epsilon_3 - \epsilon_3\epsilon_4 - \epsilon_1\epsilon_2 + \epsilon_1\epsilon_2\epsilon_4 + \epsilon_1\epsilon_3\epsilon_4 + \epsilon_2\epsilon_3\epsilon_4 - 2\epsilon_1\epsilon_2\epsilon_3\epsilon_4 \\
&= -\epsilon_3\epsilon_4(1-\epsilon_1)(1-\epsilon_2) - \epsilon_1\epsilon_2(1-\epsilon_3)(1-\epsilon_4) \\
&\leq 0.
\end{aligned}$$

It follows that $Z_{\pi_3}^{(1)} \leq \min(Z_{\pi_2}^{(1)}, Z_{\pi_2}^{(2)}) \leq Z_{\pi_3}^{(2)}$. Similarly, one obtains $Z_{\pi_3}^{(1)} \leq \min(Z_{\pi_1}^{(1)}, Z_{\pi_1}^{(2)}) \leq Z_{\pi_3}^{(2)}$. Therefore, $Z_{\pi_3}^{(1)}$ is the second best synthetic channel across all 3 permutations, which implies that π_3 satisfies (5.8). □

Chapter 6

Atomic storage with multi-version codes

6.1 Introduction

While previous chapters assume storage of fixed information, this chapter investigates the problem of storing changing information. More precisely, the problem of shared memory emulation is considered.

The emulation of a consistent fault-tolerant shared memory in a distributed asynchronous message-passing network has been an active area of research in distributed computing theory. Several applications demand concurrent and consistent access to the stored value by multiple writers and readers. In their celebrated paper [26], Attiya, Bar-Noy, and Dolev proposed a fault-tolerant algorithm (ABD algorithm) for emulating a shared memory that achieves atomic consistency (also known as linearizability) [108, 109]. ABD uses a replication-based storage scheme at the servers to attain fault tolerance. In [110], a two-layer replication based system is also presented, in which one layer is dedicated exclusively to metadata, and the other layer for storage. Variations of replication-based system algorithms appear in practical systems [23, 111].

Erasure coding is well known to lead to smaller storage costs compared to replication [112]. In erasure coding, each server stores a function of the value called a coded symbol. A

decoder can recover the value by accessing a number (called the *coding parameter*) of coded symbols. The number of bits used to represent a coded symbol is typically much smaller than the number of bits used to represent the value. However, introducing erasure coding in asynchronous networks raises several challenges that need careful management.

Example 6.1. *To illustrate some of the challenges raised from the use of erasure codes, we consider an asynchronous network with* $N = 7$ *servers. A writer uses an erasure code with coding parameter* 2 *for each version. Due to asynchrony, each reader observes different coded symbols corresponding to different versions. Figure 6.1 illustrates one particular scenario of read and write executions. A writer has finished writing 3 versions of a data object, and version 4 of the same object is being written to the servers. Some coded symbols may not arrive at their destinations, and this can be due to hardware failures, or asynchrony. Read client 1 and read client 2 observe different coded symbols from different versions. We highlight below some of the questions that should be addressed.*

- *Is it acceptable for read client* 1 *to return version* 1*, even though it observes but cannot decode later versions?*
- *What should read client* 2 *return, if any?*
- *How many versions should each server store?*
- *What coding parameters (e.g., code length, dimension, minimum distance) would be a good choice?*

Following [26], several papers [113, 114, 27, 28, 115, 29, 116] developed algorithms that use erasure coding instead of replication for fault-tolerance, with the goal of improving storage efficiency. Erasure-code based implementations of consistent data storage appear in [27, 115, 117, 118, 119] for crash failures. In [114, 28, 29] erasure codes are used in algorithms for implementing atomic memory that tolerate Byzantine failures. In [115, 120, 121], the authors

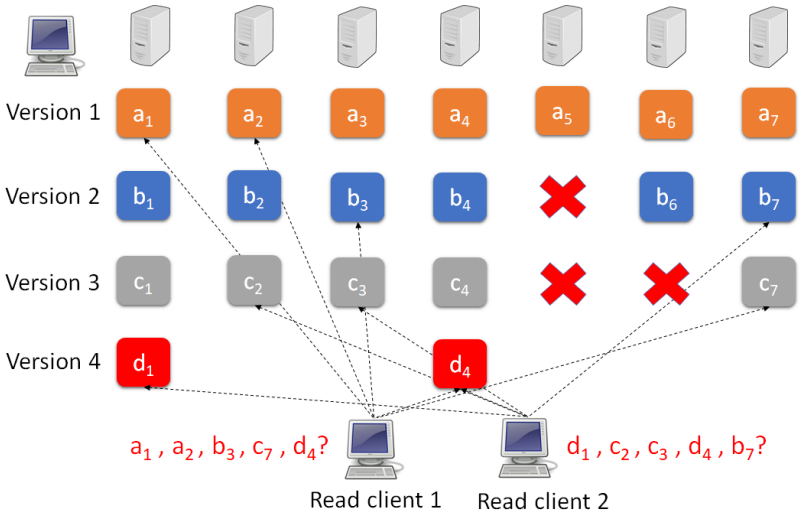


Figure 6.1: Erasure coding in asynchronous message-passing networks.

provide algorithms that permit repair of crashed servers, while implementing consistent storage. Bounds on the performance costs for erasure-code-based implementations appear in [122, 123, 119].

Since f-tolerant shared memory emulation is impossible with less than $2f+1$ servers [25], ABD algorithm uses the lowest number of servers, given by $2f+1$, to emulate an atomic shared object, with a total storage cost of $2f+1$ units. We assume throughout the paper that ABD uses $2f+1$ servers. On the other hand, erasure-code-based emulations use a higher number of servers $N > 2f+1$, such that the overall storage is reduced to less than that of ABD. References [122, 119] showed that the overall storage cost of emulation algorithms fundamentally grows with the server failures f and the number of concurrent writes at a point, denoted by μ. In particular, when μ is large compared to f, it is shown that the storage cost of ABD is optimal [122, 119]. On the other hand, a result that applies to most emulation algorithms is that the worst-case overall storage cost is lower bounded by $\frac{\mu N}{N-f+\mu-1}$ for $\mu \leq f+1$ [122], but its achievability remains an open problem. A main question of interest in this chapter is the following:

Question 1: Can coding across μ versions reduce the worst-case storage to $\frac{\mu N}{N-f+\mu-1}$?

Recently, multi-version code [30], answered this question in the affirmative for a coding-theoretic model. While the model in [30] differs from the model of interest in a number of ways (e.g., it lacks formal notions of write and read protocols, and decoding requirement is loosely based on the concept of atomicity), a key result is that one can *jointly* code μ versions using a simple yet asymptotically optimal scheme, which encodes each version *separately* and stores enough data of only the latest version observed in each server. In this work, we leverage the insights from [30] in the design of storage-efficient shared memory emulation algorithms, and analyze the merits of using multi-version codes.

System model: We study the emulation of shared atomic memory in an asynchronous message-passing network. Shared atomic storage can be emulated by composing individual atomic objects [108, 25]. Thus, we seek to implement only one atomic read/write memory object. We consider a network with fixed nodes and reliable channels. Nodes can have crash failures. An arbitrary number of client nodes can fail. Every new invocation at a client waits for a response of a preceding invocation at the same client (called well-formedness). The stored value is of size 1 unit. We assume fixed system parameters N, f, ν, where N is the number of servers, up to f server failures can be tolerated, $f \leq (N-1)/2$, and ν is called the *liveness parameter*, explained below.

System requirements and costs: We require the following safety and liveness properties, irrespective of the number of client failures.

• *Atomicity:* The algorithm must emulate a shared atomic read-write object that supports concurrent access by the clients in the system, where the observed global external behaviors "look like" the object is being accessed sequentially [109].

• *ν-concurrency wait-freedom:* We require a write operation to terminate if the number of server failures in the execution is bounded by f, and a read operation to terminate if the number of server failures is bounded by f and the number of concurrent writes with the read

is less than ν. We call such liveness property ν-concurrency wait-freedom. Note that the number of concurrent writes at any point in an execution is allowed to be arbitrarily large, and not limited by the liveness parameter ν.

Trading-off between consistency and liveness has been an active research and engineering topic, with different algorithms and implementations proposed. For instance, BigTable, a distributed storage system by Google, prioritizes safety over liveness [7], whereas Amazon's Dynamo does not compromise liveness at the expense of providing a weaker *eventual consistency* [22]. The algorithms in [27, 121] provide ν-concurrency wait-freedom. In practice, our algorithms do not need to know the exact worst-case concurrency level over all executions. Instead, it can use ν as an estimate of the concurrency, say, for 90% of the read operations. If a reader is not able to return the value, it can re-try and complete the read once the number of concurrent writes reduces to less than ν.

We consider storage and communication costs defined below.

• *Storage cost:* The storage cost is defined to be the total amount of data stored across all servers, at a point during the execution of an algorithm. We assume that metadata (e.g., timestamp) is of negligible size compared to the stored value, and is hence ignored. The *steady-state* storage cost corresponds to a steady-state point, for which there is no ongoing write, and the completed writes have delivered their messages to all live servers. The *worst-case* storage cost is the largest storage cost among all points in all executions.

• *Communication cost:* The read (resp. write) communication cost is the largest amount of the total transmitted data, among all read (resp. write) operations of all executions. Metadata cost is again neglected.

Related work

Assume that a read operation is concurrent with several writes, including failed writes, i.e., writes invoked by failed writers. Then, in erasure coding, it is possible that the reader obtains information of different values, but does not have a sufficient number of coded symbols to decode and return any value. In order to handle the difficulty brought by concurrent writes, several techniques and liveness guarantees have been proposed, described below.

Algorithms in [27, 28, 29] store history of received coded symbols, and hide ongoing writes from a read until enough number of coded symbols have been propagated to the servers. However, the worst-case storage cost grows unbounded with the number of concurrent writes. Algorithms in [115, 117, 118] propagate full replicas at a first phase before performing erasure coding at a second phase. In SCCK [119] a writer communicates full replicas, and each server, upon receipt of a full replica, either stores a coded symbol or the full replica depending on its state, leading to a worst-case storage of $2N$ units. ORCAS-A [115] is similar to our algorithms in that the server stores only the latest version. However, the read operation uses reader registration to be explained later. The algorithm in [117] achieves the lowest overall storage in the steady state at the expense of costly write communication on the order of N^2 units. In [118], replicas of all ongoing writes are stored in an edge layer of servers, and coded symbols are stored in a back-end layer. The total worst-case storage can be unbounded even with garbage collection in the edge layer.

The strongest liveness guarantee for read operations is *wait-freedom*, which guarantees that all operations invoked by correct clients eventually terminate despite the concurrent invocations of other clients, implemented by *reader registration* in [114, 115, 117, 118]. That is, a read operation registers itself at the servers it contacts, and keeps receiving symbols from them until successful recovery of a value. However, the amount of communication of a read operation can be unbounded and depends on concurrent writes with the read. In

contrast, our algorithms guarantee liveness if the number of concurrent writes with a read is smaller than ν, and only uses 2 or 3 phases of communications. A similar liveness setting is found in CASGC [27], but we will demonstrate the advantage of our algorithms in Section 6.7 in terms of the storage cost and protocol simplicity. SCCK [119] satisfies the *finite-write termination* liveness, namely, in every execution with finitely many writes, every read operation invoked by a non-failed reader terminates [124]. In HGR [29], read operations satisfy *obstruction-freedom*, that is, a read returns if there is a sufficiently long period during the read when no other operation takes steps.

Contributions of the chapter

We summarize our contributions below.

- We first propose a single-writer multi-reader (SWMR) atomic shared memory emulation algorithm, Algorithm 6.1. Algorithm 6.1 can be regarded as non-trivial extension of the ABD algorithm. Write operations complete in *one-round* of communication with the servers. We illustrate through examples the challenges and intuitions behind the correctness of Algorithm 6.1.
- Based on Algorithm 6.1, we propose several atomic multi-writer multi-reader (MWMR) algorithms: Algorithm 6.3, its variant Algorithm 6.3-A for a specific setting to be specified, and Algorithm 6.4.
- Our algorithms, except Algorithm 6.4, have a simple structure reminiscent of the ABD algorithm as servers only need to store information associated with the latest value they observe, without any logs or history. We call this *in-place update*.
- All algorithms satisfy ν-concurrency wait-freedom.
- The write and read operations only consist of 2 or 3 communication phases. We analyze

	Alg. 6.3-A	Alg. 6.3	Alg. 6.4	ABD [26]	CASGC [27]	SCCK [119]
Worst-case storage	$\frac{N}{k}$	$\frac{2f+1}{+\frac{(k-1)(\nu+k-1)}{k}}$	$\frac{N}{k}+\frac{wN}{N-2f}$	$2f+1$	$\frac{(\nu+w_{CASGC})N}{N-2f}$	$2N$
Steady-case storage	$\frac{N}{k}$	$\frac{N}{k}$	$\frac{N}{k}$	$2f+1$	$\frac{\nu N}{N-2f}$	$\frac{N}{N-2f}$
Write comm.	$\frac{N}{k}$	$\frac{2f+1}{+\frac{(k-1)(\nu+k-1)}{k}}$	$\frac{N}{k}+\frac{f}{N-2f}$	$2f+1$	$\frac{N}{N-2f}$	N
Read comm.	$\frac{2N}{k}$	$\frac{2(2f+1}{+\frac{(k-1)(\nu+k-1)}{k})}$	$2(\frac{N}{k}+\frac{f}{N-2f})$	$2(2f+1)$	$\frac{2N}{N-2f}$	$2N$
Write concurr.	$\leq \nu-1$	UB	UB	UB	UB	UB
Read liveness	ν-concurr.	ν-concurr.	ν-concurr.	wait-freedom	ν-concurr.	FW termination
In-place update	yes	yes	yes	yes	no	no

Table 6.1: Comparison of the storage and communication (comm.) costs for fixed N, f, ν. Here $k = \lceil \frac{N-2f}{\nu} \rceil$ satisfies (6.1). Assume ABD uses only $2f+1$ servers, CASGC uses parameters $(k_{CASGC}, \delta) = (N-2f, \nu-1)$, and SCCK uses coding parameter $k_{SCCK} = N-2f$. w and w_{CASGC} are parameters associated with the number of ongoing writes at a point, which can be arbitrarily large. UB means unbounded. ν-concurrency (concurr.) means ν-concurrency wait-freedom.

the storage and communication costs of our algorithms. In particular, we prove that Algorithm 6.3-A has an asymptotic optimal worst-case storage cost matching the lower bound in [122], which answers Question 1 affirmatively. We also show that for all parameters Algorithm 6.3-A is better than known algorithms by a factor of up to 2.

