UNIVERSITY OF CALIFORNIA,
IRVINE

Flexible Coding for Distributed Systems

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Weiqi Li

Dissertation Committee:
Chancellor's Professor Hamid Jafarkhani, Chair
Assistant Professor Zhiying Wang, Chair
Chancellor's Professor Syed Ali Jafar

2021

DEDICATION

To my parents and friends, for their unwavering support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I would like to thank my advisors, Professor Jafarkhani & Professor Wang, for their advice and guidance during my PhD program. They provide me enough freedom to explore my interest and valuable resources for my research. Without their help and support, the thesis would never have been possible. Their high academic and research standards set great examples in my future life.

I would like to thank my colleagues in Professor Jafarkhani's research group: Xiaoyi Liu, Jun Guo, Mehdi Ganji, Xun Zou, Lisi Jiang, Saeed Karimi Bidhendi, Carles Diaz, and Professor Wang's group: Marwen Zorgui, Zhen Chen, Peng Fei, Alireza Javani, Xiaoran Li, Keqing Fu. We had many valuable discussions about our research. It was a precious experience working with them.

I would like to thank my parents for their selfless love. Their supports and encouragements help me a lot during my whole life.

VITA

Weiqi Li

EDUCATION

Doctor of Philosophy in Electrical Engineering and Computer Science **2021**
University of California, Irvine *Irvine, CA*

Master of Information Engineering **2016**
Xi'an Jiaotong University *Xi'an, Shannxi*

Bachelor of Information Engineering **2013**
Xi'an Jiaotong University *Xi'an, Shannxi*

RESEARCH EXPERIENCE

Graduate Research Assistant **2016–2021**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

Teaching Assistant **Jan. 2021–Mar. 2021**
University of California, Irvine *Irvine, California*

REFEREED JOURNAL PUBLICATIONS

On the Sub-Packetization Size and the Repair Bandwidth of Reed-Solomon Codes **2019**
IEEE Transactions on Information Theory

Repairing Reed-Solomon Codes Over $GF(2^{\ell})$ **2019**
IEEE Communications Letters

Storage Codes with Flexible Number of Nodes **2021**
Submitted to IEEE Transactions on Information Theory

REFEREED CONFERENCE PUBLICATIONS

A tradeoff between the sub-packetization size and the repair bandwidth for Reed-Solomon code **2017**
55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)

On the I/O costs in repairing short-length Reed-Solomon codes **2019**
IEEE International Symposium on Information Theory (ISIT)

Flexible Partial MDS Codes **2021**
12TH Annual Non-Volatile Memories Workshop (NVMW)

Flexible Constructions for Distributed Matrix Multiplication **2021**
IEEE International Symposium on Information Theory (ISIT)

ABSTRACT OF THE DISSERTATION

Flexible Coding for Distributed Systems

By

Weiqi Li

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2021

Chancellor's Professor Hamid Jafarkhani, Chair
Assistant Professor Zhiying Wang, Chair

In this dissertation, the constructions and schemes for flexible coding in distributed systems are investigated. Depending on the system parameters, the proposed methods allow adaptive choices of code constructions or data reconstruction schemes that provide desirable cost functions, such as network traffic, complexity, and latency. First, the repair of a node failure for Reed-Solomon (RS) codes, one of the most popular codes used in practical storage systems, is considered. Code constructions and repair schemes that provide a tradeoff between the repair communication cost and the coding complexity are presented. Second, facing the fact that the failures are unpredictable in a distributed system, a framework for flexible codes to achieve the optimal latency of accessing information is proposed. Instead of accessing a fixed number of nodes with a conventional code, a flexible code allows one to recover the entire information from a flexible number of storage nodes, and use all the available nodes efficiently. Constructions for different storage scenarios are proposed. Third, flexible coding in distributed matrix multiplication for failure tolerance is presented to reduce the computing latency. The goal is to allow a master node to efficiently obtain the computation results from a flexible number of available servers. The number of failures is unknown a priori and we provide code constructions that can efficiently make use of the computation results from all available servers. Given the storage capacity of the servers, the computation load optimization problem is analyzed.

Chapter 1

Introduction

Due to the high demand for accessing and storing large amount of data, distributed systems consisting of hundreds of thousands of devices are widely used. In distributed systems, failures are quite common [84] and hard to predict. To protect data from failures, error-correcting codes are ubiquitous for data storage systems such as Google File System [28] and Facebook's Warm BLOB [68], since it requires only a small storage overhead compared to simple replication of data. By using error-correcting codes, data is divided into fragments and encoded with redundant fragments to be stored in a set of nodes. If some nodes fail, either permanently or transiently, the data can be recovered from the fragments stored in the other available nodes. However, data reconstruction and failure recovery is costly in terms of latency, data access, node activation, network traffic, power consumption, etc. In this dissertation, we investigate the communication cost between nodes when failures are repaired, the constructions of error-correcting codes to protect data from a flexible number of failures, and the application of error-correcting codes to distributed computing.

We first consider the repair problem of the Reed-Solomon (RS) codes [80]. RS code is one of the most commonly used codes because it achieves lowest redundancy for given failure tolerance and has efficient encoding and decoding methods, see, e.g., [83, 39]. The RS code can be viewed

as a polynomial over a finite field $GF(q^\ell)$ evaluated at a set of points, where ℓ is called the *sub-packetization size*. The amount of transmitted information from surviving nodes to repair a failed node is defined as the *repair bandwidth*. Smaller ℓ facilitates the implementation of RS codes with lower complexity, however, when ℓ is small, say $\ell = 1$, the data in surviving nodes is hard to divide into smaller pieces thus result in a bigger repair bandwidth compared to the lost information. To reduces the network traffic in distributed storage, the optimal bandwidth for RS code [94] is achieved while ℓ is exponentially large with the code length. The tradeoff between these the sub-packetization size and the repair bandwidth remains an open problem. Several works [17, 67, 114, 113, 106] in the literature provide constructions with different sub-packetization sizes and the repair bandwidth. In Chapter 2, we present code constructions and repair schemes to provide a flexible tradeoff between the sub-packetization size and the repair bandwidth. In addition, we generalize our schemes to manage multiple failures.

We also consider the reconstruction of the entire information in the presence of a flexible number of failures. In the literature, most of the codes have a fixed redundancy level, while in practical systems, the number of failures varies over time. When the number of failures is smaller than the designed redundancy level, the redundant storage nodes are not used efficiently, resulting in an unnecessarily large latency. In Chapter 3, we present *flexible storage codes*, a class of error-correcting codes that can recover information from a flexible number of storage nodes. The main idea is that fast servers can access more information symbols to compensate for the effect of the slow servers, and thus reduce the latency. Code constructions for different storage scenarios are presented, including LRC (locally recoverable) codes, PMDS (partial MDS) codes, and MSR (minimum storage regenerating) codes. We analyze the latency of accessing information and perform simulations on Amazon clusters to show the efficiency of the presented codes.

Large-scale matrix multiplication is applied in big data computing required by many applications like machine learning problems. To parallelize the computation, the computation tasks are divided and encoded into multiple severs and the required results can be obtained from the computation

of the servers. Codes have been shown to be effective for combating slow servers in such systems [57]. In Chapter 4, the distributed matrix multiplication problem with unknown number of stragglers is considered, where the goal is to allow a master to efficiently and flexibly obtain the product of two massive matrices by distributing the computation across N servers. We assume there are at most $N - R$ stragglers but the exact number is not known a priori. Motivated by reducing the latency, a flexible solution is proposed to fully utilize the computation capability of available servers. The computing job for each server is separated into several tasks, constructed based on Entangled Polynomial (EP) codes by Yu el al [110]. The final results can be obtained by a flexible number of servers, while the number of tasks required in each server adjusts to the number of available servers. The required finite field size of the proposed solution is less than $2N$. Moreover, the optimal partitioning of the input matrices is discussed. Our constructions can also be generalized to secure computing, where the servers are not able to know any information about the original tasks, and batch matrix multiplication, where a batch of matrices are multiplied together.

Chapter 2

On the Sub-Packetization Size and the Repair Bandwidth of Reed-Solomon Codes

2.1 Introduction

Erasure codes are ubiquitous in distributed storage systems because they can efficiently store data while protecting against failures. Reed-Solomon (RS) code is one of the most commonly used codes because it achieves the Singleton bound [65] and has efficient encoding and decoding methods, see, e.g., [83, 39]. Codes matching the Singleton bound are called maximum distance separable (MDS) codes, and they have the highest possible failure-correction capability for a given redundancy level. In distributed storage, every code word symbol corresponds to a storage node, and communication costs between storage nodes need to be considered when node failures are repaired. In this chapter, we study the *repair bandwidth* of RS codes, defined as the amount of transmission required to repair a single node erasure, or failure, from all the remaining nodes (called helper nodes).

For a given erasure code, when each node corresponds to a single finite field symbol over $\mathbb{F} =$

$GF(q^\ell)$, we say the code is scalar; when each node is a vector of finite field symbols in $\mathbb{B} = GF(q)$ of length ℓ, it is called a vector code or an array code. In both cases, we say the *sub-packetization size* of the code is ℓ. Here q is a power of a prime number. Shanmugam et al. [86] considered the repair of scalar codes for the first time. Recently, Guruswami and Wootters [40] proposed a repair scheme for RS codes. The key idea of both chapters is that: rather than directly using the helper nodes as symbols over $\mathbb{F}$ to repair the failed node, one treats them as vectors over the subfield $\mathbb{B}$. Thus, a helper may transmit less than ℓ symbols over $\mathbb{B}$, resulting in a reduced bandwidth. For an RS code with length n and dimension k over the field $\mathbb{F}$, denoted by $RS(n, k)$, [40] achieves the repair bandwidth of $n-1$ symbols over $\mathbb{B}$. Moreover, when $n = q^\ell$ (called the full-length RS code) and $n - k = q^{\ell-1}$, the scheme provides the optimal repair bandwidth. Dau and Milenkovic [18] improved the scheme such that the repair bandwidth is optimal for the full-length RS code and any $n - k = q^s, 1 \leq s \leq \log_q(n - k)$.

For the full-length RS code, the schemes in [40] and [18] are optimal for single erasure. However, the repair bandwidth of these schemes still has a big gap from the minimum storage regenerating (MSR) bound derived in [20]. In particular, for an arbitrary MDS code, the repair bandwidth b, measured in the number of symbols over $GF(q)$, is lower bounded by

$$b \geq \frac{\ell(n-1)}{n-k}. \tag{2.1}$$

An MDS code satisfying the above bound is called an MSR code. In fact, most known MSR codes are vector codes, see [77, 71, 93, 99, 78, 34, 105]. For the repair of RS codes, Ye and Barg proposed a scheme that asymptotically approaches the MSR bound as n grows [103] when the sub-packetization size is $\ell = (n - k)^n$. Tamo et al. [94] provided an RS code repair scheme achieving the MSR bound when the sub-packetization size is $\ell \approx n^n$.

The repair problem for RS codes can also be generalized to multiple erasures. In this case, the schemes in [17] and [67] work for the full-length code, [114] and [113] work for centralized repair,

and [106] proposed a scheme achieving the multiple-erasure MSR bound.