The storage and communication costs of our MWMR algorithms, ABD, and two previous coding-based algorithms are shown in Table 6.1. The two coding-based algorithms are listed because they employ somewhat similar protocol structures as ours. Detailed discussions can be found in Section 6.7.

Organization: In Section 6.2, we introduce useful definitions and lemmas. The SWMR and MWMR algorithms are presented and analyzed in Sections 6.3, 6.4 and 6.6. Detailed comparisons with previous algorithms and conclusions are drawn in Section 6.7.

6.2 Preliminaries

In this section, we introduce necessary definitions, the principles of erasure codes and multi-version codes, and a lemma that is used to prove atomicity throughout the paper.

We consider algorithms that tolerate f failures out of N servers, with a liveness parameter ν. We define a *quorum* Q to be a subset of the server nodes, such that its size satisfies $|Q| \geq N - f$. It follows that for any two quorums Q_1, Q_2, we have $|Q_1 \cap Q_2| \geq N - 2f$. We assume that every data value comes from a finite set $\mathcal{V}$. In this paper we refer to $\log_2 |\mathcal{V}|$ as 1 unit. We also arbitrarily choose v_0 from $\mathcal{V}$ to be a default value. Different versions of the data value are associated with different *tags*.

Erasure codes: Let Φ be an (N, k) maximum distance separable (MDS) code (e.g. Reed-Solomon code) that takes a value in $\mathcal{V}$ as input and outputs N coded symbols in $\mathcal{W}$, where $\log_2 |\mathcal{W}| = \frac{1}{k} \log_2 |\mathcal{V}|$, corresponding to $\frac{1}{k}$ unit. Any k of the N coded symbols suffice to decode the value. We say that a tag t is *decodable* if the read operation collects at least k distinct coded symbols with tag t.

Multi-version codes: An (N, f, ν) multi-version code [30] is an erasure code that encodes a total of ν value versions. Every server gets an arbitrary non-empty subset of the ν versions and stores a function of them without knowing other servers' subsets. The decoding requirement is that from any $N - 2f$ servers, the common version among these servers with the highest tag, or some newer version, can be recovered.

The intuition of our coding strategy is explained informally below. A naive code can encode each version separately with k being $N - 2f$, which is the size of the intersection of two quorums. It needs to store all the ν concurrent versions at the servers to ensure a correct read. The benefit of erasure code is that the worst-case storage $\frac{\nu N}{N-2f}$ is smaller than that of

ABD, $2f+1$, when, for example, $\nu < 2f+1, N \gg f$.

We intend to *jointly* code ν concurrent versions and further reduce the storage. Motivated by the asymptotic optimal multi-version code in [30], we encode each version *separately* using $k = \lceil \frac{N-2f}{\nu} \rceil$, and store only the version of the highest tag at each server, resulting in a worst-case storage of $N/\lceil \frac{N-2f}{\nu} \rceil$ (see details in Algorithm 6.3-A). We will show by the Pigeonhole principle that this modification of k, together with careful handling of the decodable versions, guarantees correctness if a read is concurrent with less than ν writes. For fixed N, f, ν, compared to naive code, our code reduces the storage size by a factor of up to 2. Moreover, the algorithm can employ an in-place update at the servers.

Example 6.2. *Let $N = 11, f = 2, \nu = 3$. ABD uses the minimum number of servers, $2f+1 = 5$, and the storage is 5. The naive code and our Algorithm 6.3-A both use $N = 11$ servers, and have the worst-case storage of $\frac{\nu N}{N-2f} = 4.71$ and $N/\lceil \frac{N-2f}{\nu} \rceil = 3.67$, respectively.*

Remark 6.1. *For fixed system parameters N, f, ν, the coding parameter is $k = \lceil \frac{N-2f}{\nu} \rceil$. In fact, we can use only $\widetilde{N} = (k-1)\nu + 2f + 1 \leq N$ servers and do not use the remaining, while keeping the same the coding parameter $\lceil \frac{\widetilde{N}-2f}{\nu} \rceil = \lceil \frac{N-2f}{\nu} \rceil$. Throughout the paper, we will use the reduced number of servers $N = \widetilde{N}$, and thus the integer k satisfies*

$$k = \lceil \frac{N-2f}{\nu} \rceil = 1 + \frac{N-(2f+1)}{\nu}. \tag{6.1}$$

Next we state a lemma of a sufficient condition for atomicity, which will be used to prove correctness for our algorithms.

Lemma 6.1 (Lemma 13.16 of [25])**.** *Let β denote a sequence of actions of the external interface of a read/write object. Suppose β is well formed for each client and contains no incomplete operations. Let Π be the set of all operations in β. A sufficient condition for atomicity of β is: there exists a partial ordering $\prec$ of all the operations in Π, satisfying the following properties:*

(1) If the response for π_1 precedes the invocation for π_2 in Π, then it cannot be the case that $\pi_2 \prec \pi_1$.

(2) If π_1 is a write operation in Π and π_2 is any operation in Π, then either $\pi_1 \prec \pi_2$ or $\pi_2 \prec \pi_1$.

(3) The value returned by each read operation is the value written by the last preceding write operation according to $\prec$ (or the default value, if there is no such write).

We now define the partial ordering that we use in conjunction with Lemma 6.1 in the correctness proofs. We define tags of operations for each algorithm in its corresponding section.

Definition 6.1 (Partial Ordering $\prec$)**.** *Consider an execution α and consider two operations π_1, π_2 that complete in α. Let $T(\pi_1)$ and $T(\pi_2)$ respectively denote the tags of operations π_1 and π_2. Then we define the partial ordering on the operations as: $\pi_1 \prec \pi_2$ if*

(1) $T(\pi_1) < T(\pi_2)$; or

(2) $T(\pi_1) = T(\pi_2)$ if π_1 is a write and π_2 is a read.

6.3 Single-writer multi-reader algorithm

6.3.1 Algorithm description

In this section, we describe our SWMR algorithm (See Algorithm 6.1). The different phases of the write and read protocols are executed sequentially. In each phase, a client sends messages to servers to which the non-failed servers respond. Termination of each phase depends on getting responses from at least one quorum.

The write protocol has one phase, where a tag is incremented and the associated value is encoded using an (N, k) erasure code and propagated to at least a quorum. For a tag-value pair $r = (t, v)$, we write $tag(r) = t$.

The read protocol is carried out in two phases, one for getting values, and one for writing back. The *write-back* phase is triggered whenever the reader can recover a certain value v from the responses and safely return it. The read returns a value with tag t that is decodable and also satisfies some conditions, as specified by Lines 14 through 18 in Algorithm 6.1. The intuition behind Line 15 is that coded symbols corresponding to any *old* write operation can only be stored in at most f servers at any point of the execution. Line 16 ensures that the returned value does not violate atomicity. The read protocol has an *_abort_* internal action. In case the *_abort_* action is invoked, the client does not return and the operation which invokes it does not terminate. The action indicates to the reader that the liveness bound is violated causing the read not to terminate. From the viewpoint of a practical storage system, we note that the *_abort_* action can prompt the reader to invoke a read request again; however, we do not formally incorporate such an invocation in the description of Algorithm 6.1. We comment on the possibility of invoking multiple rounds of read briefly in Section 6.3.3. If $\nu \geq N - 2f$, the code used in Algorithm 6.1 reduces to the replication-based ABD algorithm. We have $k = 1$, i.e., every server stores a full replica. Moreover, the reader recovers the value corresponding to the highest tag observed among the responses and the value satisfies the condition in Line 16. Throughout the section, we assume that $\nu \geq 2$ and $k > 1$. Algorithm 6.1 possess the *in-place update* property: a server only needs to store information associated with the value of the highest observed tag. This property is reminiscent of the ABD algorithm. However, unlike ABD and because of the use of erasure codes, a careful handling of the read protocol is required in order to satisfy atomicity. Unlike most erasure-coded atomic shared memory emulations, Algorithm 6.1 does not store any additional tag at the servers to keep track of the latest finished write. This allows Algorithm 6.1 to enjoy a simple *one-round* write protocol without compromising atomicity. From these perspectives, Algorithm 6.1 can be regarded as a non-trivial extension of the ABD algorithm. We illustrate below through examples the intuition behind the read protocol of Algorithm 6.1. A formal analysis of the correctness of Algorithm 6.1 follows in the subsequent subsections.

Algorithm 6.1 SWMR algorithm

Write client protocol

1: *state variable*: Tag t, $t \in \mathbb{N}$. Initially, $t \leftarrow 0$.
2: **function** WRITE(v)
3: $t \leftarrow t + 1$.
4: Let $(y_1, y_2, \ldots, y_N) = \Phi(v)$.
5: || **for** $s \in \{1, 2, \ldots, N\}$
6: Send $put(t, y_s)$ to server s,
7: **wait until** receive acknowledgments from a quorum.
8: **end function**

Read client protocol

9: **function** READ( )
10: || **for** $s \in \{1, 2, \ldots, N\}$
11: Send request $get()$ to server s,
12: **wait until** receive responses from a quorum.
13: Let R be the set of response pairs.
14: Let T be the set of decodable tags t occurring in R such that
15: (i) t has at least $f + 1$ coded symbols, or
16: (ii) the number of tags strictly higher than t is at most ν.
17: **if** $T \neq \emptyset$ **then**
18: Let $t = \max(T)$ and v its value.
write-back phase
19: Let $(y_1, y_2, \ldots, y_N) = \Phi(v)$.
20: || **for** $s \in \{1, 2, \ldots, N\}$
21: Send $put(t, y_s)$ to server s,
22: **wait until** receive acknowledgments from a quorum.
23: **return** v.
24: **else**
25: $\_abort\_$.
26: **end if**
27: **end function**

Server s protocol

28: *state variable* at server s: A pair (t, y), where $t \in \mathbb{N}$, $y \in \mathcal{W}$.
29: Initially, server s stores $(t, y) = (0, y_s)$, where y_s is the sth component of $\Phi(v_0)$.
30: **Upon receipt of *get*() do**
31: respond with (t, y_s).
32: Upon receipt of $put(t_{new}, y_{new})$ **do**
33: If $t_{new} > t$, then set $t \leftarrow t_{new}$ and $y \leftarrow y_{new}$. In any case respond with acknowledgement.

Example 6.3. *Let $N = 7, f = 2, \nu = 2$. Then, Algorithm 6.1 uses $k = 2$. For value v_i, we denote by $y_{i,s}$ the coded symbol sent to server $s \in \{1, \ldots, 7\}$.*

Assume the writer has finished writing successively the values v_1, v_2, v_3 such that servers 1 and 2 store $v_{1,1}$ and $v_{1,2}$, respectively. Servers in $\{3, \ldots, 7\}$ constitute the quorum replying to the write operations of v_2 and v_3. The writer then starts writing v_4 such that server 5 updates its content to $v_{4,5}$.

Consider a first read operation that started after the write of v_2 completed and before the write of v_3 started. The servers in $\{1, \ldots, 5\}$ constitute the read quorum and the reader collects $\{v_{1,1}, v_{1,2}, v_{2,3}, v_{3,4}, v_{4,5}\}$. The read is concurrent with 2 write operations. Clearly, only the value v_1 can be recovered by the reader. However, returning v_1 violates atomicity. Indeed, neither Line 15 nor Line 16 in Algorithm 6.1 are satisfied.

Consider a second read operation that started after the write of v_2 completed and before the write of v_3 started. The servers in $\{1, \ldots, 5\}$ constitute the read quorum and the reader collects $\{v_{1,1}, v_{1,2}, v_{2,3}, v_{2,4}, v_{3,5}\}$. The read is concurrent with 1 write operation. Both values v_1 and v_2 can be recovered by the reader. However, to guarantee atomicity, only v_2 can be returned. In Algorithm 6.1, while both v_1 and v_2 satisfy Line 16, only v_2 is allowed to be returned by virtue of Line 18.

Consider a third read operation that started after the write of v_2 completed and before the write of v_3 started. The servers in $\{3, \ldots, 7\}$ constitute the read quorum and the reader collects $\{v_{2,3}, v_{2,4}, v_{4,5}, v_{2,6}, v_{3,7}\}$. The read is concurrent with 2 write operations. Only v_2 can be recovered by the reader and it can be safely returned. In Algorithm 6.1, v_2 satisfies Line 15 and is returned by the read protocol.

It is worth noting that while the first read and the third read above are both concurrent with 2 write operations, only one of them completes. We investigate formally the read liveness guarantees in Theorem 6.3 and Remark 6.5.

6.3.2 Safety properties

In this section, we present safety properties satisfied by Algorithm 6.1. We first show in Lemma 6.2 that there always exists some value that can be recovered from the servers during the execution of Algorithm 6.1. Then, we show that Algorithm 6.1 emulates an atomic shared memory in Theorem 6.1, using Lemma 6.1 on the partial ordering $\prec$ in Definition 6.1. To show this, we prove Lemmas 6.3, 6.4, 6.5 and 6.6. We finally prove a safety property in Lemma 6.7 that will be used in Section 6.3.3 where we describe liveness properties.

We now define the *tag of an operation*. In the definition, we use the fact that every completed read or write operation propagates *put* messages to the servers with a particular tag. Note that the tag of an operation is defined for every operation that completes in an execution. Furthermore, the tag is not defined for read operations that abort, since these operations do not propagate a *put* message and are not considered complete.

Definition 6.2 (Tag of an operation π)**.** *Let π be an operation in an execution α. The tag of operation π is defined to be the tag associated with the* put *messages that the operation propagates to the servers.*

Lemma 6.2 (Persistence of data)**.** *The value written by either the latest complete write or a newer write is available from every set of at least $N - f$ servers.*

Proof. Let Q_w denote the quorum of servers that replied to the *put* message of the latest finished write π_w (if no write has finished in the execution, Q_w can be any quorum from the set of live servers, and $T(\pi_w)$ is assumed to be 0). Thus, each server in Q_w has a tag that is at least as large as $T(\pi_w)$. Because there is at most one single ongoing incomplete write operation for a single writer, the number of tags in Q_w is at most 2. Thus, one of the tags, say t, appears in at least $\lceil \frac{|Q_w|}{2} \rceil \geq \lceil \frac{N-2f}{\nu} \rceil = k$ servers and $t \geq T(\pi_w)$. Therefore, the value corresponding to t is available in the system. □

Lemma 6.3. *Consider any execution α of the algorithm and consider a write or read operation π_1 that completes in α. Let $T(\pi_1)$ denote the tag of the operation π_1 and let Q_1 denote the quorum of servers from which responses are received by π_1 to its* put *message. Consider a read operation π_r in α that is invoked after the termination of the write operation π_1. Suppose that the read π_r receives responses to its* get *message from a quorum Q_r. Then,*
(1) Every server s in $Q_1 \cap Q_r$ responds to the get *message from π_r with a tag that is at least as large as $T(\pi_1)$.*
(2) If, among the responses to the get *message of π_r from the servers in $Q_1 \cap Q_r$, the number of tags is at most ν, then there is some tag t such that (i) $t \geq T(\pi_1)$, and (ii) from the servers in $Q_1 \cap Q_r$, operation π_r receives at least k responses to its* get *message with tag t.*

Proof. Proof of (1). Consider any server s in $Q_1 \cap Q_r$. From the server protocol we note that at every point after the reception of π_1's *put* message, it stores a tag that is no smaller than $T(\pi_1)$. Thus it responds to the *get* message with a tag that is at least as large as $T(\pi_1)$.

Proof of (2). Among the responses from $Q_1 \cap Q_r$, the read π_r receives at most ν different tags. By the Pigeonhole principle, there is at least one tag t such that it receives at least $\lceil \frac{|Q_1 \cap Q_r|}{\nu} \rceil$ responses with t. Since $|Q_1 \cap Q_r| \geq N - 2f$, we infer that the operation π_r receives at least $\lceil \frac{N-2f}{\nu} \rceil = k$ responses with tag t. From (1), we infer $t \geq T(\pi_1)$ to complete the proof. □

Lemma 6.4. *Consider an execution α of Algorithm 6.1. Let π_1 be a write or read operation that completes in α, and let π_2 be a read operation that completes in α. Let $T(\pi_1)$ denote the tag of operation π_1 and $T(\pi_2)$ denote the tag of operation π_2. If π_2 begins after the termination of π_1, then $T(\pi_2) \geq T(\pi_1)$.*

Proof. Let Q_1 denote the quorum that responds to π_1's *put* message. Let Q_2 denote the quorum that responds to π_2's *get* message. Let $Q = Q_1 \cap Q_2$. We prove the claim by

contradiction. Suppose that $T(\pi_2) < T(\pi_1)$. Either Line 15 or Line 16 should be satisfied so that π_2 completes.

From (1) in Lemma 6.3, we infer that every server in Q responds to the *get* message of π_2 with a tag that is at least as large as $T(\pi_1)$. Because $T(\pi_1) > T(\pi_2)$, the value returned by π_2 must have been obtained using the responses from servers in $Q_2 \backslash Q_1$. Thus $|\{u = T(\pi_2) | u = tag(r), \text{for some } r \in R\}| \leq |Q_2 \backslash Q_1| \leq N - (N - f) = f$. Thus Line 15 cannot be satisfied. Assume Line 16 is satisfied. Because every server in Q responds with a tag that is at least as large as $T(\pi_1)$, which is greater than $T(\pi_2)$, and because $Q \subseteq Q_2$, we infer that the number of distinct response tags from Q is at most ν. Property (2) in Lemma 6.3 implies that there exists a tag $t \geq T(\pi_1)$ that appears in at least k responses from Q. From the read protocol, we infer that the tag of the read operation should be at least as large as t. That is, $T(\pi_2) \geq t$. But we know that $t \geq T(\pi_1) > T(\pi_2)$, which is a contradiction. □

Remark 6.2. *There can be multiple values that can be safely returned by a read operation. Indeed, as can be inferred from the proof of Lemma 6.4, any value satisfying Line 15 in Algorithm 6.1 can be returned safely, even if a higher tag can be recovered by the read operation.*

Remark 6.3. *If $k > f$, the reader protocol is simplified so that Lines 15 and 16 are omitted. Indeed, any decodable tag has at least $k \geq f + 1$ coded symbols, which automatically satisfies Line 15 in Algorithm 6.1 and can be safely returned because of the previous remark.*

Lemma 6.5. *Consider an execution α of Algorithm 6.1. Let π_1 be a write or read operation that completes in α, and π_2 be a write operation that completes in α. Let $T(\pi_1)$ denote the tag of operation π_1 and $T(\pi_2)$ denote the tag of operation π_2. If π_2 begins after the termination of π_1, then $T(\pi_2) > T(\pi_1)$.*

Proof. The result follows from the single writer assumption and the fact that the writer increments its tag at the beginning of a new write in Line 3. □

Lemma 6.6. *Let π_1, π_2 be write operations that terminate in an execution α of Algorithm 6.1. Then, $T(\pi_1) \neq T(\pi_2)$.*

The proof follows because of the SWMR setting and Lemma 6.5. The following theorem states the main result on atomicity, and the proof follows from Definitions 6.1 and 6.2, combined with Lemmas 6.1, 6.4, 6.5, and 6.6.

Theorem 6.1. *Algorithm 6.1 emulates an atomic read-write object.*

Proof. Let β denote a sequence of actions of the external interface of a read/write object satisfying the conditions in Lemma 6.1. Let Π be the set of operations in β. Note that because Π consists of operations that complete, every operation in π has a tag. Definition 6.1 imposes a partial order $\prec$ on the set Π. Let π_1 and π_2 be two operations in Π. Let $T(\pi_1)$ denote the tag of operation π_1 and $T(\pi_2)$ denote the tag of operation π_2. We show that operations π_1 and π_2 satisfy Properties **(1)**, **(2)** and **(3)** of Lemma 6.1.

Proof of **(1):** We consider two cases. First, we consider the case where π_2 is a read. Second, we consider the case where π_2 is a write. In the first case, if π_2 is a read, then Lemma 6.4 implies $T(\pi_2) \geq T(\pi_1)$. As per Definition 6.1, we infer that it is not the case that $\pi_2 \prec \pi_1$. In the second case, if π_2 is a write, then Lemma 6.5 implies that $T(\pi_2) > T(\pi_1)$. As per Definition 6.1, we infer that it is not the case that $\pi_2 \prec \pi_1$.

Proof of **(2):** Recall that π_1 is a write operation. First consider the case where π_2 is a read operation. There are only two possibilities: either $T(\pi_1) > T(\pi_2)$, then $\pi_2 \prec \pi_1$. Otherwise, $\pi_1 \prec \pi_2$. Now consider the case where π_2 is write operation. From Lemma 6.6, either $T(\pi_1) > T(\pi_2)$, which implies $\pi_2 \prec \pi_1$, otherwise, $T(\pi_2) > T(\pi_1)$, which implies $\pi_1 \prec \pi_2$. This completes the proof of **(2)**.

Proof of **(3):** Consider a read operation π_1 that returns value v. Let $T(\pi_1)$ denote the tag of the read operation.

We first consider the case of $T(\pi_1) = 0$. By Lemma 6.4 and since the tag of a write operation is greater or equal to 1 by Line 3, there are no writes preceding π_1. By the read protocol, the read receives at least k responses with codeword symbols obtained by applying the encoding function Φ on the default value v_0, decodes and returns v_0.

We now consider the case of $T(\pi_1) > 0$. From the server protocol, we note that a write operation π_2 encoded a value w with codeword symbols corresponding to tag $T(\pi_1)$. From our definition of the tag of an operation, we note that the tag of the operation π_2 is equal to $T(\pi_1)$. From our definition of partial order, we note that π_2 is the last write operation that precedes π_1 as per $\prec$. To complete the proof, we need to show that π_1 returns w. From the read protocol, we note that the read operation receives at least k messages from k distinct servers with tag $T(\pi_1)$ and the corresponding codeword symbols. From the write and server protocols, we infer that these k codeword symbols were obtained by applying the (N, k) code to w. Therefore, the reader decodes w and returns. This completes the proof. □

Next, we use Lemma 6.3 to show a safety property that will be used later.

Lemma 6.7. *Consider any execution α of Algorithm 6.1. Let π_r denote a read operation in α that receives a quorum Q_r of responses to its* get *message. Let $\mathcal{S}$ denote the set of all writes that terminate before the invocation of π_r in α. If $\mathcal{S}$ is non-empty, let t_w denote the largest among the tags of the operations in $\mathcal{S}$. If $\mathcal{S}$ is empty, let $t_w = 0$.*

If the number of writes concurrent with the read π_r is smaller than ν, then there is some tag t such that

(1) $t \geq t_w$,

(2) π_r receives at least k responses to its get *message with tag t, and*

(3) the number of tags that are higher than t is smaller than ν.

Proof. We first argue that among the responses to the read's *get* message from Q_r, the reader

gets fewer than ν distinct response tags that are larger than t_w. By definition of the tag t_w, if the read receives a tag t that is larger than t_w, then the tag t corresponds to the tag of a write operation π that is concurrent with π_r. Since the number of writes that are concurrent with the read is smaller than ν, the read receives fewer than ν distinct tags that are larger than t_w.

We assume that $\mathcal{S} \neq \emptyset$. The case $\mathcal{S} = \emptyset$ can be treated in a similar way. Consider the write operation π_w in $\mathcal{S}$ whose tag is t_w. Let Q_w denote the quorum of servers from which responses were received by the writer to the *put* message of π_w. Property (1) in Lemma 6.3 implies that every server s in $Q_w \cap Q_r$ responds with a tag that is at least as large as t_w. Note that we have already shown that π_r receives fewer than ν distinct tags to its *get* message that are larger than t_w. That is, among the responses received by π_r from the servers in $Q_w \cap Q_r$ to its *get* message, there are at most $\nu - 1$ distinct tags that are larger than t_w. Since some of the servers in $Q_w \cap Q_r$ may respond with tag t_w, we infer that among the responses received by π_r from servers in $Q_w \cap Q_r$, there are at most ν distinct tags. Property (2) of Lemma 6.3 implies the statement of the lemma, since it implies that there is at least one tag t which is no smaller than t_w such that at least k responses with tag t are received by π_r. It can be seen that (3) holds. □

Remark 6.4. *A write operation π_1 needs to be counted as "concurrent" with the read π_r only if its tag $T(\pi_1)$ is larger than t_w defined in Lemma 6.7, and π_1 starts before the termination of π_r. In particular, suppose π_1 is a failed write operation, then it is not counted as concurrent with the read unless $T(\pi_1) > t_w$.*

6.3.3 Liveness properties

We first state the ν-concurrency wait-freedom of Algorithm 6.1 in Theorems 6.2 and 6.3. Then we show in Theorem 6.4 that Algorithm 6.1 satisfies finite-write termination if the

read operation is allowed to take several phases.

Theorem 6.2 (Termination of writes)**.** *Consider any fair execution α of SWMR Algorithm 6.1 where the number of server failures is at most f and the write client does not fail. Then, every write operation terminates in α.*

Proof. Consider a fair execution α and let π denote an arbitrary write operation in α. Consider a non-failing server s in α. In a fair execution, eventually server s receives the *put* message of operation π. From the server protocol, we note that server s responds to the *put* message of the write with an acknowledgement. Therefore, in a fair execution, eventually operation π receives an acknowledgement from every non-failing server s. Since the number of server failures is no bigger than f, there is at least one quorum Q consisting entirely of non-failing servers. Therefore, the write operation π receives acknowledgments from at least one quorum of servers. From the write protocol, we infer that operation π terminates. □

Theorem 6.3 (Termination of reads)**.** *Consider any fair execution α of Algorithm 6.1 where the number of server failures is at most f. Consider any read operation that is invoked at a non-failing client in α. If the number of writes concurrent with the read is strictly smaller than ν and the read client does not fail, then the read operation completes in α.*

Proof. Consider a fair execution α and consider a read operation π_r in α such that the number of writes that are concurrent with π_r is smaller than ν. We show that π_r completes in α. Since the number of server failures in α is at most f, we note that π_r receives responses from a quorum Q_r of servers to its *get* message. To show completion of π_r, we show that

(1) there is a tag that is decodable, and
(2) it satisfies either Line 15 or Line 16, and
(3) the *write-back* phase terminates.

Since α is an execution where there are at most f failures, there is at least one quorum

consisting entirely of non-failing servers. Since every server eventually responds to the *get* message of the read, the read gets responses from some quorum of servers Q_r. Lemma 6.7 implies that there is a tag $\bar{t}$ such that the read π_r receives at least k responses with tag $\bar{t}$. In other words, Lemma 6.7 implies (1).

We now show (2). We distinguish two cases. If Line 15 is satisfied, then (2) follows. Otherwise, by virtue of Lemma 6.7 (iii), Line 16 is satisfied and (2) follows.

The termination of the *write-back* phase follows from the fact that the number of server failures in α is at most f. Thus, there is at least one quorum of servers that eventually responds to the *put* message from the read π_r. We have thus shown (1), (2) and (3), which imply that the read operation π_r terminates. □

Remark 6.5. *The condition of concurrency being smaller than ν in Theorem 6.3 is a sufficient condition for the termination of a read operation, but it is not necessary. Indeed, as highlighted in Remark 6.2 and Remark 6.3, the reader may safely return a value v with tag t when the number of tags strictly higher than t, denoted by u, satisfies $u \geq \nu$. The third read in Example 6.3 is an illustration of the aforementioned cases.*

In practice, if a read operation aborts, the reader in Algorithm 6.1 can repeatedly invoke new read operations until it can decode a value that it can safely return. Due to asynchrony, a read may sample symbols from different writes at different times. Consequently, a read may not be able to see k matching pieces of any single new value for an indefinite long period, as long as new values continue to be written concurrently with the read. Therefore, we require reads to return in executions where a finite number of writes are invoked, thus only guaranteeing finite-write (FW) termination.