Motivation: A flexible tradeoff between the sub-packetization size and the repair bandwidth is an open problem: Only the full-length RS code with high repair bandwidth and the MSR-achieving RS code with large sub-packetization are established. Our chapter aims to provide more points between the two extremes – the full-length code and the MSR code. One straightforward method is to apply the schemes of [40] and [18] to the case of $\ell > \log_q n$ with fixed (n, k). However, the resulting normalized repair bandwidth $\frac{b}{\ell(n-1)}$ grows with ℓ, contradictory to our intuition that larger ℓ implies smaller normalized bandwidth.

The need for small repair bandwidth is motivated by reducing the network traffic in distributed storage [20], and the need for the small sub-packetization is due to the complexity in field arithmetic operations, discussed below. It is demonstrated that the time complexity of multiplications in larger fields are much higher than that of smaller fields [31]. Moreover, multiplication in Galois fields are usually done by pre-computed look-up tables and the growing field size has a significant impact on the space complexity of multiplication operations. Larger fields require huge memories for the look-up table. For example, in $GF(2^{16})$, 8 GB are required for the complete table, which is impractical in most current systems [36]. Some logarithm tables and sub-tables are used to alleviate the memory problems for large fields, while increasing the time complexity at the same time [36], [74], [64]. For example, in the Intel SIMD methods, multiplications over $GF(2^{16})$ need twice the amount of operations as over $GF(2^8)$, and multiplications over $GF(2^{32})$ need 4 times the amount of operations compared to $GF(2^8)$, which causes the multiplication speed to drop significantly when the field size grows [74].

To illustrate the impact of the sub-packetization size on the complexity, let us take encoding for example. To encode a single parity check node, we need to do k multiplications and k additions over $GF(q^\ell)$. For a given systematic $RS(n, k)$ code over $GF(q^\ell)$, we can encode $k\ell \log_2 q$ bits of information by multiplications of $(n-k)k\ell \log_2 q$ bits and additions of $(n-k)k\ell \log_2 q$ bits. So, when M bits are encoded into $RS(n, k)$ codes, we need $M/(k\ell \log_2 q)$ copies of the code and we

need multiplications of $M(n-k)$ bits and additions of $M(n-k)$ bits in $GF(q^\ell)$ in total. Although the total amount of bits we need to multiply is independent of ℓ, the complexity over a larger field is higher in both time and space. For a simulation of the RS code speed using different field sizes on different platforms, we refer the readers to [75]. The results suggest that RS codes have faster implementation in both encoding and decoding for smaller fields.

Besides the complexity, the small sub-packetization level also has many advantages such as easy system implementation, great flexibility and bandwidth-efficient access to missing small files [38], [51], which makes it important in distributed storage applications.

As can be seen from the two extremes, a small sub-packetization level also means higher costs in repair bandwidth, and not many other codes are known besides the extremes. For vector codes, Guruswami, Rawat [38] and Li, Tang [59] provided small sub-packetization codes with small repair bandwidth, but only for single erasure. Kralevska et al. [55] also presented a tradeoff between the sub-packetization level and the repair bandwidth for the proposed HashTag codes implemented in Hadoop. For scalar codes, Chowdhury and Vardy [16] extended Ye and Barg's MSR scheme [103] to a smaller sub-packetization size, but it only works for certain redundancy r and single erasure.

Contributions: In this work, we first design three single-erasure RS repair schemes, using the cosets of the multiplicative group of the finite field $\mathbb{F}$. Note that the RS code can be viewed as n evaluations of a polynomial over $\mathbb{F}$. The evaluation points of the three schemes are part of one coset, of two cosets, and of multiple cosets, respectively, so that the evaluation point size can vary from a very small number to the whole field size. In the schemes designed in this chapter, we have a parameter a that can be tuned, and provides a tradeoff between the sub-packetization size and the repair bandwidth.

- For an $RS(n,k)$ code, our first scheme achieves the repair bandwidth $\frac{\ell}{a}(n-1)(a-s)$ for some a, s such that $n < q^a, r \triangleq n-k > q^s$ and a divides ℓ. Specifically, for the $RS(14,10)$ code used in Facebook [84], we achieve repair bandwidth of 52 bits with $\ell = 8$, which is

35% better than the naive repair scheme.

- Our second scheme reaches the repair bandwidth of $(n-1)\frac{\ell+a}{2}$ for some a such that $n \leq 2(q^a-1)$, a divides ℓ and $\frac{\ell}{r} < a$.
- The first realization of our third scheme attains the repair bandwidth of $\frac{\ell}{r}(n+1+(r-1)(q^a-2))$ when $n \leq (q^a-1)\log_r \frac{\ell}{a}$. Another realization of the third scheme attains the repair bandwidth of $\frac{\ell}{r}(n-1+(r-1)(q^a-2))$ where $\ell \approx a(\frac{n}{q^a-1})^{(\frac{n}{q^a-1})}$. The second realization can also be generalized to any d helpers, for $k \leq d \leq n-1$.

We provide characterizations of linear multiple-erasure repair schemes, and propose two schemes for multiple erasures, where the evaluation points are in one coset and in multiple cosets, respectively. Again, the parameter a is tunable.

- We prove that any linear repair scheme for multiple erasures in a scalar MDS code is equivalent to finding a set of dual codewords satisfying certain rank constraints.
- For an $RS(n,k)$ code with $e < \frac{1}{a-s}\sqrt{\log_q n}$ erasures, our first scheme achieves the repair bandwidth $\frac{e\ell}{a}(n-e)(a-s)$ for some a, s such that $n < q^a, r = n-k > q^s$ and a divides ℓ.
- For an $RS(n,k)$ code, our second scheme works for $e \leq n-k$ erasures and $n-e$ helpers. The repair bandwidth depends on the location of the erasures and in most cases, we achieve $\frac{e\ell}{d-k+e}(n-e+(n-k+e)(q^a-2))$ where $\ell \approx a(\frac{n}{q^a-1})^{(\frac{n}{q^a-1})}$ and a divides ℓ.
- We demonstrate that repairing multiple erasures simultaneously is advantageous compared to repairing single erasures separately.

The comparison of our schemes, as well as the comparison to previous works, are shown in Tables 2.1 and 2.2, and are discussed in more details in Sections 2.3.4 and 2.4.4.

The chapter is organized as follows. In Section II, we briefly review the linear repair of RS codes and provide the preliminaries used in this chapter. In Section III, we present three RS repair schemes for single erasure. Then, we discuss the repair schemes for multiple erasures in Section IV. In Section V, we provide the conclusion.

Table 2.1: Comparison of different schemes for single erasure. When $a = \ell$, our scheme in one coset is the scheme in [40], [18]. When $a = 1$, our schemes in multiple cosets is the schemes in [103], [94].

	repair bandwidth	code length	restrictions
Schemes in [40], [18]	$(n-1)(\ell - s)$	$n \leq q^{\ell}$	$q^s \leq r$
Scheme in [103]	$< \frac{\ell}{r}(n+1)$	$n = \log_r \ell$	
Scheme in [94]	$\frac{\ell}{r}(n-1)$	$n^n \approx \ell$	
Our scheme in one coset	$\leq \frac{\ell}{a}(n-1)(a-s)$	$n \leq (q^a - 1)$	$q^s \leq r, a \vert \ell$
Our scheme in two cosets	$< (n-1)\frac{\ell+a}{2}$	$n \leq 2(q^a - 1)$	$\frac{\ell}{r} \leq a, a \vert \ell$
Our scheme in multiple cosets 1	$< \frac{\ell}{r}(n+1+(r-1)(q^a-2))$	$n \leq (q^a - 1)m$	$\ell/a = r^m$ for some integer m
Our scheme in multiple cosets 2	$\frac{\ell}{r}(n-1+(r-1)(q^a-2))$	$n \leq (q^a - 1)m$	$\ell/a \approx m^m$ for some integer m

Table 2.2: Comparison of different schemes for multiple erasures. When $a = \ell$ and $s = \ell$ our scheme in one coset is the scheme 1 in [67]. When $a = 1$, our schemes in multiple cosets is the scheme in [106].

	repair bandwidth	code length	restrictions
Scheme 1 in [67]	$\leq (n-e)e - \frac{e(e-1)(q-1)}{2}$	$n \leq q^{\ell}$	$q^{\ell-1} \leq r, e < \sqrt{\log_q n}$
Scheme 2 in [67]	$\leq \min_{e' \geq e}((n-e')(\ell - \lfloor \log_q(\frac{n-k+e'-1}{2e'-1}) \rfloor))$	$n \leq q^{\ell}$	
Scheme in [106]	$\frac{ed\ell}{d-k+e}$	$n^n \approx \ell$	
Our scheme for multiple erasures in one coset	$\leq \frac{e\ell}{a}(n-e)(a-s)$	$n \leq (q^a - 1)$	$q^s \leq r, a \vert \ell, e < \frac{1}{a-s}\sqrt{\log_q n}$
Our scheme for multiple erasures in multiple cosets	$\frac{e\ell}{n-k}(n-e+(n-k+e)(q^a-2))$	$n \leq (q^a - 1)m$	$\ell/a \approx m^m$ for some integer m

Notation: Throughout this chapter, for positive integer i, we use $[i]$ to denote the set $\{1, 2, \ldots, i\}$. For integers a, b, we use $a \mid b$ to denote that a divides b. For real numbers a_n, b_n, which are functions of n, we use $a \approx b$ to denote $\lim_{n\to\infty} \frac{a_n}{b_n} = 1$. For sets $A \subseteq B$, we use B/A to denote the difference of A from B. For a finite field $\mathbb{F}$, we denote by $\mathbb{F}^* = \mathbb{F}/\{0\}$ the corresponding multiplicative group. We write $\mathbb{E} \leq \mathbb{F}$ for $\mathbb{E}$ being a subfield of $\mathbb{F}$. For element $\beta \in \mathbb{F}$ and E as a subset of $\mathbb{F}$, we denote $\beta E = \{\beta s, \forall s \in E\}$. A^T denotes the transpose of the matrix A.

2.2 Preliminaries

In this section, we review the linear repair scheme of RS code in [40], and provide a basic lemma used in our proposed schemes.

The *Reed-Solomon code* $RS(A, k)$ over $\mathbb{F} = GF(q^\ell)$ of dimension k with n evaluation points $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq \mathbb{F}$ is defined as

$$RS(A, k) = \{(f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n)) : f \in \mathbb{F}[x], \deg(f) \leq k - 1\},$$

where $\deg()$ denotes the degree of a polynomial, $f(x) = u_0 + u_1 x + u_2 x^2 + \cdots + u_{k-1} x^{k-1}$, and $u_i \in \mathbb{F}, i = 0, 1, \ldots, k-1$ are the messages. Every evaluation symbol $f(\alpha), \alpha \in A$, is called a code word symbol or a storage node. The *sub-packetization size* is defined as ℓ, and $r \triangleq n - k$ denotes the number of parity symbols.

Assume e nodes fail, $e \leq n - k$, and we want to recover them. The number of helper nodes are denoted by d. The amount of information transmitted from the helper nodes is defined as the *repair bandwidth* b, measured in the number of symbols over $GF(q)$. All the remaining $n - e = d$ nodes are assumed to be the helper nodes unless stated otherwise. We define the *normalized repair bandwidth* as $\frac{b}{\ell d}$, which is the average fraction of information transmitted from each helper. By

[20, 12], the minimum storage regenerating (MSR) bound for the bandwidth is

$$b \geq \frac{e\ell d}{d-k+e}. \tag{2.2}$$

As mentioned before, codes achieving the MSR bound require large sub-packetization sizes. In this section, we focus on the single erasure case.