In Figure 6.2, we describe the read protocol for FW termination. We define a read *iteration* to be an execution of Lines 2 through 16 in Figure 6.2. The variable L represents the list of received $(tag, values)$ pairs, T is the set of valid tags to return, *once* indicates whether

the current iteration is the first iteration or not, and Γ is the highest tag in the first iteration. Unlike the read protocol in Algorithm 6.1, a read invoking multiple iterations can combine responses from different iterations to form a quorum of responses that would allow it to reconstruct a value it can return. This is reflected in Line 12. Each decodable tag t corresponds to a quorum of servers and each server contributes only one $(tag, value)$ pair, which can come from any iteration. Informally, decoding t with such a quorum is equivalent to a single-iteration read operation as in Algorithm 6.1 if Line 13 or Line 14 is satisfied. Moreover, a read is allowed to return any value with a tag that is at least as large as the highest tag it observed in the first iteration (Lines 9 and 15).

Algorithm 6.2 Read protocol for FW termination

1: **function** READ( ) at read client c_i
Initialize $L \leftarrow \emptyset$, $T \leftarrow \emptyset$, $once \leftarrow false, \Gamma \leftarrow 0$.
2: **repeat**
3: || **for** $s \in \{1, 2, \ldots, N\}$
4: Send query request $get()$ to server s,
5: **wait until** receive acknowledgments from a quorum.
6: Let R be the set of response pairs.
7: $L \leftarrow L \cup R$.
8: **if** $once = false$ **then**
9: Let Γ be the maximum tag in R;
10: $once \leftarrow true$.
11: **end if**
12: Let T be the set of decodable tags t in L such that:
1) there exists a quorum of responses in L from which t is decodable,
2) one of the following condition is satisfied
13: (i) t appears in the responses of at least $f + 1$ distinct servers, or
14: (ii) the number of tags in the quorum that are strictly higher than t is at most ν, or
15: (iii) $t \geq \Gamma$.
16: **until** $T \neq \emptyset$.
17: Let $t = \max(T)$ and v its value.
write-back phase
18: Let $(y_1, y_2, \ldots, y_N) = \Phi(v)$.
19: || **for** $s \in \{1, 2, \ldots, N\}$
20: Send $put(t, y_s)$
21: **wait until** receive acknowledgments from a quorum.
22: **return** v.
23: **end function**

Theorem 6.4 (Finite-write termination)**.** *Algorithm* 1 *with re-invoked reads as in Figure 6.2 guarantees atomicity and FW termination.*

Proof. We first prove atomicity using a similar proof to Theorem 6.1. Note that Lemmas 6.3, 6.5 and 6.6 still hold. So we only need to prove the statement of Lemma 6.4. Let t_1 denote the tag of a preceding write or read and Q_1 the quorum that replied to its *put* message.

First, assume Line 15 in Figure 6.2 is satisfied. Let Q_{first} denote the quorum of servers that replied to the read in its first iteration. As $Q_{first} \cap Q_1 \neq \emptyset$, it follows that $\Gamma \geq t_1$. That is, the tag Γ, selected in Line 9, is higher than or equal to the tag of all preceding writes and reads. Thus, a value returned by a reader with tag satisfying Line 15 satisfies the statement of Lemma 6.4. Otherwise, Line 13 or Line 14 is satisfied. Since at most one response from each server is used in decoding, Lemma 6.4 can be proved in the same manner as the case with single-iteration read.

Next, we show FW termination. Consider a fair execution with a finite number of writes. Since non-failed writes terminate, let P be a point that all writes either completed or failed. There can be at most $\nu - 1$ failed writes (in fact, since there is a single writer, there is at most 1 failed write). Suppose there is a read that does not complete by point P. Because the execution is fair and at most f servers fail, the read can take steps and start a new iteration. Hence the read is re-invoked after point P. There is at most $\nu - 1$ concurrent writes (corresponding to failed writes) with the read, and because $\nu \geq 2$, the read terminates by Theorem 6.3. □

6.3.4 Storage and communication costs of Algorithm 6.1

By the design of Algorithm 6.1, the result below follows.

Theorem 6.5. *The worst-case storage cost of Algorithm 6.1 is* $\frac{N}{k} = \frac{N}{\lceil \frac{N-2f}{\nu} \rceil}$ *units. The write communication cost is* $\frac{N}{k}$*. The read communication cost is* $\frac{2N}{k}$*.*

Remark 6.6. *A reader does not need to write-back a value v with tag t if the reader has observed a tag higher than t among the responses. By the well-formedness assumption, the write client does not start a new write until it completes the previous one.*

6.4 First algorithm for multiple writers

6.4.1 Algorithm description

Assume in this section that $\nu < N - 2f, \nu \geq 1$ and thus $k > 1$. We extend Algorithm 6.1 to the MRMW setting, in which we assume the presence of an arbitrary number of writers and an arbitrary number of readers. The proposed algorithm, referred to as Algorithm 6.3, combines replication and multi-version codes while achieving consistency and low storage costs.

Algorithm 6.3 contains a description of the protocol. Each server maintains a $(tag, element)$ tuple, where *element* can be a full replica or a coded symbol. We assume that tags are tuples of the form (z, id), where z is an integer and id is an identifier of a write client. The ordering on the set of tags $\mathcal{T}$ is defined lexicographically, using the usual ordering on the integers and a predefined ordering on the client identifiers. Note that the servers only maintain the highest tag and the corresponding element.

In the write protocol, the *query* phase first obtains the tags of the servers in order to generate a higher tag. In the *pre-write* phase, a writer propagates a full replica to at least $k+f$ servers to ensure that the consistency of the data is not compromised in the presence of concurrent writers. In the *finalize* phase, coded symbols are sent to a quorum of servers. As $k = \lceil \frac{N-2f}{\nu} \rceil$, it follows that $N - f \geq f + k + (\nu - 1)k \geq f + k$. Hence, the *pre-write* quorum is smaller than the *finalize* quorum.

The read protocol of Algorithm 6.3 is essentially similar to Algorithm 6.1, except that a reader can receive coded symbols and/or replicas. In particular, a decodable tag may come from a replica and/or k matching coded symbols. The *write-back* in the read is the same as the write protocol, but with no *query* phase.

Algorithm 6.3 First MWMR algorithm

Write client protocol

1: **function** WRITE(v)
query phase
2: || **for** $s \in \{1, 2, \ldots, N\}$
3: Send *get_tag*() to server s,
4: **wait until** receive responses from a quorum.
5: Select the largest tag from the query phase; let its integer component be z.
6: Form a new tag t as $(z+1, id)$, where id is the identifier of the client performing the operation.
pre-write phase
7: || **for** $s \in \{1, 2, \ldots, k+2f\}$
8: Send $put(t, v)$ to server s,
9: **wait until** receive responses from $k+f$ servers.
finalize phase
10: Let $(y_1, y_2, \ldots, y_N) = \Phi(v)$.
11: || **for** $s \in \{1, 2, \ldots, N\}$
12: Send $put(t, y_s)$ to server s,
13: **wait until** receive responses from a quorum.
14: **end function**

Read client protocol

15: **function** READ( )
16: || **for** $s \in \{1, 2, \ldots, N\}$
17: Send *get*() message to server s,
18: **wait until** receive responses from a quorum.
19: Let R be the set of response pairs.
20: Let T be the set of decodable tags t occurring in R such that
21: (i) t appears in at least $f+1$ responses, or
22: (ii) the number of tags strictly higher than t is at most ν.
23: **if** $T \neq \emptyset$ **then**
24: Let $t = \max(T)$ and v its value.
write-back phase
25: || **for** $s \in \{1, 2, \ldots, k+2f\}$
26: Send $put(t, v)$ to server s,
27: **wait until** receive responses from $k+f$ servers.
28: Let $(y_1, y_2, \ldots, y_N) = \Phi(v)$.
29: || **for** $s \in \{1, 2, \ldots, N\}$
30: Send $put(t, y_s)$ to server s,
31: **wait until** receive responses from a quorum.
32: **return** v.
33: **else**
34: *_abort_*.
35: **end if**
36: **end function**

Server s protocol

37: *state variable* at server s: A pair (t, x), where $t \in \mathcal{T}$, $x \in \mathcal{V} \cup \mathcal{W}$.
38: Initially, server s stores $(t, y) = (t_0, y_s)$, where y_s is the sth component of $\Phi(v_0)$.
39: **Upon receipt of *get_tag*() do**
40: respond with the stored tag t.
41: **Upon receipt of *get*() do**
42: respond with $(t, *)$, where $*$ can be a coded symbol or a full value.
43: **Upon receipt of** $put(t_{new}, v_{new})$ such that $v_{new} \in \mathcal{V}$, **do**
44: If $t_{new} > t$, then set $t \leftarrow t_{new}$ and $x \leftarrow v_{new}$. In any case, respond with acknowledgement.
45: **Upon receipt of** $put(t_{new}, y_{new})$ such that $y_{new} \in \mathcal{W}$, **do**
46: If $t_{new} \geq t$, then set $t \leftarrow t_{new}$ and $x \leftarrow y_{new}$. In any case, respond with acknowledgement.

***Pre-write* phase role:** Algorithm 6.3 extends Algorithm 6.1 and preserves its *in-place* update property. It can be argued that if servers overwrite coded symbols with newer coded symbols, then the storage system may lose the data indefinitely, even if the number of server failures is less than f. Indeed, one can imagine a scenario with N writers. Each writer updates one specific server with coded symbols corresponding to a different new version, then, the N writers fail. Each server stores coded symbols of a different value, and no single version can be recovered. Furthermore, it can be checked carefully through specific examples, or through Lemma 6.8, that $k+f$ is the minimum number of servers that need to store full replicas in the *pre-write* phase in order to avoid losing the stored data, irrespective of the number of concurrent writes. Furthermore, under a specific constraint on the write concurrency, the need for a *pre-write* phase is indeed obviated and the *in-place* property is maintained. See Section 6.5 for details.

Remark 6.7. *During the* finalize *phase of the write protocol, the writer does not need to send coded symbols to all the servers. Indeed, the writer sends full replicas to the first $k+2f$ servers in the* pre-write *phase. Then, during the* finalize *phase, the writer sends coded symbols to the remaining $N-k-2f$ servers and only a finalize message with the corresponding tag to the first $k+2f$ servers so as to minimize the write communication cost.*

6.4.2 Safety properties

Definition 6.3 (Tag of an operation π)**.** *The tag of operation π in an execution α is defined to be the tag associated with the* put *messages that the operation propagates to the servers. The* put *message used in the definition can be from either a* pre-write *or a* finalize *phase since they have the same tag.*

Lemma 6.8 (persistence of data)**.** *The value with the highest tag can be fully recovered at any point of an execution, as long as the number of server failures is bounded by f.*

Proof. We analyze the storage content of the system, at an arbitrary point P of an execution. Let t be the maximum tag in the system, at point P. If t is stored with a full replica at some server, then we can immediately recover the value with tag t. Otherwise, let u be the number of the different coded symbols with tag t that are stored in the system. Since there is no full replica with tag t, it follows from the write protocol that the writer of tag t has finished its *pre-write* phase and started its *finalize* phase. Thus, the value with tag t has been stored in at least $k + f$ servers. Moreover, these replicas must have been replaced by their corresponding coded symbols, or are stored in failed servers. Finally, noting that at most f servers can fail, it follows that $u \geq (k + f) - f = k$. This means that there exists a sufficient number of coded symbols that allow recovery of the value with tag t. □

The proof of atomicity follows along the lines of the SWMR algorithm, with appropriate modifications. In particular, the statements of Lemmas 6.3, 6.4 and 6.5 hold by considering the write quorum to be the *finalize* quorum. The statement of Lemma 6.6 follows from the *query* phase. We start by stating the equivalent of Lemma 6.3 in the context of Algorithm 2.

Lemma 6.9. *Consider any execution α of Algorithm 2 and consider a write or read operation π_1 that completes in α. Let $T(\pi_1)$ denote the tag of the operation π_1 and let Q_1 denote the quorum of servers from which responses are received by π_1 to its* finalize *message. Consider a read operation π_r in α that is invoked after the termination of the write operation π_1. Suppose that the read π_r receives responses to its* get *message from a quorum Q_r. Then,*

1. *Every server s in $Q_1 \cap Q_r$ responds to the* get *message from π_r with a tag that is at least as large as $T(\pi_1)$.*
2. *If, among the responses to the* get *message of π_r from the servers in $Q_1 \cap Q_r$, the number of tags is at most ν, then there is some tag t such that*
   (i) $t \geq T(\pi_1)$, and

(ii) from the servers in $Q_1 \cap Q_r$, π_r receives a full replica or k coded symbols with tag t.

Proof. Proof of (1). Similar to the proof of (1) in Lemma 6.3.

Proof of (2). Among the responses from $Q_1 \cap Q_r$, the read π_r receives at most ν different tags. If among these responses, the reader receives a full replica, then the Lemma follows. Otherwise, all the responses from $Q_1 \cap Q_r$ are associated with coded symbols. The rest of the proof is similar to Lemma 6.3. □

Lemma 6.10. *Consider an execution α of Algorithm 2. Let π_1 be a write or read operation that completes in α, and let π_2 be a read operation that completes in α. Let $T(\pi_1)$ denote the tag of operation π_1 and $T(\pi_2)$ denote the tag of operation π_2. If π_2 begins after the termination of π_1, then $T(\pi_2) \geq T(\pi_1)$.*

Proof. Similar to Lemma 6.4. □

Remark 6.2 holds in the context of Algorithm 6.3. In particular, a reader can safely return a value for which it has received at $f + 1$ responses (which can be coded or non-coded).