Assume $\mathbb{B} \leq \mathbb{F}$, namely, $\mathbb{B}$ is a subfield of $\mathbb{F}$. A linear repair scheme requires some symbols of the subfield $\mathbb{B}$ to be transmitted from each helper node [40]. If the symbols from the same helper node are linearly dependent, the repair bandwidth decreases. In particular, the scheme uses dual code to compute the failed node and uses trace function to obtain the transmitted subfield symbols, as detailed below.

Assume $f(\alpha^*)$ fails for some $\alpha^* \in A$. For any polynomial $p(x) \in \mathbb{F}[x]$ of which the degree is smaller than r, $(v_1 p(\alpha_1), v_2 p(\alpha_2), \ldots, v_n p(\alpha_n))$ is a dual codeword of $RS(A,k)$, where $v_i, i \in [n]$ are non-zero constants determined by the set A (see for example [65, Thm. 4 in Ch.10]). We can thus repair the failed node $f(\alpha^*)$ from

$$v_{\alpha^*} p(\alpha^*) f(\alpha^*) = -\sum_{i=1,\alpha_i \neq \alpha^*}^{n} v_i p(\alpha_i) f(\alpha_i) \tag{2.3}$$

The summation on the right side means that we add all the i elements from $i = 1$ to $i = n$ except when $\alpha_i \neq \alpha^*$.

The trace function from $\mathbb{F}$ onto $\mathbb{B}$ is defined as

$$tr_{\mathbb{F}/\mathbb{B}}(\beta) = \beta + \beta^q + \cdots + \beta^{q^{\ell-1}}, \tag{2.4}$$

where $\beta \in \mathbb{F}$, $\mathbb{B} = GF(q)$ is called the *base field*, and q is a power of a prime number. It is a linear

mapping from $\mathbb{F}$ to $\mathbb{B}$ and satisfies

$$tr_{\mathbb{F}/\mathbb{B}}(\alpha\beta) = \alpha tr_{\mathbb{F}/\mathbb{B}}(\beta) \tag{2.5}$$

for all $\alpha \in \mathbb{B}$.

We define the rank $rank_{\mathbb{B}}(\{\gamma_1, \gamma_2, ..., \gamma_i\})$ to be the cardinality of a maximal subset of $\{\gamma_1, \gamma_2, ..., \gamma_i\}$ that is linearly independent over $\mathbb{B}$. For example, for $\mathbb{B} = GF(2)$ and $\alpha \notin \mathbb{B}$, $rank_{\mathbb{B}}(\{1, \alpha, 1+\alpha\}) = 2$ because the subset $\{1, \alpha\}$ is the maximal subset that is linearly independent over $\mathbb{B}$ and the cardinality of the subset is 2.

Assume we use polynomials $p_j(x)$, $j \in [\ell]$ to generate ℓ different dual codewords, called *repair polynomials*. Combining the trace function and the dual code, we have

$$tr_{\mathbb{F}/\mathbb{B}}(\upsilon_{\alpha^*} p_j(\alpha^*) f(\alpha^*)) = -\sum_{i=1, \alpha_i \neq \alpha^*}^{n} tr_{\mathbb{F}/\mathbb{B}}(\upsilon_i p_j(\alpha_i) f(\alpha_i)). \tag{2.6}$$

In a repair scheme, the helper $f(\alpha_i)$ transmits

$$\{tr_{\mathbb{F}/\mathbb{B}}(\upsilon_i p_j(\alpha_i) f(\alpha_i)) : j \in [\ell]\}. \tag{2.7}$$

Suppose $\{\upsilon_{\alpha^*} p_1(\alpha^*), \upsilon_{\alpha^*} p_2(\alpha^*), \ldots, \upsilon_{\alpha^*} p_\ell(\alpha^*)\}$ is a basis for $\mathbb{F}$ over $\mathbb{B}$, and assume $\{\mu_1, \mu_2, \ldots, \mu_\ell\}$ is its dual basis. Then, $f(\alpha^*)$ can be repaired by

$$f(\alpha^*) = \sum_{j=1}^{\ell} \mu_j tr_{\mathbb{F}/\mathbb{B}}(\upsilon_{\alpha^*} p_j(\alpha^*) f(\alpha^*)). \tag{2.8}$$

Since υ_{α^*} is a non-zero constant, we equivalently suppose that $\{p_1(\alpha^*), \ldots, p_\ell(\alpha^*)\}$ is a basis.

In fact, by [40] any linear repair scheme of RS code for the failed node $f(\alpha^*)$ is equivalent to choosing $p_j(x), j \in [\ell]$, with degree smaller than r, such that $\{p_1(\alpha^*), \ldots, p_\ell(\alpha^*)\}$ forms a basis

for $\mathbb{F}$ over $\mathbb{B}$. We call this the **full rank condition**:

$$rank_{\mathbb{B}}(\{p_1(\alpha^*), p_2(\alpha^*), \ldots, p_\ell(\alpha^*)\}) = \ell. \tag{2.9}$$

The repair bandwidth can be calculated from (2.7) and by noting that $v_i f(\alpha_i)$ is a constant:

$$b = \sum_{\alpha \in A, \alpha \neq \alpha^*} rank_{\mathbb{B}}(\{p_1(\alpha), p_2(\alpha), \ldots, p_\ell(\alpha)\}). \tag{2.10}$$

We call this the **repair bandwidth condition**.

The goal of a good RS code construction and its repair scheme is to choose appropriate evaluation points A and polynomials $p_j(x), j \in [\ell]$, that can reduce the repair bandwidth in (2.10) while satisfying (2.9).

The following lemma is due to the structure of the multiplicative group of $\mathbb{F}$, which will be used for finding the evaluation points in the code constructions in this chapter. Similar statements can be found in [83, Ch. 2.6].

Lemma 1. Assume $\mathbb{E} \leq \mathbb{F} = GF(q^\ell)$, then $\mathbb{F}^*$ can be partitioned to $t \triangleq \frac{q^\ell - 1}{|\mathbb{E}| - 1}$ cosets: $\{\mathbb{E}^*, \beta\mathbb{E}^*, \beta^2\mathbb{E}^*, \ldots, \beta^{t-1}\mathbb{E}^*\}$, where β is a primitive element of $\mathbb{F}$.

Proof: The $q^\ell - 1$ elements in $\mathbb{F}^*$ are $\{1, \beta, \beta^2, \ldots, \beta^{q^\ell - 2}\}$ and $\mathbb{E}^* \subseteq \mathbb{F}^*$. Assume that t is the smallest nonzero number that satisfies $\beta^t \in \mathbb{E}^*$, then we know that $\beta^k \in \mathbb{E}^*$ if and only if $t|k$. Also, $\beta^{k_1} \neq \beta^{k_2}$ when $k_1 \neq k_2$ and $k_1, k_2 < q^\ell - 2$. Since there are only $|\mathbb{E}| - 1$ nonzero distinct elements in $\mathbb{E}^*$ and $\beta^{q^\ell - 1} = 1$, we have $t = \frac{q^\ell - 1}{|\mathbb{E}| - 1}$ and the t cosets are $\mathbb{E}^* = \{1, \beta^t, \beta^{2t}, \ldots, \beta^{(|\mathbb{E}|-2)t}\}$, $\beta\mathbb{E}^* = \{\beta, \beta^{t+1}, \beta^{2t+1}, \ldots, \beta^{(|\mathbb{E}|-2)t+1}\}, \ldots, \beta^{t-1}\mathbb{E}^* = \{\beta^{t-1}, \beta^{2t-1}, \beta^{3t-1}, \ldots, \beta^{(|\mathbb{E}|-1)t-1}\}$. ■

2.3 Reed-Solomon Repair Schemes for Single Erasure

In this section, we present our schemes in which the evaluation points are part of one coset, two cosets and multiple cosets for a single erasure. From these constructions, we achieve several different points on the tradeoff between the sub-packetization size and the normalized repair bandwidth. The main ideas of the constructions are:

(i) In all our schemes, we take an original RS code, and construct a new code over a larger finite field. Thus, the sub-packetization size ℓ is increased.

(ii) For the schemes using one and two cosets, the code parameters n, k are kept the same as the original code. Hence, for given $n, r = n - k$, the sub-packetization size ℓ increases, but we show that the normalized repair bandwidth remains the same.

(iii) For the scheme using multiple cosets, the code length n is increased and the redundancy r is fixed. Moreover, the code length n grows faster than the sub-packetization size ℓ. Therefore, for fixed n, r, the sub-packetization ℓ decreases, and we show that the normalized repair bandwidth is only slightly larger than the original code.

2.3.1 Schemes in one coset

Assume $\mathbb{E} = GF(q^a)$ is a subfield of $\mathbb{F} = GF(q^\ell)$ and $\mathbb{B} = GF(q)$ is the base field, where q is a prime number. The evaluation points of the code over $\mathbb{F}$ that we construct are part of one coset in Lemma 1.

We first present the following lemma about the basis.

Lemma 2. Assume $\{\xi_1, \xi_2, \ldots, \xi_\ell\}$ is a basis for $\mathbb{F} = GF(q^\ell)$ over $\mathbb{B} = GF(q)$, then $\{\xi_1^{q^s}, \xi_2^{q^s}, \ldots, \xi_\ell^{q^s}\}$, $s \in [\ell]$ is also a basis.

Proof: Assume $\{\xi_1^{q^s}, \xi_2^{q^s}, \ldots, \xi_\ell^{q^s}\}$, $s \in [\ell]$ is not a basis for $\mathbb{F}$ over $\mathbb{B}$, then there exist nonzero $(\alpha_1, \alpha_2, \ldots, \alpha_\ell), \alpha_i \in \mathbb{B}, i \in [\ell]$, that satisfy

$$\begin{aligned}
&\alpha_1\xi_1^{q^s} + \alpha_2\xi_2^{q^s} + \cdots + \alpha_\ell\xi_\ell^{q^s} \\
=&0 \\
=&(\alpha_1\xi_1 + \alpha_2\xi_2 + \cdots + \alpha_\ell\xi_\ell)^{q^s},
\end{aligned} \tag{2.11}$$

which is in contradiction to the assumption that $\{\xi_1, \xi_2, \ldots, \xi_\ell\}$ is a basis for $\mathbb{F}$ over $\mathbb{B}$. ■

The following theorem shows the repair scheme using one coset for the evaluation points.

Theorem 1. There exists an $RS(n, k)$ code over $\mathbb{F} = GF(q^\ell)$ with repair bandwidth $b \leq \frac{\ell}{a}(n-1)(a-s)$ symbols over $\mathbb{B} = GF(q)$, where q is a prime number and a, s satisfy $n < q^a, q^s \leq n-k$, $a|\ell$.

Proof: Assume a field $\mathbb{F} = GF(q^\ell)$ is extended from $\mathbb{E} = GF(q^a)$, $a \mid \ell$, and β is a primitive element of $\mathbb{F}$. We focus on the code $RS(A, k)$ of dimension k over $\mathbb{F}$ with evaluation points $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq \beta^m\mathbb{E}^*$ for some $0 \leq m < \frac{q^\ell-1}{q^a-1}$, which is one of the cosets in Lemma 1. The base field is $\mathbb{B} = GF(q)$ and (2.6) is used to repair the failed node $f(\alpha^*)$.

Construction I: Inspired by [40], for $s = a - 1$, we choose

$$p_j(x) = \frac{tr_{\mathbb{E}/\mathbb{B}}(\xi_j(\frac{x}{\beta^m} - \frac{\alpha^*}{\beta^m}))}{\frac{x}{\beta^m} - \frac{\alpha^*}{\beta^m}}, j \in [a], \tag{2.12}$$

where $\{\xi_1, \xi_2, \ldots, \xi_a\}$ is a basis for $\mathbb{E}$ over $\mathbb{B}$. The degree of $p_j(x)$ is smaller than r since $q^s \leq r$. When $x = \alpha^*$, by (2.4) we have

$$p_j(\alpha^*) = \xi_j. \tag{2.13}$$

So, the polynomials satisfy

$$rank_{\mathbb{B}}(\{p_1(\alpha^*), p_2(\alpha^*), \ldots, p_a(\alpha^*)\}) = a. \tag{2.14}$$

When $x \neq \alpha^*$, since $tr_{\mathbb{E}/\mathbb{B}}(\xi_j(\frac{x}{\beta^m} - \frac{\alpha^*}{\beta^m})) \in \mathbb{B}$, and $\frac{x}{\beta^m} - \frac{\alpha^*}{\beta^m}$ is a constant independent of j, we have

$$rank_{\mathbb{B}}(\{p_1(x), p_2(x), \ldots, p_a(x)\}) = 1. \tag{2.15}$$

Let $\{\eta_1, \eta_2, \eta_3, \ldots, \eta_{\ell/a}\}$ be a basis for $\mathbb{F}$ over $\mathbb{E}$, the ℓ repair polynomials are chosen as

$$\{\eta_1 p_j(x), \eta_2 p_j(x), \ldots, \eta_{\ell/a} p_j(x) : j \in [a]\}. \tag{2.16}$$

Since $p_j(x) \in \mathbb{E}$, we can conclude that

$$\begin{aligned} &rank_{\mathbb{B}}(\{\eta_1 p_j(\alpha^*), \eta_2 p_j(\alpha^*), \ldots, \eta_{\ell/a} p_j(\alpha^*) : j \in [a]\}) \\ &\quad = \frac{\ell}{a} rank_{\mathbb{B}}(\{p_1(\alpha^*), p_2(\alpha^*), \ldots, p_a(\alpha^*)\}) = \ell \end{aligned} \tag{2.17}$$

satisfies the full rank condition, and for $x \neq \alpha^*$

$$\begin{aligned} &rank_{\mathbb{B}}(\{\eta_1 p_j(x), \eta_2 p_j(x), \ldots, \eta_{\ell/a} p_j(x) : j \in [a]\}) \\ &\quad = \frac{\ell}{a} rank_{\mathbb{B}}(\{p_1(x), p_2(x), \ldots, p_a(x)\}) = \frac{\ell}{a}. \end{aligned} \tag{2.18}$$

From (2.10) we can calculate the repair bandwidth

$$b = \frac{\ell}{a}(n-1). \tag{2.19}$$

Construction II: For $s \leq a-1$, inspired by [18], we choose

$$p_j(x) = \xi_j \prod_{i=1}^{q^s-1} \left(\frac{x}{\beta^m} - \left(\frac{\alpha^*}{\beta^m} - w_i^{-1}\xi_j \right) \right), j \in [a], \tag{2.20}$$

where $\{\xi_1, \xi_2, \ldots, \xi_a\}$ is a basis for $\mathbb{E}$ over $\mathbb{B}$, and $W = \{w_0 = 0, w_1, w_2, \ldots, w_{q^s-1}\}$ is an s-dimensional subspace in $\mathbb{E}$, $s < a, q^s \leq r$. It is easy to check that the degree of $p_j(x)$ is smaller than r since $q^s \leq r$. When $x = \alpha^*$, we have

$$p_j(\alpha^*) = \xi_j^{q^s} \prod_{i=1}^{q^s-1} w_i^{-1}. \tag{2.21}$$

Since $\prod_{i=1}^{q^s-1} w_i^{-1}$ is a constant, from Lemma 2 we have

$$rank_{\mathbb{B}}(\{p_1(\alpha^*), p_2(\alpha^*), \ldots, p_a(\alpha^*)\}) = a. \tag{2.22}$$

For $x \neq \alpha^*$, set $x' = \frac{\alpha^*}{\beta^m} - \frac{x}{\beta^m} \in \mathbb{E}$, we have

$$\begin{aligned} p_j(x) &= \xi_j \prod_{i=1}^{q^s-1} \left(\frac{x}{\beta^m} - \left(\frac{\alpha^*}{\beta^m} - w_i^{-1}\xi_j \right) \right) \\ &= \xi_j \prod_{i=1}^{q^s-1} (w_i^{-1}\xi_j - x') \\ &= \xi_j \prod_{i=1}^{q^s-1} (w_i^{-1}x') \prod_{i=1}^{q^s-1} (\xi_j/x' - w_i) \\ &= (x')^{q^s} \prod_{i=1}^{q^s-1} (w_i^{-1}) \prod_{i=0}^{q^s-1} (\xi_j/x' - w_i). \end{aligned} \tag{2.23}$$

By [35, p. 4], $g(y) = \prod_{i=0}^{q^s-1} (y - w_i)$ is a linear mapping from $\mathbb{E}$ to itself with dimension $a - s$ over

$\mathbb{B}$. Since $(x')^{q^s}\prod_{i=1}^{q^s-1}(w_i^{-1})$ is a constant independent of j, we have

$$rank_{\mathbb{B}}(\{p_1(x), p_2(x), \ldots, p_a(x)\}) \leq a - s. \tag{2.24}$$

Let $\{\eta_1, \eta_2, \eta_3, \ldots, \eta_{\ell/a}\}$ be a basis for $\mathbb{F}$ over $\mathbb{E}$, then the ℓ polynomials are chosen as $\{\eta_1 p_j(x)$, $\eta_2 p_j(x), \ldots, \eta_{\ell/a} p_j(x), j \in [a]\}$. From (2.21) and (2.23) we know that $p_j(x) \in \mathbb{E}$, so we can conclude that

$$\begin{aligned} &rank_{\mathbb{B}}(\{\eta_1 p_j(\alpha^*), \eta_2 p_j(\alpha^*), \ldots, \eta_{\ell/a} p_j(\alpha^*) : j \in [a]\}) \\ &\quad = \frac{\ell}{a} rank_{\mathbb{B}}(\{p_1(\alpha^*), p_2(\alpha^*), \ldots, p_a(\alpha^*)\}) = \ell \end{aligned} \tag{2.25}$$

satisfies (2.9), and for $x \neq \alpha^*$

$$\begin{aligned} &rank_{\mathbb{B}}(\{\eta_1 p_j(x), \eta_2 p_j(x), \ldots, \eta_{\ell/a} p_j(x) : j \in [a]\}) \\ &= \frac{\ell}{a} rank_{\mathbb{B}}(\{p_1(x), p_2(x), \ldots, p_a(x)\}) \leq \frac{\ell}{a}(a - s). \end{aligned} \tag{2.26}$$

Now from (2.10) we can calculate the repair bandwidth

$$b \leq \frac{\ell}{a}(n-1)(a-s). \tag{2.27}$$

Combining (2.19) and (2.27) will complete the proof of Theorem 1. ■

Rather than directly using the schemes in [40] and [18], the polynomials (2.12) and (2.20) that we use are similar to [40] and [18], respectively, but are mappings from $\mathbb{E}$ to $\mathbb{B}$. Moreover, we multiply each polynomial with the basis for $\mathbb{F}$ over $\mathbb{E}$ to satisfy the full rank condition. In this case, our scheme significantly reduces the repair bandwidth when the code length remains the same. Our evaluation points are in a coset rather than the entire field $\mathbb{F}$ as in [40] and [18]. It should be noted

that a here can be an arbitrary number that divides ℓ and when $a = \ell$, our schemes are exactly the same as those in [40] and [18]. Note that the normalized repair bandwidth $\frac{b}{\ell(n-1)}$ decreases as a decreases. Therefore, our scheme outperforms those in [40] and [18] when applied to the case of $\ell > \log_q n$.

Example 1. Assume $q = 2, \ell = 9, a = 3$ and $\mathbb{E} = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^6\}$. Let $A = \mathbb{E}^*$, $n = 7, k = 5$ so $r = n - k = 2$. Choose $s = \log_2 r = 1$ and $W = \{0, 1\}$ in Construction II. Then, we have $p_j(x) = \xi_j(x - \alpha^* + \xi_j)$. Let $\{\xi_1, \xi_2, \xi_3\}$ be $\{1, \alpha, \alpha^2\}$. It is easy to check that $rank_{\mathbb{B}}(\{p_1(\alpha^*), p_2(\alpha^*), p_3(\alpha^*)\}) = 3$ and $rank_{\mathbb{B}}(\{p_1(x), p_2(x), p_3(x)\}) = 2$ for $x \neq \alpha^*$. Therefore the repair bandwidth is $b = 36$ bits as suggested in Theorem 1. For the same (n, k, ℓ), the repair bandwidth in [18] is 48 bits. For another example, consider $RS(14, 10)$ code used in Facebook [84], we have repair bandwidth of 52 bits for $\ell = 8$, while [18] requires 60 bits and the naive scheme requires 80 bits.

Remark 1. The schemes in [40] and [18] can also be used in an RS code over $\mathbb{E}$ with repair bandwidth $(n - 1)(a - s)$, and with ℓ/a copies of the code. Thus, they can also reach the repair bandwidth of $\frac{\ell}{a}(n-1)(a-s)$. It should be noted that by doing so, the code is a vector code, however our scheme constructs a scalar code. To the best of our knowledge, this is the first example of such a scalar code in the literature.

2.3.2 Schemes in two cosets

Now we discuss our scheme when the evaluation points are chosen from two cosets. In this scheme, we choose the polynomials that have full rank when evaluated at the coset containing the failed node, and rank 1 when evaluated at the other coset.

Theorem 2. There exists an $RS(n, k)$ code over $\mathbb{F} = GF(q^\ell)$ with repair bandwidth $b < (n-1)\frac{\ell+a}{2}$ symbols over $\mathbb{B} = GF(q)$, where q is a prime number and a satisfies $n \leq 2(q^a - 1)$, $a|\ell$, $\frac{\ell}{a} \leq n - k$.

Proof: Assume a field $\mathbb{F} = GF(q^\ell)$ is extended from $\mathbb{E} = GF(q^a)$ and β is the primitive element of $\mathbb{F}$. We focus on the code $RS(A, k)$ over $\mathbb{F}$ of dimension k with evaluation points A consisting of $n/2$ points from $\beta^{m_1}\mathbb{E}^*$ and $n/2$ points from $\beta^{m_2}\mathbb{E}^*$, $0 \leq m_1 < m_2 \leq \frac{q^\ell - 1}{q^a - 1}$ and $m_2 - m_1 = q^s$, $s \in \{0, 1, \ldots, \frac{\ell}{a}\}$.