Lemma 6.11. *Consider an execution α of Algorithm 2. Let π_1 be a write or read operation that completes in α, and π_2 be a write operation that completes in α. Let $T(\pi_1)$ denote the tag of operation π_1 and $T(\pi_2)$ denote the tag of operation π_2. If π_2 begins after the termination of π_1, then $T(\pi_2) > T(\pi_1)$.*

Proof. The operation π_1 terminates after completing its *finalize* phase, during which it receives responses from a quorum, $Q_f(\pi_1)$. From the server protocol, we can observe that every server s in $Q_f(\pi_1)$ stores a tag that is at least as large as $T(\pi_1)$ at the point of responding to the second *put* message of π_1. We denote the quorum of servers that respond to the

query phase of π_2 as $Q_q(\pi_2)$. It follows that π_2 receives tags that are no smaller than $T(\pi_1)$ from every server $s \in Q_f(\pi_1) \cap Q_q(\pi_2)$. Because the integer part is incremented, the largest integer part of the tag z that the writer in π_2 observes is no less than $T(\pi_1)$. It follows that $T(\pi_2) > T(\pi_1)$. □

Lemma 6.12. *Let π_1, π_2 be write operations that terminate in an execution α of Algorithm 2. Then $T(\pi_1) \neq T(\pi_2)$.*

Proof. Let π_1, π_2 be write operations that terminate in an execution α. Let C_1, C_2 respectively indicate the identifiers of the client nodes at which operations π_1, π_2 are invoked. We consider two cases.

Case 1: $C_1 \neq C_2$: From the write protocol, we note that $T(\pi_i) = (z_i, C_i)$. Since $C_1 \neq C_2$ we have $T(\pi_1) \neq T(\pi_2)$.

Case 2: $C_1 = C_2$: Recall that operations at the the same clients are *well-formed*: where a new invocation awaits the response of a preceding invocation. It means that one of the operations should complete before the other starts. Suppose that, without loss of generality, the write operation π_1 completes before the write operation π_2 starts. Then, Lemma 6.11 implies that $T(\pi_2) > T(\pi_1)$. This implies that $T(\pi_2) \neq T(\pi_1)$. □

Theorem 6.6. *Algorithm 6.3 emulates an atomic read-write object.*

Proof. Same steps as in Theorem 6.1, with Lemmas 6.4, 6.5 and 6.6 replaced by Lemmas 6.10, 6.11 and 6.12, respectively. □

The statement of the safety property in Lemma 6.7 holds also in the setting of Algorithm 6.3, and can also be used to prove the liveness properties of Algorithm 6.3.

Lemma 6.13. *Consider any execution α of Algorithm 6.3. Let π_r denote a read operation in α that receives a quorum Q_r of responses to its* get *message. Let $\mathcal{S}$ denote the set of all*

writes that terminate before the invocation of π_r in α. If $\mathcal{S}$ is non-empty, let t_w denote the largest among the tags of the operations in $\mathcal{S}$. If $\mathcal{S}$ is empty, let $t_w = 0$.

If the number of writes concurrent with the read π_r in α is smaller than ν, then there is some tag t such that
(1) $t \geq t_w$,
(2) π_r can recover the value with tag t, and
(3) the number of tags that are higher than t is smaller than ν.

6.4.3 Liveness properties

The proofs of the liveness properties of Algorithm 2 are similar to those of Algorithm 1, and are omitted.

Theorem 6.7 (Termination of writes)**.** *Consider any fair execution α of Algorithm 6.3 where the number of server failures is at most f, and the write client does not fail. Then, every write operation terminates in α.*

Theorem 6.8 (Termination of reads)**.** *Consider any fair execution α of Algorithm 6.3 where the number of server failures is at most f. Consider any read operation that is invoked at a non-failing client in α. If the number of writes concurrent with the read is strictly smaller than ν, and the read client does not fail, then the read operation completes in α.*

Similar to Algorithm 6.1, if a read operation aborts, the reader in Algorithm 6.3 can repeatedly invoke new read operations until it can decode a value that it can safely return. When the read operation takes many iterations, we use the same read protocol as in Figure 6.2. Next we consider FW termination of a read. The challenge lies in the fact that even if there are finitely many writes, there can be infinitely many read operations that write back, preventing the read from satisfying Lines 13, 14 or 15 in Figure 6.2 in any iteration. In our proof, we make use of Lemma 6.8.

Lemma 6.14 (Finite-write termination)**.** *Algorithm 6.3 with re-invoked reads as in Figure 6.2 guarantees atomicity and FW termination.*

Proof. The proof of atomicity is similar to Theorem 6.4. We present the proof for the termination of a read operation.

Consider a fair execution with a finite number of writes. Consider a read operation π_r. For $i \geq 1$, let s_i denote the point such that π_r initialed iteration i and t_i denote the point that π_r received responses from a quorum of servers, denoted Q_i.

Assume that π_r did not terminate at the end of iteration i for some $i \geq 1$. If $Q_i = Q_{i+1}$ and the responses from each server $s \in Q_i = Q_{i+1}$ are the same in both iterations, then, it follows that each server $s \in Q_i = Q_{i+1}$ has not changed its state, and hence its response, between points t_i and s_{i+1}. Thus, π_r has observed an instantaneous image of the system at point t_i. By Lemma 6.8, using its responses from Q_i, π_r could have recovered a value with tag satisfying Line 14 in Figure 6.2. Henceforth, π_r could have terminated, a contradiction. It follows that either $Q_i \neq Q_{i+1}$ or $Q_i = Q_{i+1}$ and there exists a server $s \in Q_i = Q_{i+1}$ that has responded with different $(tag, element)$ pairs in Q_i and Q_{i+1}.

Note that each server can only increase the stored tag, or change from a replica to a coded symbol for the same tag. Because we assume an execution with a finite number of writes, each server may change its stored $(tag, element)$ pair, and thus its responses, only a finite number of times. Moreover, as there are finitely many quorum sets, there must exist an iteration $j \geq 1$ such that if π_r failed in iteration j, then $Q_j = Q_{j+1}$, and each server in $s \in Q_j = Q_{j+1}$ replies with the same $(tag, element)$ pair. By the previous argument, π_r will terminate at the end of iteration $j + 1$. Moreover, j is finite and satisfies $j \leq 2(\binom{N}{N-f} + \ldots + \binom{N}{N})N_w = (2^{N+1} - 2^{N-f})N_w$, where N_w is the number of writes during the execution. □

6.4.4 Storage and communication costs of Algorithm 6.3

Theorem 6.9. *The worst-case storage cost of Algorithm 6.3 is* $k+2f+\frac{N-k-2f}{k}$. *The steady-state storage cost is given by* $\frac{N}{k}$. *The write communication cost is* $k+2f+\frac{N-k-2f}{k}$. *The read communication cost is* $2(k+2f+\frac{N-k-f}{k})$. *Here* $k=\lceil\frac{N-2f}{\nu}\rceil$.

Proof. The first $k+2f$ servers stores a full replica in the worst case and stores a coded symbol in the steady state; the remaining servers stores a coded symbol. Hence the storage cost results hold.

A write operation proceeds in three phases. As we do not account for the cost of tags, the cost of a write operation is dominated by the cost of its *pre-write* and *finalize* phases. A writer sends its uncoded value to the first $k+2f$ servers in the *pre-write* phase. Based on Remark 6.7, in the *finalize* phase, the writer needs to send coded symbols to only $N-k-2f$ servers. In total, the write communication cost is at most $k+2f+\frac{N-k-2f}{k}$. The worst-case read cost corresponds to twice the write communication cost and it corresponds to a situation in which a reader needs to write back its value. □

Remark 6.8. *A reader's* write-back *proceeds in two phases:* pre-write *and* finalize. *If a reader has received at least one coded response with tag* t, *then the* pre-write *phase of tag* t *must have already completed. The reader does not need to carry out the* pre-write *step and it may only perform the* finalize *phase. Moreover, the* write-back *procedure can be entirely skipped if the read observes* $N-f$ *coded symbols with tag* t.

6.5 Algorithm 6.3-A: algorithm with asymptotically optimal storage cost

In this section, we assume that $\nu \geq 2$ and $k > 1$. We present a MRMW algorithm that is similar to Algorithm 6.3, except that the write procedure is limited to two phases.

Definition 6.4. *Consider an execution α and a point P. Let t_w denote the largest tag among all write operations that completed before P ($t_w = 0$ if no write has completed).* The number of concurrent writes at point P *is the number of write operations that started before P such that their tag is greater than t_w.*

The above definition allows us to ignore write operations due to failed writes while counting the write concurrency at a point P, as long as the failed writes are followed by a successful write before point P. It is worth noting the difference with Remark 6.4, which counts the number of concurrent writes *with the read operation*, while Definition 6.4 counts the number of concurrent writes *at a given point*. Throughout this section, we assume that the following condition is satisfied:

Condition 1: *at any point during the execution, the number of concurrent writes is smaller than ν.*

Algorithm 6.3-A description: By virtue of Condition 1, the write protocol in Algorithm 6.3 can be simplified as the need for a *pre-write* phase is obviated. After acquiring its tag, a writer propagates coded symbols to a quorum of servers before terminating. The read and server protocols are exactly the same as in Algorithm 6.1.

We note that if the number of writers is less than ν (e.g., in applications with pre-determined write clients), then Condition 1 is ensured by default. In particular, the SWMR setting is a

special case of Condition 1. That is the reason that Algorithm 6.3-A is almost the same as Algorithm 6.1.

The proof techniques used for Algorithm 6.1 hold for Algorithm 2-A. For instance, the *persistence of data* property follows from Lemma 6.7. The only difference lies in showing that any two write operations have distinct tags, which follows from the *query* phase of Algorithm 2-A. Finally, atomicity and liveness, i.e., Theorems 6.6, 6.7, and 6.8 can be proved for Algorithm 2-A.

Theorem 6.10. *Let $2 \leq \nu \leq f+2$, the storage cost of Algorithm 6.3-A is $\frac{N}{\lceil \frac{N-2f}{\nu} \rceil}$ at any point of the execution. Moreover, the storage cost of Algorithm 6.3-A is asymptotically matches the lower bound of [122] for $\nu \gg 1, N \gg 1, f = o(N)$.*

Proof. The storage cost at any point is $\frac{N}{\lceil \frac{N-2f}{\nu} \rceil} = \frac{N}{1+\frac{N-2f-1}{\nu}} = \frac{\nu N}{N-2f+\nu-1}$ by Equation (6.1). On the other hand, a careful analysis of [122, Theorem 6.5] shows that a worst-case storage lower bound of $\frac{(\nu-1)N}{N-f+\nu-2}$ holds under the following weaker liveness requirement*:

Liveness requirement for [122, Theorem 6.5]: Consider a fair execution where the number of server failures is no more than f, the number of writers is less than ν, $\nu \leq f+2$, and there is one reader. Each client invokes at most 1 operation. A write operation should terminate if the writer does not fail. If, at a point in the execution, all the writers fail, and a read operation is invoked, then the read operation should terminate.

In particular, this liveness condition is satisfied by Algorithm 2-A under Condition 1. Moreover, the ratio of the storage cost of Algorithm 2-A to the lower bound, given by $\frac{\nu}{\nu-1}\frac{N-f-2+\nu}{N-2f-1+\nu}$, approaches 1 as N, ν increase, such that $f = o(N)$. □

The bound in [122, Theorem 6.5] applies also to CASGC [27] and SCCK [119]. In particular,

*In [122, Theorem 6.5], the algorithm is restricted to one that write protocols consist of phases and at most one phase in a write operation sends messages dependent on the object value, which is satisfied by our Algorithm 6.3-A.

we consider CASGC with parameters $(k_{CASGC}, \delta) = (N - 2f, 0)$, where each server stores latest observed $\delta + 1$ finished writes. Both CASGC and SCCK with parameter $k_{SCCK} = N - 2f$ have worst-case storage of $\frac{N\nu}{N-2f}$, which also matches the lower bound of [122] for $\nu \gg 1, N \gg 1, f = o(N)$. Note that while Algorithm 2-A satisfies ν-concurrency wait-freedom, CASGC and SCCK do not guarantee liveness of a read operation if the read is concurrent with at least one write operation.

Algorithm 2-A has a worst-case storage of $N/\lceil\frac{N-2f}{\nu}\rceil$, which is smaller than SCCK and CASGC by a factor of up to 2. See Example 6.4 in Section 6.7 for illustration.

For executions such that there are less than ν concurrent writes at any point, a worst-case storage lower bound of $\Omega(min(f, \nu))$ is given in [119] under lock-freedom, which is a weaker liveness than ν-concurrency wait-freedom and FW-termination. Hence this lower bound also applies to SCCK and Algorithm 2-A. SCCK achieves a total storage of $\min(3\nu, 6f)$ units for $N = 3f$, matching asymptotically the bound $\Omega(min(f, \nu))$ [119]. When $N = 3f$, Algorithm 2-A has a worst-case storage which matches $\Omega(min(f, \nu))$ with a smaller multiplicative constant. Indeed, when $\nu \geq f$, Algorithm 2-A reduces to ABD (i.e., $k = 1$) with a worst-case storage of $2f + 1$. When $\nu < f$, then $k > 1$ and the storage is $N/\lceil\frac{N-2f}{\nu}\rceil$, which is smaller than SCCK.

6.6 Second algorithm for multiple writers

6.6.1 Algorithm description

In Algorithm 6.3, replication is used to allow for the concurrency of multiple writers. New writes are done *in-place* at the server as higher tags overwrite older tags. One drawback is that a write operation needs to send its unencoded value to the servers during the write. Thus, it uses larger messages compared to implementations which do not send any unencoded

values to the servers [27, 115].

Contrary to Algorithm 6.3, Algorithm 6.4 preserves momentarily some history related to finished writes in order to accommodate the presence of concurrent writers. A write operation proceeds in 3 phases. 1) *query*: a write operation acquires a suitable tag to use with its value. 2) *propagate*: coded symbols are first propagated to a quorum, and 3) *finalize*: older versions are deleted to reduce the cost of storage.

Additionally, each server stores an extra local variable t_{fin}, that indicates the highest written tag in the system, that is known to the server. Upon receipt of a *put* message with tag t, the server adds the corresponding coded symbol to its list only if $t_{fin} \leq t$. Upon receiving a *finalize* message from a write operation, a server updates t_{fin} and discards coded values with tags smaller than t_{fin}.

Similar to Algorithm 6.1 and Algorithm 6.3, Algorithm 6.4 handles garbage collection and liveness guarantees using the idea of multi-version codes. However, multi-version codes require the use of higher storage size per coded symbol, compared to other erasure-coded register implementations. To account for this discrepancy, Algorithm 6.4 uses two coding parameters during each write operation. Namely, it uses $k' = N - 2f$ during the second phase of the write, and uses $k = \lceil \frac{N-2f}{\nu} \rceil$ during the last phase. Hence, the *finalize* phase consists in increasing the size of the coded symbol of the current write, in addition to deleting older writes.

While the writer can send the full encoded symbols to the servers in the last phase, we propose the use of the following technique to reduce the write communication cost. We define the following variables: $x = \text{lcm}(k, k'), z = \frac{x}{k}, z' = \frac{x}{k'}$. In the second phase of the write, the value is encoded using an (Nz, x) MDS code. Then, the writer stores z' fragments at at least $N - f$ servers. In the third phase, the writer communicates $z - z'$ fragments to the servers who responded to the second phase of the write, and communicates z fragments

to the remaining f servers. The write operation terminates by receiving acknowledgments from at least $N - f$ servers. Note that the last phase ensures that each acknowledging server stores a fraction $\frac{1}{k}$ of the unencoded value. We note that a similar technique was used in ORCAS-B [115], where the objective was to *trim* the stored encoded value at the servers, while our objective is to *enlarge* the stored encoded value.

Definition 6.5 (Tag of an operation π)**.** *We define tags of operations and the partial ordering in the same way as in Definition 6.2 and Definition 6.1.*

6.6.2 Safety and liveness properties

We first state a lemma that proves that some version of data can be always recovered, as long as the number of server failures is bounded by f, irrespective of the number of (concurrent) writers.

Lemma 6.15 (persistence of data)**.** *The value of the completed write with the highest tag or a newer value can be fully recovered from the servers lists, as long as the number of server failures is bounded by f.*

Proof. We analyze the storage content of the system, at an arbitrary point P of an execution. Let π_w denote the write that finished the *propagate* phase with the highest tag in the execution at point P, and let t_w be its tag. Note that t_w is at least as large as the tag of the latest finished write. Let Q_p be the quorum that replied to the propagate phase of π_w. By design, $t_{fin} \leq t_w$ at every server $s \in Q_p$, and each server $s \in Q_p$ stores a fraction of size at least $\frac{1}{k'}$ with tag t_w. As no more than f servers fail in an execution, the size of the available coded symbols with tag t_w across the servers is of size at least $\frac{|Q_p|-f}{k'} \geq \frac{N-2f}{k'} = 1$. Hence, the value with tag t_w can be recovered. □

The proof of atomicity and liveness follows along the lines of the proofs in Algorithm 6.3,

Algorithm 6.4 Second MWMR algorithm

Notation: $k = \lceil \frac{N-2f}{\nu} \rceil, k' = N - 2f$,
$x = \text{lcm}(k, k'), z = \frac{x}{k}, z' = \frac{x}{k'}$.

Write client protocol

1: **function** WRITE(v)
query phase
2: || **for** $s \in \{1, 2, \ldots, N\}$
3: Send *get_tag*() to server s
4: **wait until** receive responses from a quorum.
5: Select the largest tag from the query phase; let its integer component be t_o.
6: Form a new tag t as (t_o+1, id), where id is the identifier of the client performing the operation.
pre-write phase
7: Encode v using an (Nz, x) MDS code to get Nz codeword symbols.
8: Divide the Nz codeword symbols into N groups $y_1, y_2, \ldots, y_N$, each of size z.
9: For each y_i, denote by $y_{i,z'}$ any subset of y_i of size z'.
10: || **for** $s \in \{1, 2, \ldots, N\}$
11: Send $put(t, y_{s,z'})$ to server s,
12: **wait until** receive responses from a quorum, denoted Q_p.
finalize phase
13: For $i \in \{1, \ldots, N\}$, let $\bar{y}_{i,z'} \triangleq y_i \backslash y_{i,z'}$.
14: || **for** $s \in \{1, \ldots, N\}$
15: If $s \in Q_p$, send *finalize*$(t, \bar{y}_{s,z'})$ to server s, otherwise, send *finalize*(t, y_s),
16: **wait until** receive responses from a quorum, denoted Q_f.
17: **end function**

Read client protocol

18: **function** READ( )
19: || **for** $s \in \{1, 2, \ldots, N\}$
20: Send *get*() to server s,
21: **wait until** receive responses from a quorum.
22: Let R be the set of response lists.
23: Let T be the set of decodable tags t occurring in R such that
24: (i) t appears in at least $f + 1$ lists,
25: (ii) Or, the number of tags strictly higher than t is at most ν.
26: **if** $T \neq \emptyset$ **then**
27: Let $t = \max(T)$, and v its value.
write-back phase
28: Encode v using an (Nz, x) MDS code to get Nz codeword symbols.
29: Divide the Nz codeword symbols into N groups $y_1, y_2, \ldots, y_N$, each of size z.
30: For each y_i, denote by $y_{i,z}$ any subset of y_i of size z'.
31: || **for** $s \in \{1, 2, \ldots, N\}$
32: Send $put(t, y_{s,z'})$ to server s,
33: **wait until** receive responses from a quorum, denoted Q_p.
34: For $i \in \{1, \ldots, N\}$, let $\bar{y}_{i,z'} \triangleq y_i \backslash y_{i,z'}$.
35: || **for** $s \in \{1, \ldots, N\}$
36: If $s \in Q_p$, send *finalize*$(t, \bar{y}_{s,z'})$ to s, otherwise, send *finalize*(t, y_s).
37: **wait until** receive responses from a quorum, denoted Q_f.
38: **return** v.
39: **else**
40: *_abort_*.
41: **end if**
42: **end function**

Server s protocol

43: *state variables* at server s: A list of pairs (t, x), where $t \in \mathcal{T}$, x correspond to coded symbols with tag t; a tag $t_{fin} \in \mathcal{T}$.
44: Initially, server s stores the default pair $(0, y_s)$, where y_s is of size $\frac{1}{k}$ and corresponds to the sth component of $\Phi(v_0)$; t_{fin} is initialized to 0.
45: **Upon receipt of *get_tag*() do**
46: respond with the list of stored tags.

47: **Upon receipt of *get*() do**
48: respond with the list of stored tuples (t, x).

49: **Upon receipt of** $put(t, y_{s,z'})$ **do**
50: If $t > t_{fin}$, add $(t, y_{s,z'})$ to the stored list. In any case respond with acknowledgement.

51: **Upon receipt of *finalize*(t, y_s) do**
52: If $t > t_{fin}$: delete $(t, y_{s,z'})$ in case it is stored locally; add (t, y_s) to the stored list; delete all pairs with tag smaller than t; set $t_{fin} \leftarrow t$. In any case respond with acknowledgement.

53: **Upon receipt of *finalize*$(t, \bar{y}_{s,z'})$ do**
54: If $t > t_{fin}$: add $(t, \bar{y}_{s,z'})$ to the stored list; delete all pairs with tag smaller than t; set $t_{fin} \leftarrow t$. In any case respond with acknowledgement.

using Q_f in Algorithm 6.4 for the quorum of write. We state the equivalent theorems in the context of Algorithm 6.4 and omit the proofs.

Theorem 6.11. *Algorithm 6.4 emulates an atomic read-write object.*

Theorem 6.12 (Termination of Writes)**.** *Consider any fair execution α of Algorithm 6.4 where the number of server failures is no bigger than f, and the write client does not fail. Then, every write operation terminates in α.*

Theorem 6.13 (Termination of Reads)**.** *Consider any fair execution α of Algorithm 6.4 where the number of server failures is no bigger than f. Consider any read operation that is invoked at a non-failing client in α. If the number of writes concurrent with the read is strictly smaller than ν, then the read operation completes in α.*

Lemma 6.16 (Finite-write termination)**.** *Algorithm 6.4 with re-invoked reads as in Figure 6.2 guarantees atomicity and FW termination.*

6.6.3 Storage and communication costs of Algorithm 6.4

As servers store coded symbols of writes operations before performing garbage collection, the overall storage can be unbounded. In the following we give a bound on the storage cost of an execution of Algorithm 6.4. Toward that, we define the notion of a superseded write operation at a point in an execution. The definition is provided in [27], and is stated here for completeness.

Definition 6.6 (superseded write operation)**.** *In an execution α of Algorithm 6.4, consider a write operation π that completes its query phase. Let $T(\pi)$ denote the tag of the write. Then, the write operation is said to be superseded at a point P of the execution if there are at least one terminated write operation, with a tag that is bigger than $T(\pi)$, such that every message on behalf of this write operations has been delivered by point P.*

Servers do not store values of superseded writes, as stated below.

Lemma 6.17. *Consider an execution α of Algorithm 6.4 and consider any point P of α. If a write operation π is superseded at point P, then no non-failed server has a coded element corresponding to the value of the write operation π at point P.*

Proof. Consider an execution α of Algorithm 6.4 and a point P in α. Consider a write operation π that is superseded at point P. This means that there exists at least one write operation π_1 such that: i) operation π_1 terminates in α, ii) the tag $T(\pi_1)$ of π_1 is greater than $T(\pi)$, and iii) every message on behalf of operation π_1 is delivered by point P. In particular, every non-failed server s has received a *finalize* message with tag $T(\pi_1)$. Furthermore, the *finalize* message with tag $T(\pi_1)$ arrives at server s by point P. It follows that either Line 52 or Line 54 has been executed. In both cases, coded symbols with tags smaller than $T(\pi_1)$ are deleted, including any stored symbols with tag $T(\pi)$. Moreover, beyond this point, any *put* or *finalize* message with tag $T(\pi)$ are discarded at every non-failed server s. Hence, the statement of the lemma follows. □

Theorem 6.14. *Consider an execution α of Algorithm 6.4 such that, at any point of the execution, the number of writes that have completed their query phase by that point and are not superseded at that point is upper bounded by $w + 1$. The storage cost of the execution is at most $\frac{N}{k} + \frac{wN}{k'}$ units. The write communication cost is at most $\frac{N}{k} + \frac{f}{k'}$ units. The read communication cost is at most $\frac{2N}{k} + \frac{wN}{k'} + \frac{f}{k'}$ units. Here $k = \lceil \frac{N-2f}{\nu} \rceil, k' = N - 2f$.*

Proof. Consider an execution α where at any point of the execution, the number of writes that have completed their query phase by that point and are not superseded at that point is upper bounded by $w + 1$. Consider an arbitrary point P of the execution α, and consider a server s that is non-failed at point P.

We infer from the write and server protocols that, at point P, server s does not store a coded element corresponding to any write operation that has not completed its query phase

by point P. We also infer from Lemma 6.17 that server s does not store a coded element corresponding to an operation that is superseded at point P. Therefore, if server s stores a coded element corresponding to a write operation at point P, we infer that the write operation has completed its query phase but is not superseded by point P. By assumption on the execution α, the number of coded elements at point P of α at server s is upper bounded by $w+1$. Moreover, from the write and server protocols, we infer that at point P, server s can only store a single tag whose coded symbols are of size $\frac{1}{k}$, in addition to other coded symbols of size $\frac{1}{k'}$, corresponding to other write operations that are not superseded. Thus, the worst-case of server s corresponds to storing coded symbols of size $\frac{1}{k}$, in addition to storing extra w coded symbols, each of size $\frac{1}{k}$, which correspond to w write operations that are not superseded and such that none of these w write operations has delivered a *finalize* message to server s. The overall worst-case storage follows.

The write communication cost is at most $Nz'+(N-f)(z-z')+fz = Nz+fz' = \frac{N}{k}x+\frac{f}{k'}x$. As x coded symbols amount to one unit, the result follows. The read communication cost is obtained by considering the worst-case total storage and accounting for the *write_back* operation. □

Remark 6.9. *The write operation in Algorithm 6.4 can be modified in a way to have data communicated only during one phase. This is done by: 1) acquiring a suitable tag in the same way; 2) communicating $\frac{1}{k}$ fraction of the data to every server in the second phase and waiting for an acknowledgment from a quorum; 3) communicating a* finalize *message with the tag of the current write, such that a server upon receiving it, updates its t_{fin} and deletes older tags. The write cost in this case is $\frac{N}{k}$ units, while the worst-case storage is $\frac{N(1+w)}{k}$ units, in the context of Theorem 6.14. The write cost is then smaller while the storage cost is larger, compared with Theorem 6.14. Thus, the storage and communication costs can be traded-off from this perspective.*

6.7 Discussion and conclusion

Different from most previous erasure-code-based algorithms, our algorithms are parameterized by the liveness parameter ν. We discuss now the merits and the disadvantages of our approach. We mainly focus on MWMR algorithms listed in Table 6.1.

Our MWMR algorithms guarantee atomicity and satisfy ν-concurrency wait-freedom. Algorithm 6.3-A is suited for environments where the write concurrency is bounded at all points of the execution. Algorithm 6.3 and Algorithm 6.4 tolerate any arbitrary write concurrency level. However, Algorithm 6.4 is suitable if we expect that, for most of the time, not many writes are invoked concurrently. Moreover, Algorithm 6.3 and Algorithm 6.3-A possess the *in-place* update property.

For $\nu < 2f + 1, N \gg f$, and sufficiently large number of ongoing writes, in descending order of the worst-case storage cost, we have CASGC, Algorithm 6.4, SCCK, Algorithm 6.3 and ABD. In descending order of the steady-state storage cost, we have ABD, CASGC, Algorithm 6.3/Algorithm 6.4, and SCCK. Detailed comparisons are given below.

We compare Algorithm 6.3 and Algorithm 6.4 with the MRMW version of ABD [27]. While our algorithms have higher worst-case storage than ABD, for $\nu < 2f + 1$ their steady-state storage outperforms ABD by

$$2f + 1 - \frac{N}{k} = \frac{(N - 2f - 1)(2f + 1 - \nu)}{N - 2f + \nu - 1} = \frac{k - 1}{k}(2f + 1 - \nu). \tag{6.2}$$

Compared to ABD, our algorithms make use of a higher number of servers to offer storage reduction, at the expense of liveness if a read operation is concurrent with at least ν write operations.

Next we compare Algorithm 6.3 and Algorithm 6.4 with SCCK [119], which is a MWMR

algorithm that combines replication and erasure codes. Assume its coding parameter is $k_{SCCK} = N - 2f$, such that the steady-state storage is minimized. The worst-case storage of SCCK is $2N$, which is at least twice the storage of ABD and Algorithm 6.3, and smaller than that of Algorithm 6.4. Meanwhile, SCCK has a steady-state storage $\frac{N}{N-2f}$, which is lower than Algorithm 6.3 and Algorithm 6.4. Note that SCCK only provides finite-write termination guarantees and also a weaker safety guarantee, namely regularity [119].