In this case we view $\mathbb{E}$ as the base field and repair the failed node $f(\alpha^*)$ by

$$tr_{\mathbb{F}/\mathbb{E}}(\upsilon_{\alpha^*} p_j(\alpha^*) f(\alpha^*)) = -\sum_{i=1, \alpha_i \neq \alpha^*}^{n} tr_{\mathbb{F}/\mathbb{E}}(\upsilon_i p_j(\alpha_i) f(\alpha_i)).$$

Inspired by [40, Theorem 10], for $j \in [\frac{\ell}{a}]$, we choose

$$p_j(x) = \begin{cases} (\frac{x}{\beta^{m_2}})^{j-1}, \text{ if } \alpha^* \in \beta^{m_1}\mathbb{E}^*, \\ (\frac{x}{\beta^{m_1}})^{j-1}, \text{ if } \alpha^* \in \beta^{m_2}\mathbb{E}^*. \end{cases} \tag{2.28}$$

The degree of $p_j(x)$ is smaller than r when $\frac{\ell}{a} \leq r$. Then, we check the rank in each case.

When $\alpha^* \in \beta^{m_2}\mathbb{E}^*$, if $x = \beta^{m_1}\gamma \in \beta^{m_1}\mathbb{E}^*$, for some $\gamma \in \mathbb{E}^*$,

$$p_j(x) = \left(\frac{x}{\beta^{m_1}}\right)^{j-1} = \gamma^{j-1}, \tag{2.29}$$

so

$$rank_{\mathbb{E}}(\{p_1(x), p_2(x), \ldots, p_{\frac{\ell}{a}}(x)\}) = 1. \tag{2.30}$$

If $x = \beta^{m_2}\gamma \in \beta^{m_2}\mathbb{E}^*$, for some $\gamma \in \mathbb{E}^*$,

$$p_j(x) = \left(\frac{x}{\beta^{m_1}}\right)^{j-1} = (\beta^{m_2 - m_1})^{j-1}\gamma^{j-1}. \tag{2.31}$$

Since $m_2 - m_1 = q^s$ and $\{1, \beta, \beta^2, \ldots, \beta^{\frac{\ell}{a}-1}\}$ is the polynomial basis for $\mathbb{F}$ over $\mathbb{E}$, from Lemma

2 we know that

$$rank_{\mathbb{E}}(\{p_1(x), p_2(x), \ldots, p_{\frac{\ell}{a}}(x)\}) = \frac{\ell}{a}. \tag{2.32}$$

When $\alpha^* \in \beta^{m_1}\mathbb{E}^*$, if $x = \beta^{m_1}\gamma \in \beta^{m_1}\mathbb{E}^*$, for some $\gamma \in \mathbb{E}^*$,

$$\begin{aligned} p_j(x) &= \left(\frac{x}{\beta^{m_2}}\right)^{j-1} \\ &= (\beta^{m_1-m_2})^{j-1}\gamma^{j-1} \\ &= (\beta^{m_2-m_1})^{1-\frac{\ell}{a}}(\beta^{m_2-m_1})^{\frac{\ell}{a}-j}\gamma^{j-1}. \end{aligned} \tag{2.33}$$

Since $(\beta^{m_2-m_1})^{1-\frac{\ell}{a}}$ is a constant, from Lemma 2 we know that

$$rank_{\mathbb{E}}(\{p_1(x), p_2(x), \ldots, p_{\frac{\ell}{a}}(x)\}) = \frac{\ell}{a}. \tag{2.34}$$

If $x = \beta^{m_2}\gamma \in \beta^{m_2}\mathbb{E}^*$, for some $\gamma \in \mathbb{E}^*$,

$$p_j(x) = \left(\frac{x}{\beta^{m_2}}\right)^{j-1} = \gamma^{j-1}, \tag{2.35}$$

so

$$rank_{\mathbb{E}}(\{p_1(x), p_2(x), \ldots, p_{\frac{\ell}{a}}(x)\}) = 1. \tag{2.36}$$

Therefore, $\{p_j(\alpha^*), j \in [\frac{\ell}{a}]\}$ has full rank over $\mathbb{E}$, for any evaluation point $\alpha^* \in A$. For x from the coset containing α^*, the polynomials have rank ℓ/a, and for x from the other coset, the polynomials

have rank 1. Then, the repair bandwidth in symbols over $\mathbb{B}$ can be calculated from (2.10) as

$$\begin{aligned} b &= \frac{\ell}{a}(\frac{n}{2}-1)\log_q|\mathbb{E}| + \frac{n}{2}\log_q|\mathbb{E}| \\ &= (n-1)\frac{\ell+a}{2} - \frac{\ell-a}{2} \\ &< (n-1)\frac{\ell+a}{2}. \end{aligned} \tag{2.37}$$

Thus, the proof is completed. ■

Example 2. Take the $RS(14, 11)$ code over $\mathbb{F} = GF(2^{12})$ for example. Let β be the primitive element in $\mathbb{F}$, $a = 4$, $s = \ell/a = 3$ and $A = \mathbb{E}^* \cup \beta\mathbb{E}^*$. Assume $\alpha^* \in \beta\mathbb{E}^*$, then $\{p_j(x), j \in [3]\}$ is the set $\{1, x, x^2\}$. It is easy to check that when $x \in \beta\mathbb{E}^*$ the polynomials have full rank and when $x \in \mathbb{E}^*$ the polynomials have rank 1. The total repair bandwidth is 100 bits. For the same (n, k, ℓ), the repair bandwidth of our scheme in one coset is 117 bits. For the scheme in [40], which only works for $\ell/a = 2$, we can only choose $a = 6$ and get the repair bandwidth of 114 bits for the same (n, k, ℓ).

2.3.3 Schemes in multiple cosets

In the schemes in this subsection, we extend an original code to a new code over a larger field and the evaluation points are chosen from multiple cosets in Lemma 1 to increase the code length. The construction ensures that for fixed n, the sub-packetization size is smaller than the original code. If the original code satisfies several conditions to be discussed soon, the repair bandwidth in the new code is only slightly larger than that of the original code. Particularly, if the original code is an MSR code, then we can get the new code in a much smaller sub-packetization level with a small extra repair bandwidth. Also, if the original code works for any number of helpers and multiple erasures, the new code works for any number of helpers and multiple erasures, too. We discuss multiple erasures in Section 2.4.

We first prove a lemma regarding the ranks over different base fields, and then describe the new code.

Lemma 3. Let $\mathbb{B} = GF(q), \mathbb{F}' = GF(q^{\ell'}), \mathbb{E} = GF(q^a)$, $\mathbb{F} = GF(q^{\ell}), \ell = a\ell'$. a and ℓ' are relatively prime and q can be any power of a prime number. For any set of $\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\} \subseteq \mathbb{F}' \leq \mathbb{F}$, we have

$$rank_{\mathbb{E}}(\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\}) = rank_{\mathbb{B}}(\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\}). \tag{2.38}$$

Proof: Assume $rank_{\mathbb{B}}(\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\}) = c$ and without loss of generality, $\{\gamma_1, \gamma_2, ..., \gamma_c\}$ are linearly independent over $\mathbb{B}$. Then, we can construct $\{\gamma'_{c+1}, \gamma'_{c+2}, ..., \gamma'_{\ell'}\} \subseteq \mathbb{F}'$ to make $\{\gamma_1, \gamma_2, ..., \gamma_c, \gamma'_{c+1}, \gamma'_{c+2}, ..., \gamma'_{\ell'}\}$ form a basis for $\mathbb{F}'$ over $\mathbb{B}$.

Assume we get $\mathbb{F}$ by adjoining β to $\mathbb{B}$. Then, from [63, Theorem 1.86] we know that $\{1, \beta, \beta^2, ..., \beta^{\ell'-1}\}$ is a basis for both $\mathbb{F}$ over $\mathbb{E}$, and $\mathbb{F}'$ over $\mathbb{B}$. So, any symbol $y \in \mathbb{F}$ can be presented as a linear combination of $\{1, \beta, \beta^2, ..., \beta^{\ell'-1}\}$ with some coefficients in $\mathbb{E}$. Also, we know that there is an invertible linear transformation with coefficients in $\mathbb{B}$ between $\{\gamma_1, \gamma_2, ..., \gamma_c, \gamma'_{c+1}, \gamma'_{c+2}, ..., \gamma'_{\ell'}\}$ and $\{1, \beta, \beta^2, ..., \beta^{\ell'-1}\}$, because they are a basis for $\mathbb{F}'$ over $\mathbb{B}$. Combined with the fact that $\{1, \beta, \beta^2, ..., \beta^{\ell'-1}\}$ is also a basis for $\mathbb{F}$ over $\mathbb{E}$, we can conclude that any symbol $y \in \mathbb{F}$ can be represented as

$$y = x_1\gamma_1 + x_2\gamma_2 + ... + x_c\gamma_c + x_{c+1}\gamma'_{c+1} + ... + x_\ell\gamma'_{\ell'} \tag{2.39}$$

with some coefficients $x_i \in \mathbb{E}$, which means that $\{\gamma_1, \gamma_2, ..., \gamma_c, \gamma'_{c+1}, \gamma'_{c+2}, ..., \gamma'_{\ell'}\}$ is also a basis

for $\mathbb{F}$ over $\mathbb{E}$. Then, we have that $\{\gamma_1, \gamma_2, ..., \gamma_c\}$ are linearly independent over $\mathbb{E}$,

$$\begin{aligned} & rank_{\mathbb{E}}(\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\}) \\ \geq & c \\ = & rank_{\mathbb{B}}(\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\}). \end{aligned} \tag{2.40}$$

Since $\mathbb{B} \leq \mathbb{E}$, we also have

$$rank_{\mathbb{E}}(\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\}) \leq rank_{\mathbb{B}}(\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\}). \tag{2.41}$$

The proof is completed. ■

Theorem 3. Assume there exists a $RS(n', k')$ code $\mathscr{E}'$ over $\mathbb{F}' = GF(q^{\ell'})$ with evaluation points set A'. The evaluation points are linearly independent over $\mathbb{B} = GF(q)$. The repair bandwidth is b' and the repair polynomials are $p'_j(x)$. Then, we can construct a new $RS(n, k)$ code $\mathscr{E}$ over $\mathbb{F} = GF(q^{\ell}), \ell = a\ell'$ with $n = (q^a - 1)n', k = n - n' + k'$ and repair bandwidth of $b = ab'(q^a - 1) + (q^a - 2)\ell$ symbols over $\mathbb{B} = GF(q)$ if we can find new repair polynomials $p_j(x) \in \mathbb{F}[x], j \in [\ell']$, with degrees less than $n - k$ that satisfy

$$rank_{\mathbb{E}}(\{p_1(x), p_2(x), \ldots, p_{\ell'}(x)\}) = rank_{\mathbb{B}}(\{p'_1(\alpha), p'_2(\alpha), \ldots, p'_{\ell'}(\alpha)\}) \tag{2.42}$$

for all $\alpha \in A', x \in \alpha\mathbb{E}^*$, where $\mathbb{E} = GF(q^a)$.

Proof: We first prove the case when a and ℓ' are necessarily relatively prime using Lemma 3, the case when a and ℓ' are not relatively prime are proved in Section 2.8.1. Assume the evaluation points of $\mathscr{E}'$ are $A' = \{\alpha_1, \alpha_2, \ldots, \alpha_{n'}\}$, then from Lemma 3 we know that they are also linearly independent over $\mathbb{E}$, so there does not exist $\gamma_i, \gamma_j \in \mathbb{E}^*$ that satisfy $\alpha_i\gamma_i = \alpha_j\gamma_j$, which implies that

$\{\alpha_1\mathbb{E}^*, \alpha_2\mathbb{E}^*, \ldots, \alpha_{n'}\mathbb{E}^*\}$ are distinct cosets. Then, we can extend the evaluation points to be

$$A = \{\alpha_1\mathbb{E}^*, \alpha_2\mathbb{E}^*, \ldots, \alpha_{n'}\mathbb{E}^*\}. \tag{2.43}$$

and $n = (q^a - 1)n'$. We keep the same redundancy $r = n' - k'$ for the new code so $k = n - r$.

For the new code $\mathscr{E}$, we use $p_j(x) \in \mathbb{F}[x], j \in [\ell']$ to repair the failed node $f(\alpha^*)$

$$tr_{\mathbb{F}/\mathbb{E}}(v_{\alpha^*}p_j(\alpha^*)f(\alpha^*)) = -\sum_{\alpha\in A, \alpha\neq\alpha^*} tr_{\mathbb{F}/\mathbb{E}}(v_\alpha p_j(\alpha)f(\alpha)). \tag{2.44}$$

Assume the failed node is $f(\alpha^*)$ and $\alpha^* \in \alpha_i\mathbb{E}^*$. Then, for the node $x \in \alpha_i\mathbb{E}^*$, because the original code satisfies the full rank condition, we have

$$\begin{aligned} &rank_{\mathbb{E}}(\{p_1(x), p_2(x), \ldots, p_{\ell'}(x)\}) \\ =&rank_{\mathbb{B}}(\{p'_1(\alpha_i), p'_2(\alpha_i), \ldots, p'_{\ell'}(\alpha_i)\}) = \ell', \end{aligned} \tag{2.45}$$

then we can recover the failed node with $p_j(x)$, and each helper in the coset containing the failed node transmits ℓ' symbols over $\mathbb{E}$.

For a helper in the other cosets, $x \in \alpha_\epsilon\mathbb{E}^*, \epsilon \neq i$, by (2.42),

$$\begin{aligned} &rank_{\mathbb{E}}(\{p_1(x), p_2(x), \ldots, p_{\ell'}(x)\}) \\ =&rank_{\mathbb{B}}(\{p'_1(\alpha_\epsilon), p'_2(\alpha_\epsilon), \ldots, p'_{\ell'}(\alpha_\epsilon)\}), \end{aligned} \tag{2.46}$$

then every helper in these cosets transmits $\frac{b'}{n'-1}$ symbols in $\mathbb{E}$ on average.

The repair bandwidth of the new code can be calculated from the repair bandwidth condition (2.10)

as

$$b = \frac{b'}{n'-1} \cdot (n'-1)|\mathbb{E}^*| \cdot a + (|\mathbb{E}^*|-1)\ell' \cdot a$$
$$= ab'(q^a-1) + (q^a-2)\ell \tag{2.47}$$

which completes the proof. ■

Note that the calculation in (2.47) and (2.37) are similar in the sense that a helper in the coset containing the failure naively transmits the entire stored information, and the other helpers use the bandwidth that is the same as the original code.

As a special case of Theorem 3, when $b' = \frac{\ell'}{r}(n'-1)$ matching the MSR bound (2.1), we get

$$b = \frac{\ell}{r}(n-1) + \frac{\ell}{r}(r-1)(q^a-2), \tag{2.48}$$

where the second term is the extra bandwidth compared to the MSR bound.

Next, we apply Theorem 3 to the near-MSR code [103] and the MSR code [94]. The first realization of the scheme in multiple cosets is inspired by [103].

Theorem 4. There exists an $RS(n,k)$ code over $\mathbb{F} = GF(q^\ell)$ of which $n = (q^a-1)\log_r \frac{\ell}{a}$ and $a|\ell$, such that the repair bandwidth satisfies $b < \frac{\ell}{n-k}[n+1+(n-k-1)(q^a-2)]$, measured in symbols over $\mathbb{B} = GF(q)$ for some prime number q.

Proof: We first prove the case when a and ℓ' are relatively prime using Lemma 3, the case when a and ℓ' are not necessarily relatively prime are proved in Section 2.8.1. We use the code in [103] as the original code. The original code is defined in $\mathbb{F}' = GF(q^{\ell'})$ and $\ell' = r^{n'}$. The evaluation points are $A' = \{\beta, \beta^r, \beta^{r^2}, \ldots, \beta^{r^{n'-1}}\}$ where β is a primitive element of $\mathbb{F}'$.

In the original code, for $c = 0, 1, 2, \ldots, \ell'-1$, we write its r-ary expansion as $c = (c_{n'}c_{n'-1}\ldots c_1)$, where $0 \le c_i \le r-1$ is the i-th digit from the right. Assuming the failed node is $f(\beta^{r^{i-1}})$, the

repair polynomials are chosen to be

$$p'_j(x) = \beta^c x^s, c_i = 0, s = 0, 1, 2, \ldots, r-1, x \in \mathbb{F}'. \tag{2.49}$$

Here c varies from 0 to $\ell' - 1$ given that $c_i = 0$, and s varies from 0 to $r - 1$. So, we have ℓ' polynomials in total. The subscript j is indexed by c and s, and by a small abuse of the notation, we write $j \in [\ell']$.

In the new code, let us define $\mathbb{E} = GF(q^a)$ of which a and ℓ' are relatively prime. Adjoining β to $\mathbb{E}$, we get $\mathbb{F} = GF(q^\ell), \ell = a\ell'$. The new evaluation points are $A = \{\beta\mathbb{E}^*, \beta^r\mathbb{E}^*, \ldots, \beta^{r^{n'-1}}\mathbb{E}^*\}$. Since A' is part of the polynomial basis for $\mathbb{F}'$ over $\mathbb{B}$, we know that $\{\beta, \beta^r, \ldots, \beta^{r^{n'-1}}\}$ are linearly independent over $\mathbb{B}$. Hence, we can apply Lemma 3 and the cosets are distinct, resulting in $n = |A| = (q^a - 1)\log_r \frac{\ell}{a}$.

In our new code, let us assume the failed node is $f(\alpha^*)$ and $\alpha^* \in \beta^{r^{i-1}}C$, and we choose the polynomial $p_j(x)$ with the same form as $p'_j(x)$,

$$p_j(x) = \beta^c x^s, c_i = 0, s = 0, 1, 2, \ldots, r-1, x \in \mathbb{F}. \tag{2.50}$$

For nodes corresponding to $x = \beta^{r^t}\gamma \in \beta^{r^t}\mathbb{E}^*$, for some $\gamma \in \mathbb{E}^*$, we know that

$$p_j(x) = \beta^c x^s = \beta^c(\gamma\beta^{r^t})^s = \gamma^s p'_j(\beta^{r^t}). \tag{2.51}$$

Since $p'_j(\beta^{r^t}) \in \mathbb{F}'$, from Lemma 3, we have

$$\begin{aligned} &rank_{\mathbb{E}}(\{\gamma^s p'_1(\beta^{r^t}), \gamma^s p'_2(\beta^{r^t}), \ldots, \gamma^s p'_{\ell'}(\beta^{r^t})\}) \\ =&rank_{\mathbb{E}}(\{p'_1(\beta^{r^t}), p'_2(\beta^{r^t}), \ldots, p'_{\ell'}(\beta^{r^t})\}) \\ =&rank_{\mathbb{B}}(\{p'_1(\beta^{r^t}), p'_2(\beta^{r^t}), \ldots, p'_{\ell'}(\beta^{r^t})\}), \end{aligned} \tag{2.52}$$

which satisfies (2.42). Since the repair bandwidth of the original code is $b' < (n'+1)\frac{\ell'}{r}$, from (2.47) we can calculate the repair bandwidth as

$$\begin{aligned} b &= ab'(q^a-1)+(q^a-2)\ell \\ &< \frac{\ell}{r}[n+1+(r-1)(q^a-2)], \end{aligned} \tag{2.53}$$

where the second term is the extra bandwidth compared to the original code. ■

Example 3. We take an $RS(4,2)$ code in $GF(2^{16})$ as the original code and extend it with $a = 3, |\mathbb{E}^*| = 7$ to an $RS(28,26)$ code in $GF(2^{48})$ with normalized repair bandwidth of $\frac{b}{(n-1)\ell} < 0.65$. The $RS(28,26)$ code in [103] achieves the normalized repair bandwidth of $\frac{b}{(n-1)\ell} < 0.54$, while it requires $\ell = 2.7 \times 10^8$. Our scheme has a much smaller ℓ compared to the scheme in [103] while the repair bandwidth is a bit larger.

In the above theorem, we extend [103] to a linearly larger sub-packetization and an exponentially larger code length, which means that for the same code length, we can have a much smaller sub-packetization level.

Next, we show our second realization of the scheme in multiple cosets, which is inspired by [94]. Different from the previous constructions, this one allows any number of helpers, $k \le d \le n-1$. The sub-packetization size in the original code of [94] satisfies $\ell' \approx (n')^{n'}$ when n' grows to infinity, thus in our new code it satisfies $\ell \approx a(n')^{n'}$ for some integer a.

Theorem 5. Let q be a prime number. There exists an $RS(n,k)$ code over $\mathbb{F} = GF(q^\ell)$ of which $\ell = asq_1q_2...q_{\frac{n}{q^a-1}}$, where q_i is the i-th prime number that satisfies $s|(q_i-1), s = d-k+1$ and a is some integer. d is the number of helpers, $k \le d \le (n-1)$. The average repair bandwidth is $b = \frac{d\ell}{(n-1)(d-k+1)}[n-1+(d-k)(q^a-2)]$ measured in symbols over $\mathbb{B} = GF(q)$.

Proof: We first prove the case when a and ℓ' are relatively prime using Lemma 3, the case when a and ℓ' are not necessarily relatively prime are proved in Section 2.8.1. We use the code in [94]

as the original code, where the number of helpers is d'. We set $n-k=n'-k'$ and calculate the repair bandwidth for d helpers from the original code when $d'=d-k+k'$. Let us define $\mathbb{F}_q(\alpha)$ to be the field obtained by adjoining α to the base field $\mathbb{B}$. Similarly, we define $\mathbb{F}_q(\alpha_1,\alpha_2,\ldots,\alpha_n)$ for adjoining multiple elements. Let α_i be an element of order q_i over $\mathbb{B}$. The code is defined in the field $\mathbb{F}'=GF(q^{\ell'})=GF(q^{sq_1q_2\cdots,q_{n'}})$, which is the degree-s extension of $\mathbb{F}_q(\alpha_1,\alpha_2,\ldots,\alpha_{n'})$. The evaluation points are $A'=\{\alpha_1,\alpha_2,\ldots,\alpha_{n'}\}$.