We compare Algorithm 6.3 and Algorithm 6.4 to CASGC [27] with parameters $(k_{CASGC}, \delta) = (N - 2f, \nu - 1)$. All algorithms satisfy ν-concurrency wait-freedom. A main difference between Algorithm 6.4 and CASGC lies in the third phase of the write operation. CASCG has a steady-state storage given by $\frac{\nu N}{N-2f}$. Algorithm 6.4, along with Algorithm 6.3, improves upon CASCG in terms of its steady-state storage, which can be close to twice as efficient (c.f. Example 6.4). From Theorem 6.14, the worst-case storage of Algorithm 6.4 is $\frac{N}{k} + \frac{wN}{N-2f}$, while that of CASCG is $\frac{\nu N}{N-2f} + \frac{w_{CASGC} N}{N-2f}$, where w_{CASGC} is related to the write concurrency at a point and can be unbounded.

Example 6.4. *Consider the case of $\nu = N - 2f - 1$. Then, Algorithms 6.3 and 6.4 use coding parameter $k = 1 + \frac{N-(2f+1)}{\nu} = 2$. The steady-state storage of our algorithms is $\frac{N}{2}$. The steady-state storage of CASCG with parameters $(k_{CASGC}, \delta) = (N - 2f, \nu - 1)$ is $\frac{\nu N}{N-2f} = N(1 - \frac{1}{N-2f}) \to N$ as $N - 2f$ increases. Therefore, our algorithms can be close to twice as efficient as CASGC. Moreover, for the same choice of $\nu = N - 2f - 1$, whenever $\nu < 2f + 1 \iff N < 4f + 2$, our algorithms improve upon ABD for steady-state storage.*

While the optimality of the worst-case storage of Algorithm 2-A is shown asymptotically in Theorem 6.10, it is worth noting that Algorithm 2-A is competitive for non-asymptotic regimes. For instance, assuming $f - 1 \leq \nu \leq f + 1, k > 1$, then the storage cost of Algorithm 2-A is within a factor of 2 from the lower bound in Theorem 6.10. Indeed, the ratio of the storage cost of Algorithm 2-A to the lower bound satisfies $\frac{\nu}{\nu-1} \frac{N-2f-1+\nu+f-1}{N-2f-1+\nu} \leq \frac{\nu}{\nu-1} \frac{\nu+f-1}{\nu} = 1 + \frac{f}{\nu-1} \leq 2$. Gaps from the lower bound can be derived similarly for other regimes of

interest. To the best of our knowledge, Algorithm 6.3-A dominates all known algorithms for executions with less than ν concurrent writes at any point, for worst-case storage. However, it is worth noting that, unlike SCCK, Algorithm 6.3-A does not tolerate higher levels of write concurrency.

In conclusion, we proposed fault-tolerant algorithms for emulating a shared memory over an asynchronous, distributed message-passing network. Our algorithms guarantee liveness of the read as long as the number of writes concurrent with the read is smaller than a design parameter ν. The parameter ν illustrates the tradeoff between liveness of read operations and the storage size per node. The overall steady-state storage and communication costs of our algorithms outperform ABD when $\nu < 2f + 1$. While the steady-state storage of the MRMW algorithms presented in this paper is higher compared to some other coding-based schemes, the coding-based liveness guarantee and the in-place update property of Algorithm 6.3 and Algorithm 6.3-A make them appealing from practical perspective and easy to implement. We note that the worst-case storage of Algorithm 6.3 is incurred during each write operation. Hence, an open problem is how to dynamically adapt to the concurrency level when it changes over time, so that the liveness condition is strengthened and the storage cost is lowered.

Chapter 7

Concluding remarks

The problem of designing regenerating codes has received a lot of interest in the research community over the last years. For functional repair, a good understanding of the storage-bandwidth tradeoff has been established for single and multiple node failures. The situation is different under exact repair where, even for the single erasure case, a full characterization of the exact-repair region is not known in general, except for small cases and the $(k+1, k, k, 1)$ system under linear setting. In this dissertation, our goal was to study the problem of regenerating codes in the context of multiple erasures. We obtained several results, including outer bounds and code constructions. Outer bounds include a full characterization of the functional repair tradeoff, infeasibility results and outer bounds for linear codes. Code constructions include a framework for constructing MSMR codes from MSR codes and a family of codes operating at various interior points. Open problems include the generalization of the non-existence proof of linear exact-repair MBMR regenerating codes to non-linear codes. It is interesting to determine the storage and bandwidth values of an exact minimum bandwidth regenerating code. Moreover, characterization of the storage-bandwidth tradeoff for exact repair for the interior points is still not known. Another important research direction is the study of adaptive regenerating codes under exact-repair. Adaptivity of regenerating codes includes flexibility with respect to the number of helper nodes [76, 94] and/or the number of erasures that can be non-trivially repaired [80]. The progressive repair constructions for

Reed-Solomon codes, the family of interior point constructions, and the adaptive multi-node repair MBR code design presented in this dissertation are steps toward this direction. Recently, work [125] showed that determinant codes, developed for single erasures, can recover multiple erasures, and the corresponding bandwidth is computed. However, developing outer bounds for adaptive regenerating codes is still an open problem and it is of interest in order to assess the performance of existing and future code designs.

In the second part of this dissertation, motivated by the sneak path problem in resistive memories, we studied polar coding over channels with different reliability levels. In particular, we argued that the channels' ordering is important and proposed a channel ordering with empirically attractive performance. We then applied our framework to the sneak path problem in resistive memories using SPICE-like resistive, with significant bit-error rate improvement. Our work represents another step toward coding for the resistive crossbar arrays, a research area with plenty of room for improvement. For instance, by the sneak path nature, errors in neighboring cells are not independent. A future direction for research is to investigate enhanced polar decoding algorithms, taking into consideration such correlations. Another avenue for research is to investigate techniques for biasing the distribution of high-resistance values, along with techniques of coding for asymmetric channels as in [126], which may reduce the sneak path occurrences. Likewise, combinations of constrained coding and polar codes, and other coding techniques, notably spatially-coupled LDPC codes, are worth investigating.

In the last part of this dissertation, we studied the problem of storing evolving information in asynchronous distributed systems, which is of relevance for large-scale distributed systems. Several challenges face the integration of erasure codes for such systems in order to fully capture the advantages of coding over replication. Specifically, we studied the problem of emulating an atomic shared memory in asynchronous message-passing networks, for which we proposed new erasure-code based algorithms. In particular, our algorithms feature a de-

sign parameter ν, that provides a tradeoff between liveness of read operations and the storage size per node, while guaranteeing the strictest consistency requirement. We here mention few interesting research directions. Existing emulation algorithms assume the nodes to be entirely distributed and the encoding of a node in entirely based on its own received versions. However, it is natural to consider a model with node collaborations, as node collaboration may be facilitated by architectural design or geographical proximity. Moreover, the previous models assume the client to be able to perfectly decode his message. Relaxing strict decoding/encoding requirement to a probabilistic decoding/encoding may reduce significantly the storage overhead while ensuring a (parameterized) low probability of decoding failure. Finally, along the lines of [119, 123, 122], and for all the considered models, deriving lower bounds on the smallest required storage size is of high practical and theoretical importance.

Bibliography

[1] C. V. Forecast, "Cisco visual networking index: Forecast and trends, 2017–2022," *White paper, Cisco Public Information*, 2019.

[2] D. C. Daly, L. C. Fujino, and K. C. Smith, "Through the looking glass-the 2017 edition: Trends in solid-state circuits from ISSCC," *IEEE Solid-State Circuits Magazine*, vol. 9, no. 1, pp. 12–22, 2017.

[3] M. Zorgui and Z. Wang, "Code constructions for multi-node exact repair in distributed storage," *Science China Information Sciences*, vol. 61, no. 10, p. 100304, 2018.

[4] M. Elyasi and S. Mohajer, "Determinant coding: A novel framework for exact-repair regenerating codes," *IEEE Transactions on Information Theory*, vol. 62, no. 12, pp. 6683–6697, 2016.

[5] I. M. Duursma, "Shortened regenerating codes," *IEEE Transactions on Information Theory*, 2018.

[6] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Transactions on computers*, vol. 44, no. 2, pp. 192–202, 1995.

[7] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[8] K. Shvachko, H. Kuang, S. Radia, R. Chansler *et al.*, "The Hadoop distributed file system," in *MSST*, vol. 10, 2010, pp. 1–10.

[9] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci *et al.*, "Windows Azure storage: a highly available cloud storage service with strong consistency," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 143–157.

[10] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an MTTF of 1, 000, 000 hours mean to you?" in *FAST*, vol. 7, no. 1, 2007, pp. 1–16.

[11] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: Novel erasure codes for big data," in *Proceedings of the VLDB Endowment*, vol. 6, no. 5. VLDB Endowment, 2013, pp. 325–336.

[12] 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.

[13] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A hitchhiker's guide to fast and efficient data reconstruction in erasure-coded data centers," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 331–342.

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

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

[16] I. Tamo and A. Barg, "A family of optimal locally recoverable codes," *IEEE Transactions on Information Theory*, vol. 60, no. 8, pp. 4661–4676, Aug 2014.

[17] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. M. Voelker, "Total Recall: System Support for Automated Availability Management." in *NSDI*, vol. 4, 2004, pp. 25–25.

[18] A. S. Rawat, O. O. Koyluoglu, and S. Vishwanath, "Centralized repair of multiple node failures with applications to communication efficient secret sharing," *IEEE Transactions on Information Theory*, vol. 64, no. 12.

[19] P. Hu, C. W. Sung, and T. H. Chan, "Broadcast repair for wireless distributed storage systems," in *10th International Conference on Information, Communications and Signal Processing (ICICS)*. IEEE, 2015, pp. 1–5.

[20] S. Yu, "Neuro-inspired computing with emerging nonvolatile memorys," *Proceedings of the IEEE*, vol. 106, no. 2, pp. 260–285, 2018.

[21] M. E. Fouda and et.al, "Modeling and analysis of passive switching crossbar arrays," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 1, pp. 270–282, 2018.

[22] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazons Highly Available Key-value Store," in *ACM SIGOPS operating systems review*, vol. 41, no. 6. ACM, 2007, pp. 205–220.

[23] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[24] "Apache couchdb," https://couchdb.apache.org/, accessed: Nov, 2019.

[25] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[26] H. Attiya, A. Bar-Noy, and D. Dolev, "Sharing memory robustly in message-passing systems," in *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, ser. PODC '90. New York, NY, USA: ACM, 1990, pp. 363–375.

[27] V. R. Cadambe, N. Lynch, M. Médard, and P. Musial, "A coded shared atomic memory algorithm for message passing architectures," *Distributed Computing*, vol. 30, no. 1, pp. 49–73, 2017.

[28] D. Dobre, G. Karame, W. Li, M. Majuntke, N. Suri, and M. Vukolić, "PoWerStore: proofs of writing for efficient and robust storage," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 285–298.

[29] J. Hendricks, G. R. Ganger, and M. K. Reiter, "Low-overhead byzantine fault-tolerant storage," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 73–86, 2007.

[30] Z. Wang and V. R. Cadambe, "Multi-version coding - an information-theoretic perspective of consistent distributed storage," *IEEE Transactions on Information Theory*, vol. PP, no. 99, pp. 1–1, 2017.

[31] M. Zorgui and Z. Wang, "Centralized multi-node repair in distributed storage," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Sept 2016, pp. 617–624.

[32] ——, "Centralized multi-node repair for minimum storage regenerating codes," in *IEEE International Symposium on Information Theory (ISIT)*, June 2017, pp. 2213–2217.

[33] ——, "Centralized multi-node repair regenerating codes," *IEEE Transactions on Information Theory*, 2019.

[34] ——, "On the achievability region of regenerating codes for multiple erasures," in *IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 2067–2071.

[35] M. Zorgui, M. E. Fouda, Z. Wang, A. Eltawil, and F. Kurdahi, "Polar coding for selector-less resistive memories," in *10th Annual Non-Volatile Memories Workshop*, 2019.

[36] ——, "Non-stationary polar codes for resistive memories," *arXiv preprint arXiv:1904.08966*, 2019.

[37] M. Zorgui, R. Mateescu, F. Blagojevic, C. Guyot, and Z. Wang, "Storage-efficient shared memory emulation," *arXiv preprint arXiv:1803.01098*, 2018.

[38] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[39] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, 2006.

[40] N. B. Shah, K. Rashmi, P. V. Kumar, and K. Ramchandran, "Interference alignment in regenerating codes for distributed storage: Necessity and code constructions," *IEEE Transactions on Information Theory*, vol. 58, no. 4, pp. 2134–2158, 2012.

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

[42] Y. Wu and A. G. Dimakis, "Reducing repair traffic for erasure coding-based storage via interference alignment," in *2009 IEEE International Symposium on Information Theory*. IEEE, 2009, pp. 2276–2280.

[43] D. S. Papailiopoulos, A. G. Dimakis, and V. R. Cadambe, "Repair optimal erasure codes through hadamard designs," *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 3021–3037, 2013.

[44] Z. Wang, I. Tamo, and J. Bruck, "Explicit MDS codes for optimal repair bandwidth," *arXiv preprint arXiv:1411.6328*, 2014.

[45] A. S. Rawat, O. O. Koyluoglu, and S. Vishwanath, "Progress on high-rate msr codes: Enabling arbitrary number of helper nodes," in *Information Theory and Applications Workshop (ITA)*, 2016.

[46] S. Goparaju, A. Fazeli, and A. Vardy, "Minimum storage regenerating codes for all parameters," *IEEE Transactions on Information Theory*, vol. 63, no. 10, pp. 6318–6328, 2017.

[47] V. R. Cadambe, S. A. Jafar, H. Maleki, K. Ramchandran, and C. Suh, "Asymptotic interference alignment for optimal repair of MDS codes in distributed storage," *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 2974–2987, 2013.

[48] I. Tamo, Z. Wang, and J. Bruck, "Zigzag codes: MDS array codes with optimal rebuilding," *IEEE Transactions on Information Theory*, vol. 59, no. 3, pp. 1597–1616, March 2013.

[49] M. Ye and A. Barg, "Explicit constructions of optimal-access MDS codes with nearly optimal sub-packetization," *IEEE Transactions on Information Theory*, vol. 63, no. 10, pp. 6307–6317, 2017.