Assuming the failed node is $f(\alpha_i)$ and the helpers are chosen from the set R', $|R'|=d'$, the base field for repair is $\mathbb{F'}_i$, defined as $\mathbb{F'}_i \triangleq \mathbb{F}_q(\alpha_j, j\in[n'], j\neq i)$. The repair polynomials are $\{\eta_t p'_j(\alpha_i), t\in[q_i], j\in[s]\}$, where

$$p'_j(x)=x^{j-1}g'(x), j\in[s], x\in\mathbb{F}', \tag{2.54}$$

$$g'(x)=\prod_{\alpha\in A/(R'\cup\{\alpha_i\})}(x-\alpha), x\in\mathbb{F}'. \tag{2.55}$$

and $\eta_t\in\mathbb{F}', t\in[q_i]$, are constructed in [94] such that $\{\eta_t p'_j(\alpha_i), t\in[q_i], j\in[s]\}$ forms the basis for $\mathbb{F}'$ over $\mathbb{F'}_i$. The repair is done using

$$tr_{\mathbb{F}'/\mathbb{F'}_i}(\upsilon_{\alpha_i}\eta_t p'_j(\alpha_i)f'(\alpha_i)) = -\sum_{\epsilon=1,\epsilon\neq i}^{n'} tr_{\mathbb{F}'/\mathbb{F'}_i}(\upsilon_\epsilon\eta_t p'_j(\alpha_\epsilon)f'(\alpha_\epsilon)). \tag{2.56}$$

For $x\notin R'\cup\{\alpha_i\}$, $p'_j(x)=0$, so no information is transmitted. The original code reaches the MSR repair bandwidth

$$\begin{aligned} b' &= \sum_{\epsilon\in R'} rank_{\mathbb{F'}_i}(\{\eta_t p'_j(\alpha_\epsilon): t\in[q_i], j\in[s]\}) \\ &= \frac{d'\ell'}{d'-k'+1}. \end{aligned} \tag{2.57}$$

In our new code, we define $\mathbb{E}=GF(q^a)=\mathbb{F}_q(\alpha_{n+1})$ where a and ℓ' are relatively prime, and

α_{n+1} is an element of order a over $\mathbb{B}$. Adjoining the primitive element of $\mathbb{F}'$ to $\mathbb{E}$, we get $\mathbb{F} = GF(q^\ell), \ell = a\ell'$. The new code is defined in $\mathbb{F}$. We extend the evaluation points to be $A = \{\alpha_1\mathbb{E}^*, \alpha_2\mathbb{E}^*, \ldots, \alpha_{n'}\mathbb{E}^*\}$. Since $\{\alpha_1, \alpha_2, ..., \alpha_{n'}\}$ are linearly independent over $\mathbb{B}$, we can apply Lemma 3 and the cosets are distinct. So, $n = |A| = (q^a - 1)n'$.

Assuming the failed node is $f(\alpha^*)$ and $\alpha^* \in \alpha_i\mathbb{E}^*$ and the helpers are chosen from the set R, $|R| = d$, the base field for repair is $\mathbb{F}_i$, which is defined by $\mathbb{F}_i \triangleq \mathbb{F}_q(\alpha_j, j \in [n+1], j \neq i)$, for $i \in [n]$. We define the repair polynomials $\{\eta_t p_j(x), t \in [q_i], j \in [s]\}$, where

$$p_j(x) = x^{j-1}g(x), j \in [s], x \in \mathbb{F}, \tag{2.58}$$

$$g(x) = \prod_{\alpha \in A/(R \cup \{\alpha^*\})} (x - \alpha), x \in \mathbb{F}, \tag{2.59}$$

and η_t is the same as that in the original code. Then, we repair the failed node by

$$tr_{\mathbb{F}/\mathbb{F}_i}(\upsilon_{\alpha^*}\eta_t p_j(\alpha^*)f(\alpha^*)) = -\sum_{\alpha \in A, \alpha \neq \alpha^*} tr_{\mathbb{F}/\mathbb{F}_i}(\upsilon_\alpha \eta_t p_j(\alpha)f(\alpha)). \tag{2.60}$$

For $x \in \alpha\mathbb{E}^*, \alpha \in A'$, we have

$$p_j(x) = \gamma^{j-1}\alpha^{j-1}g(x), j \in [s], \tag{2.61}$$

for some $\gamma \in \mathbb{E}^*$. If $x \notin R \cup \{\alpha^*\}$, since $g(x) = 0$, no information is transmitted from node x. Next, we consider all other nodes.

For $x = \alpha\gamma, \alpha \in A'$, since $g(x)$ is a constant independent of j, $\gamma \in \mathbb{E} \subseteq \mathbb{F}_i$ and $\eta_t, \alpha_i \in \mathbb{F}'$, from

Lemma 3 we have

$$\begin{aligned}
&rank_{\mathbb{F}_i}(\{\eta_t p_1(x), \eta_t p_2(x), \ldots, \eta_t p_s(x) : t \in [q_i]\}) \\
=&rank_{\mathbb{F}_i}(\{\eta_t, \eta_t \gamma\alpha, \ldots, \eta_t \gamma^{s-1}\alpha^{s-1} : t \in [q_i]\}) \\
=&rank_{\mathbb{F}_i}(\{\eta_t, \eta_t \alpha, \ldots, \eta_t \alpha^{s-1} : t \in [q_i]\}) \\
=&rank_{\mathbb{F}'_i}(\{\eta_t, \eta_t \alpha, \ldots, \eta_t \alpha^{s-1} : t \in [q_i]\}) \\
=&rank_{\mathbb{F}'_i}(\{\eta_t p'_1(\alpha), \eta_t p'_2(\alpha), \ldots, \eta_t p'_s(\alpha) : t \in [q_i]\}),
\end{aligned} \tag{2.62}$$

which satisfies (2.42).

When $k \leq d < n-1$, assuming the helpers are randomly chosen from all the remaining nodes, the average repair bandwidth for different choices of the helpers can be calculated as

$$b = d\left[\frac{b'a}{d'} \cdot \frac{n-1-(q^a-2)}{n-1} + \ell' a \cdot \frac{q^a-2}{n-1}\right] \tag{2.63}$$

$$= \frac{d\ell}{d-k+1} + \frac{d}{n-1}\frac{\ell}{d-k+1}(d-k)(q^a-2). \tag{2.64}$$

Here in (2.63) the second term corresponds to the helpers in the failed node coset, the first term corresponds to the helpers in the other cosets, and in (2.64) we used $d' - k' = d - k$. ■

In the case of $d = n-1$, the repair bandwidth of the code in Theorem 5 can be directly calculated from (2.47) as

$$\begin{aligned}
b &= ab'(q^a-1) + (q^a-2)\ell \\
&= \frac{\ell}{r}(n-1) + \frac{\ell}{r}(r-1)(q^a-2)].
\end{aligned} \tag{2.65}$$

In (2.64) and (2.65), the second term is the extra repair bandwidth compared to the original code.

In Theorems 4 and 5, we constructed our schemes by extending previous schemes. However, it should be noted that since we only used the properties of the polynomials $p'_j(x)$, we have no

restrictions on the dimensions k' of the original codes. So, in some special cases, even if k' is negative and the original codes do not exist, our theorems still hold. Thus, we can provide more feasible points of (n, k) using our schemes. This is illustrated in the example below.

Example 4. Let us take the $RS(12, 8)$ code as an example. We set $q = 2, s = 4, q_1 = 5, q_2 = 9, q_3 = 13$ and $a = 7$. Then, $\ell' = 2340$ and $\ell = 16380$. Assuming the failed node is $f(\alpha^*)$ and $\alpha^* \in \alpha_1 C$, then we repair it in $\mathbb{F}_1$ and set the polynomials in (2.58). We can easily check that when $x \in \alpha_1 C$, $rank_{\mathbb{F}_1}(\{\eta_t p_1(x), \eta_t p_2(x), \ldots, \eta_t p_s(x) : t \in [5]\}) = 20$ and when x in other cosets, $rank_{\mathbb{F}_1}(\{\eta_t p_1(x), \eta_t p_2(x), \ldots, \eta_t p_s(x) : t \in [5]\}) = 5$. Therefore, we transmit 100 symbols in $\mathbb{F}_1$, which can be normalized to $\frac{b}{(n-1)\ell} = 0.4545$. Compared with the scheme in [94], which need $\ell = 2.4 \times 10^{19}$ and $\frac{b}{(n-1)\ell} = 0.25$, we provide a tradeoff between ℓ and b.

It should be noted that in this example, the $RS(12, 8)$ code needs to be extended from an $RS(3, -1)$ code, which does not exist. However, since we only used the properties of the polynomials $p'_j(x)$ and $p_j(x)$, the new $RS(12, 8)$ code still works.

2.3.4 Numerical evaluations and discussions

In this subsection, we compare the existing and the proposed schemes. Table 2.1 shows the repair bandwidth and the code length of each scheme. For the comparison, we first show in Figures 2.1 and 2.2 the performance of each scheme when the sub-packetization changes, given $(n, k) = (12, 10)$ and $(12, 8)$, respectively. We only consider $n - 1$ helpers. Two single points $(\log_2(\ell) = 53.5, \frac{b}{(n-1)\ell} = 0.50)$ in $RS(12, 10)$ codes and $(\log_2(\ell) = 64.4, \frac{b}{(n-1)\ell} = 0.25)$ in $RS(12, 8)$ codes are not shown in the figures, they can be achieved by both our second realization in multiple cosets and [94]. We make the following observations.

1. For a fixed (n, k), we compare the normalized repair bandwidth $b/[(n-1)\ell]$ in different sub-packetization sizes. In our schemes in multiple cosets, we have a parameter a to adjust the sub-packetization size. From Theorems 4 and 5 we know that for the two schemes,

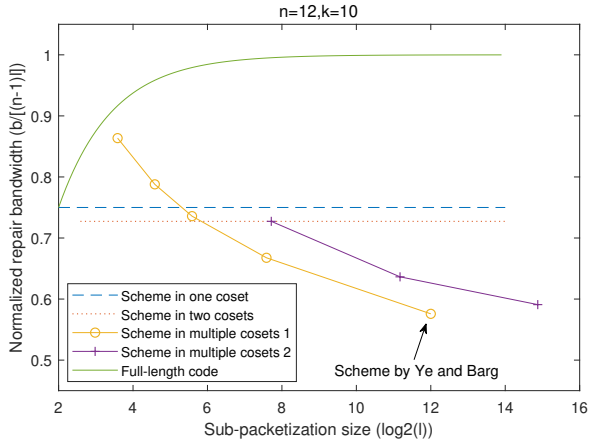

Figure 2.1: Comparison of 3 schemes, $q = 2, n = 12, k = 10, r = 2$.

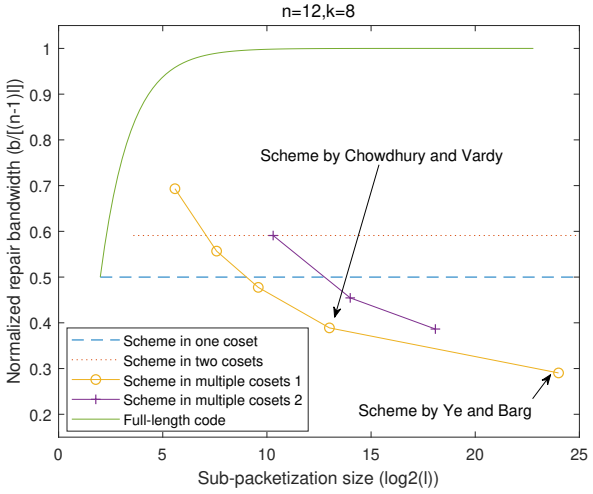

Figure 2.2: Comparison of 3 schemes, $q = 2, n = 12, k = 8, r = 4$.