[50] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, "Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff," *IEEE Transactions on Information Theory*, vol. 58, no. 3, pp. 1837–1852, March 2012.

[51] I. M. Duursma, "Outer bounds for exact repair codes," *arXiv preprint arXiv:1406.4852*, 2014.

[52] S. Mohajer and R. Tandon, "New bounds on the (n, k, d) storage systems with exact repair," in *IEEE International Symposium on Information Theory (ISIT)*, 2015, pp. 2056–2060.

[53] B. Sasidharan, K. Senthoor, and P. V. Kumar, "An improved outer bound on the storage-repair-bandwidth tradeoff of exact-repair regenerating codes," in *IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2014, pp. 2430–2434.

[54] A.-M. Kermarrec, N. Le Scouarnec, and G. Straub, "Repairing multiple failures with coordinated and adaptive regenerating codes," in *International Symposium on Network Coding (NetCod)*. IEEE, 2011, pp. 1–6.

[55] K. W. Shum and Y. Hu, "Cooperative regenerating codes," *IEEE Transactions on Information Theory*, vol. 59, no. 11, pp. 7229–7258, Nov 2013.

[56] J. Li and B. Li, "Cooperative repair with minimum-storage regenerating codes for distributed storage," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 316–324.

[57] M. Ye and A. Barg, "Cooperative repair: Constructions of optimal MDS codes for all admissible parameters," *IEEE Transactions on Information Theory*, vol. 65, no. 3, pp. 1639–1656, 2018.

[58] A. Wang and Z. Zhang, "Exact cooperative regenerating codes with minimum-repair-bandwidth for distributed storage," in *2013 Proceedings IEEE INFOCOM*, April 2013, pp. 400–404.

[59] H. Lee and J. Lee, "An outer bound on the storage-bandwidth tradeoff of exact-repair cooperative regenerating codes," *IEEE Transactions on Information Theory*, vol. 63, no. 11, pp. 7267–7282, 2017.

[60] K. Rashmi, N. Shah, and P. Kumar, "Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp. 5227–5239, Aug 2011.

[61] Y. Wu, A. G. Dimakis, and K. Ramchandran, "Deterministic regenerating codes for distributed storage," in *Allerton Conference on Control, Computing, and Communication*. Citeseer, 2007, pp. 1–5.

[62] H. Zhang, H. Li, K. W. Shum, H. Hou, and S. R. Li, "Concurrent regenerating codes," *IET Communications*, vol. 11, no. 3, pp. 362–369, 2017.

[63] M. Elyasi, S. Mohajer, and R. Tandon, "Linear exact repair rate region of (k+ 1, k, k) distributed storage systems: A new approach," in *IEEE International Symposium on Information Theory (ISIT)*, 2015, pp. 2061–2065.

[64] Z. Wang, I. Tamo, and J. Bruck, "Optimal rebuilding of multiple erasures in MDS codes," *IEEE Transactions on Information Theory*, vol. 63, no. 2, pp. 1084–1101, 2017.

[65] N. Prakash and M. N. Krishnan, "The storage-repair-bandwidth trade-off of exact repair linear regenerating codes for the case d= k= n- 1," in *IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2015, pp. 859–863.

[66] V. R. Cadambe, S. A. Jafar, H. Maleki, K. Ramchandran, and C. Suh, "Asymptotic interference alignment for optimal repair of MDS codes in distributed storage," *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 2974–2987, May 2013.

[67] R. Li, J. Lin, and P. P. Lee, "Enabling concurrent failure recovery for regenerating-coding-based storage systems: From theory to practice," *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 1898–1911, 2015.

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

[69] V. Guruswami and M. Wootters, "Repairing reed-Solomon codes," *IEEE Transactions on Information Theory*, vol. 63, no. 9, pp. 5684–5698, 2017.

[70] H. Dau, I. Duursma, H. M. Kiah, and O. Milenkovic, "Repairing Reed-Solomon codes with multiple erasures," *IEEE Transactions on Information Theory*, 2018.

[71] ——, "Repairing Reed-Solomon codes with two erasures," in *IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 351–355.

[72] J. Mardia, B. Bartan, and M. Wootters, "Repairing multiple failures for scalar MDS codes," *IEEE Transactions on Information Theory*, 2018.

[73] I. Tamo, M. Ye, and A. Barg, "The repair problem for Reed–Solomon codes: Optimal repair of single and multiple erasures with almost optimal node size," *IEEE Transactions on Information Theory*, vol. 65, no. 5, pp. 2673–2695, 2018.

[74] W. Li, Z. Wang, and H. Jafarkhani, "On the sub-packetization size and the repair bandwidth of reed-solomon codes," *IEEE Transactions on Information Theory*, 2019.

[75] J. Chen and K. W. Shum, "Repairing multiple failures in the suh-ramchandran regenerating codes," in *2013 IEEE International Symposium on Information Theory*, July 2013, pp. 1441–1445.

[76] V. Aggarwal, C. Tian, V. A. Vaishampayan, and Y.-F. R. Chen, "Distributed data storage systems with opportunistic repair," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 1833–1841.

[77] M. Hajiaghayi and H. Jafarkhani, "MDS codes with progressive engagement property for cloud storage systems," *arXiv preprint arXiv:1605.06927*, 2016.

[78] M. Ye and A. Barg, "Explicit constructions of high-rate MDS array codes with optimal repair bandwidth," *IEEE Transactions on Information Theory*, vol. 63, no. 4, pp. 2001–2014, April 2017.

[79] K. Mahdaviani, S. Mohajer, and A. Khisti, "Product–matrix MSR codes with bandwidth adaptive exact repair," *IEEE Transactions on Information Theory*, vol. 64, no. 4, pp. 3121–3135, 2018.

[80] M. Ye and A. Barg, "Explicit constructions of high-rate MDS array codes with optimal repair bandwidth," *IEEE Transactions on Information Theory*, vol. 63, no. 4, pp. 2001–2014, 2017.

[81] K. Rashmi, N. Shah, and P. Kumar, "Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp. 5227–5239, Aug 2011.

[82] D. E. Knuth, *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1997.

[83] N. Alon and M. Tarsi, "Combinatorial nullstellensatz," *Combinatorics Probability and Computing*, vol. 8, no. 1, pp. 7–30, 1999.

[84] I. H. Morgan and G. L. Mullen, "Primitive normal polynomials over finite fields," *Mathematics of Computation*, vol. 63, no. 208, pp. 759–765, 1994.

[85] N. Carella, "Topics in normal bases of finite fields," *arXiv preprint arXiv:1304.0420*, 2013.

[86] K. Senthoor, B. Sasidharan, and P. V. Kumar, "Improved layered regenerating codes characterizing the exact-repair storage-repair bandwidth tradeoff for certain parameter sets," in *2015 IEEE Information Theory Workshop (ITW)*. IEEE, 2015, pp. 1–5.

[87] C. Tian, B. Sasidharan, V. Aggarwal, V. A. Vaishampayan, and P. V. Kumar, "Layered exact-repair regenerating codes via embedded error correction and block designs," *IEEE Transactions on Information Theory*, vol. 61, no. 4, pp. 1933–1947, 2015.

[88] S. Goparaju, S. El Rouayheb, and R. Calderbank, "New codes and inner bounds for exact repair in distributed storage systems," in *2014 IEEE International Symposium on Information Theory*. IEEE, 2014, pp. 1036–1040.

[89] P. Keevash, "The existence of designs," *arXiv preprint arXiv:1401.3665*, 2014.

[90] C. J. Colbourn, *CRC handbook of combinatorial designs*. CRC press, 2010.

[91] G. M. Kamath, N. Prakash, V. Lalitha, and P. V. Kumar, "Codes with local regeneration and erasure correction," *IEEE Transactions on Information Theory*, vol. 60, no. 8, pp. 4637–4660, 2014.

[92] A. S. Rawat, N. Silberstein, O. O. Koyluoglu, and S. Vishwanath, "Optimal locally repairable codes with local minimum storage regeneration via rank-metric codes," in *Information Theory and Applications Workshop (ITA), 2013*. IEEE, 2013, pp. 1–8.

[93] C. Tian, "A note on the rate region of exact-repair regenerating codes," *arXiv preprint arXiv:1503.00011*, 2015.

[94] K. Mahdaviani, A. Khisti, and S. Mohajer, "Bandwidth adaptive & error resilient regenerating codes with minimum repair bandwidth," in *IEEE International Symposium on Information Theory (ISIT), 2016*. IEEE, 2016, pp. 235–239.

[95] K. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," in *47th Annual Allerton Conference on Communication, Control, and Computing, 2009. Allerton 2009*. IEEE, 2009, pp. 1243–1249.

[96] M. E. Fouda, A. M. Eltawil, and F. Kurdahi, "On resistive memories: One step row readout technique and sensing circuitry," *arXiv preprint arXiv:1903.01512*, 2019.

[97] M. Zidan, H. Omran, R. Naous, A. Sultan, H. Fahmy, W. Lu, and K. N. Salama, "Single-readout high-density memristor crossbar," *Scientific reports*, vol. 6, p. 18863, 2016.

[98] 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.

[99] Z. Chen, C. Schoeny, and L. Dolecek, "Coding assisted adaptive thresholding for sneak-path mitigation in resistive memories," in *IEEE Information Theory Workshop (ITW)*. IEEE, 2018, pp. 1–5.

[100] Y. Ben-Hur and Y. Cassuto, "Detection and coding schemes for sneak-path interference in resistive memory arrays," *IEEE Transactions on Communications*, 2019.

[101] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[102] M. Alsan and E. Telatar, "A simple proof of polarization and polarization for non-stationary memoryless channels," *IEEE Transactions on Information Theory*, vol. 62, no. 9, pp. 4873–4878, 2016.

[103] S. B. Korada, "Polar codes for channel and source coding," 2009.

[104] H. Mahdavifar, "Fast polarization for non-stationary channels," in *IEEE International Symposium on Information Theory*, 2017, pp. 849–853.

[105] E. Arikan, "A performance comparison of polar codes and Reed-Muller codes," *IEEE Communications Letters*, vol. 12, no. 6, pp. 447–449, 2008.

[106] K. Niu, K. Chen, and J.-R. Lin, "Beyond turbo codes: Rate-compatible punctured polar codes," in *IEEE Int. Conf. on Communications (ICC)*. IEEE, 2013, pp. 3423–3427.

[107] E. Arikan, "Systematic polar coding," *IEEE communications letters*, vol. 15, no. 8, pp. 860–862, 2011.

[108] M. P. Herlihy and J. M. Wing, "Linearizability: A correctness condition for concurrent objects," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 12, no. 3, pp. 463–492, 1990.

[109] L. Lamport, "On interprocess communication," *Distributed computing*, vol. 1, no. 2, pp. 86–101, 1986.

[110] R. Fan and N. Lynch, "Efficient replication of large data objects," in *International Symposium on Distributed Computing*. Springer, 2003, pp. 75–91.

[111] N. Lynch and A. A. Shvartsman, "Rambo: A reconfigurable atomic memory service for dynamic networks," in *International Symposium on Distributed Computing*. Springer, 2002, pp. 173–190.

[112] D. A. Patterson, G. Gibson, and R. H. Katz, *A case for redundant arrays of inexpensive disks (RAID)*. ACM, 1988, vol. 17, no. 3.

[113] M. K. Aguilera, R. Janakiraman, and L. Xu, "Using erasure codes efficiently for storage in a distributed system," in *International Conference on Dependable Systems and Networks, 2005. DSN 2005. Proceedings.* IEEE, 2005, pp. 336–345.

[114] C. Cachin and S. Tessaro, "Optimal resilience for erasure-coded byzantine distributed storage," in *International Conference on Dependable Systems and Networks, 2006. DSN 2006. Proceedings.*, pp. 115–124.

[115] P. Dutta, R. Guerraoui, and R. R. Levy, "Optimistic erasure-coded distributed storage," in *International Symposium on Distributed Computing*. Springer, 2008, pp. 182–196.

[116] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence, "Fab: building distributed enterprise disk arrays from commodity components," *ACM SIGOPS Operating Systems Review*, vol. 38, no. 5, pp. 48–58, 2004.

[117] K. M. Konwar, N. Prakash, E. Kantor, N. Lynch, M. Médard, and A. A. Schwarzmann, "Storage-optimized data-atomic algorithms for handling erasures and errors in distributed storage systems," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 2016, pp. 720–729.

[118] K. M. Konwar, N. Prakash, N. Lynch, and M. Médard, "A layered architecture for erasure-coded consistent distributed storage," in *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 2017, pp. 63–72.

[119] A. Spiegelman, Y. Cassuto, G. Chockler, and I. Keidar, "Space bounds for reliable storage: Fundamental limits of coding," in *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*. ACM, 2016, pp. 249–258.

[120] R. Guerraoui, R. R. Levy, B. Pochon, and J. Pugh, "The collective memory of amnesic processes," *ACM Transactions on Algorithms (TALG)*, vol. 4, no. 1, p. 12, 2008.

[121] K. M. Konwar, N. Prakash, N. A. Lynch, and M. Médard, "RADON: Repairable Atomic Data Object in Networks," in *20th International Conference on Principles of Distributed Systems*, 2017.

[122] V. R. Cadambe, Z. Wang, and N. Lynch, "Information-theoretic lower bounds on the storage cost of shared memory emulation," in *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*. ACM, 2016, pp. 305–313.

[123] G. Chockler and A. Spiegelman, "Space complexity of fault tolerant register emulations," in *Proceedings of the 2017 ACM Symposium on Principles of Distributed Computing*. ACM, 2017, pp. 83–92.

[124] I. Abraham, G. Chockler, I. Keidar, and D. Malkhi, "Byzantine disk paxos: optimal resilience with byzantine shared memory," *Distributed Computing*, vol. 18, no. 5, pp. 387–408, 2006.

[125] M. Elyasi and S. Mohajer, "Determinant codes with helper-independent repair for single and multiple failures," *IEEE Transactions on Information Theory*, 2019.

[126] R. Gabrys and L. Dolecek, "Coding for the binary asymmetric channel," in *Int. Conf. Comp., Net. and Comm. (ICNC)*, Jan 2012, pp. 461–465.

,