$\ell = a \cdot r^{\frac{n}{q^a-1}}$ and $\ell \approx a \cdot (\frac{n}{q^a-1})^{(\frac{n}{q^a-1})}$, respectively, which means that increasing a will decrease the sub-packetization ℓ. In our schemes in one coset and two cosets, the parameter a is determined by code length n, and will not be changed by increasing ℓ, neither will the normalized repair bandwidth. When $q = 2$, $a = 1$, the two schemes in multiple cosets coincides with [103] and [94], respectively.

2. The scheme in [16] also achieves one tradeoff point in Figure 2.2, which can be viewed as a special case of our scheme in multiple coset 1.
3. For fixed n, k, our schemes are better than the full-length code in [40] and [18] for all ℓ, except when $\ell = 4$, for which our scheme in one coset is identical to the full-length code.
4. While the repair bandwidth of the full-length code grows with ℓ, our schemes in one coset and two cosets have a constant normalized bandwidth, and our schemes in multiple cosets have a decreasing normalized bandwidth with ℓ.
5. For small ℓ: the schemes in one coset and two cosets are better than those in multiple cosets; when $n = 12, k = 10, 4 \leq \ell \leq 48$, the scheme in two cosets provides the lowest bandwidth; when $n = 12, k = 8, 4 \leq \ell \leq 768$, one can show that the scheme in one coset has the smallest bandwidth.
6. For large ℓ: the first realization in multiple cosets has better performance than the second realization in multiple cosets, but our second realization works for any number of helpers.

2.4 Reed-Solomon Repair Schemes for Multiple Erasures

In this section, we first present two definitions of the repair schemes for multiple erasures in a MDS code: linear repair scheme definition and dual code repair definition. We prove the equivalence of the two definitions. Then, we present two schemes for repairing multiple erasures in Reed-Solomon codes, where the evaluation points are in one coset and multiple cosets, respectively.

2.4.1 Definitions of the multiple-erasure repair

Let us assume a scalar MDS code $\mathscr{E}$ over $\mathbb{F} = GF(q^\ell)$ has dimension k and code length n. Let a codeword be $(C_1, C_2, ...C_n)$. Without loss of generality, we assume $\{C_1, C_2, ..., C_e\}$ are failed, $e \leq n - k$, and we repair them in the base field $\mathbb{B} = GF(q)$, where q can be any power of a prime number. We also assume that we use all the remaining $d = n - e$ nodes as helpers. The following definitions are inspired by [5] for single erasure.

Definition 1. A linear exact repair scheme for multiple erasures consists of the following.

1. A set of queries $Q_t \subseteq \mathbb{F}$ for each helper $C_t, e + 1 \leq t \leq n$. The helper C_t replies with $\{\gamma C_t, \gamma \in Q_t\}$.
2. For each failed node $C_i, i \in [e]$, a linear repair scheme that computes

$$C_i = \sum_{m=1}^{\ell} \lambda_{im} \mu_{im}, \tag{2.66}$$

where $\{\mu_{i1}, \mu_{i2}, ..., \mu_{i\ell}\}$ is a basis for $\mathbb{F}$ over $\mathbb{B}$ and coefficients $\lambda_{im} \in \mathbb{B}$ are $\mathbb{B}$-linear combinations of the replies

$$\lambda_{im} = \sum_{t=e+1}^{n} \sum_{\gamma \in Q_t} \beta_{im\gamma t} \cdot tr_{\mathbb{F}/\mathbb{B}}(\gamma C_t), \tag{2.67}$$

with the coefficients $\beta_{im\gamma t} \in \mathbb{B}$. The repair bandwidth is

$$b = \sum_{t=e+1}^{\ell} rank_{\mathbb{B}}(Q_t). \tag{2.68}$$

In the following definition, we consider $e\ell$ dual codewords of $\mathscr{E}$, and index them by $i \in [e], j \in [\ell]$, denoted as $(C'_{ij1}, C'_{ij2}, \ldots, C'_{ijn})$. Since they are dual codwords, we know that $\sum_{t=1}^{n} C_t C'_{ijt} = 0$.

Definition 2. A dual code scheme uses a set of dual codewords $\{(C'_{ij1}, C'_{ij2}, \ldots, C'_{ijn}) : i \in [e], j \in [\ell]\}$ that satisfies:

1. The **full rank condition**: Vectors

$$V_{ij} = (C'_{ij1}, C'_{ij2}, ..., C'_{ije}), i \in [e], j \in [\ell], \tag{2.69}$$

   are linearly independent over $\mathbb{B}$.

2. The **repair bandwidth condition**:

$$b = \sum_{t=e+1}^{n} rank_{\mathbb{B}}(\{C'_{ijt} : i \in [e], j \in [\ell]\}). \tag{2.70}$$

We repair nodes $[e]$ from the linearly independent equations

$$\sum_{v=1}^{e} tr_{\mathbb{F}/\mathbb{B}}(C'_{ijv}C_v) = -\sum_{t=e+1}^{n} tr_{\mathbb{F}/\mathbb{B}}(C'_{ijt}C_t), i \in [e], j \in [\ell]. \tag{2.71}$$

Here we use the same condition names as the single erasure case, but in this section, they are defined for multiple erasures.

Theorem 6. Definitions 1 and 2 are equivalent.

The equivalence of Definitions 1 and 2 follows similarly as arguments in [40], except that we need to first solve e failed nodes simultaneously and then find out the form of each individual failure (2.66). The detailed proof of Theorem 6 is shown in Section 2.8.2, part of which uses Lemma 4 in Section IV-B.

Remark 2. In this chapter, we focus on repairing RS code and apply Theorem 6 to RS code. From [65, Thm. 4 in Ch. 10] we know that with the polynomial $p_{ij}(x) \in \mathbb{F}[x]$ for which the degrees are smaller than $n-k$, $(\upsilon_1 p_{ij}(\alpha_1), \upsilon_2 p_{ij}(\alpha_2), \ldots, \upsilon_n p_{ij}(\alpha_n))$ is the dual codeword of $RS(n,k)$, where $\upsilon_i, i \in [n]$ are non-zero constants determined by the evaluation points set A. So, in RS code, Definition 2 reduces to finding polynomials $p_{ij}(x)$ with degrees smaller than $n-k$. In what follows we use $p_{ij}(\alpha_t)$ to replace the dual codeword symbol C'_{ijt} in Definition 2 for RS code. One

can easily show that the constants $v_i, i \in [n]$ do not affect the ranks in the full rank condition and the repair bandwidth condition.

2.4.2 Multiple-erasure repair in one coset

There are several studies about the multiple erasures for full-length RS codes [17] and [67]. Inspired by these works, we propose our scheme for multiple erasures in one coset.

From Theorem 6, we know that finding the repair scheme for multiple erasures in RS code is equivalent to finding dual codewords (or polynomials) that satisfy the full rank condition and repair bandwidth condition. Given a basis $\{\xi_1, \xi_2, ..., \xi_\ell\}$ for $\mathbb{F}$ over $\mathbb{B}$, we define some matrices as below. They are used to help us check the two rank conditions according to Lemmas 4 and 5, whose proofs are shown in Appendices C and D, respectively. Let the evaluation points of an RS code over $\mathbb{F}$ be $A = \{\alpha_1, \ldots, \alpha_n\}$. Let $p_{ij}(x), i \in [e], j \in [\ell]$, be polynomials over $\mathbb{F}$, and $\mathbb{B}$ a subfield of $\mathbb{F}$. Define

$$S_{it} = \begin{bmatrix} tr_{\mathbb{F}/\mathbb{B}}(\xi_1 p_{i1}(\alpha_t)) & \cdots & tr_{\mathbb{F}/\mathbb{B}}(\xi_\ell p_{i1}(\alpha_t)) \\ tr_{\mathbb{F}/\mathbb{B}}(\xi_1 p_{i2}(\alpha_t)) & \cdots & tr_{\mathbb{F}/\mathbb{B}}(\xi_\ell p_{i2}(\alpha_t)) \\ \vdots & \ddots & \vdots \\ tr_{\mathbb{F}/\mathbb{B}}(\xi_1 p_{i\ell}(\alpha_t)) & \cdots & tr_{\mathbb{F}/\mathbb{B}}(\xi_\ell p_{i\ell}(\alpha_t)) \end{bmatrix}, \tag{2.72}$$

$$S \triangleq \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1e} \\ S_{21} & S_{22} & \cdots & S_{2e} \\ \vdots & \vdots & \ddots & \vdots \\ S_{e1} & S_{e2} & \cdots & S_{ee} \end{bmatrix}. \tag{2.73}$$

Lemma 4. The following two statements are equivalent:

1. Vectors $V_{ij} = (p_{ij}(\alpha_1), p_{ij}(\alpha_2), \ldots, p_{ij}(\alpha_e)), i \in [e], j \in [\ell]$ are linearly independent over $\mathbb{B}$.
2. Matrix S in (2.73) has full rank.

Lemma 5. For $t \in [n]$, consider S_{it} in (2.72),

$$rank(\begin{bmatrix} S_{1t} \\ S_{2t} \\ \vdots \\ S_{et} \end{bmatrix}) = rank_{\mathbb{B}}(\{p_{ij}(\alpha_t) : i \in [e], j \in [\ell]\}). \tag{2.74}$$

Theorem 7. Let q be a prime number. There exists an $RS(n,k)$ code over $\mathbb{F} = GF(q^\ell)$ of which $n < q^a, q^s \leq r$ and $a|\ell$, such that the repair bandwidth for e erasures is $b \leq \frac{e\ell}{a}(n-e)(a-s)$ measured in symbols over $\mathbb{B}$, for e satisfying $a \geq \frac{e(e-1)}{2}(a-s)^2$.

Proof: We define the code over the field $\mathbb{F} = GF(q^\ell)$ extended by $\mathbb{E} = GF(q^a)$, where β is the primitive element of $\mathbb{F}$. The evaluation points are chosen to be $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq \mathbb{E}^*$, which is one of the cosets in Lemma 1. Without loss of generality, we assume the e failed nodes are $\{\alpha_1, \alpha_2, \ldots, \alpha_e\}$. The base field is $\mathbb{B} = GF(q)$.

Construction III: We first consider the special case when $s = a - 1$. In this case, inspired by [67, Proposition 1], we choose the polynomials

$$p_{ij}(x) = \frac{\delta_i tr_{\mathbb{E}/\mathbb{B}}(\frac{\mu_j}{\delta_i}(x - \alpha_i))}{x - \alpha_i}, i \in [e], j \in [a], \tag{2.75}$$

where $\{\mu_1, \mu_2, \ldots, \mu_a\}$ is the basis for $\mathbb{E}$ over $\mathbb{B}$, and $\delta_i \in \mathbb{E}, i \in [e]$, are coefficients to be determined. From [67, Theorem 3], we know that for $a > \frac{e(e-1)}{2}$, there exists $\delta_i, i \in [e]$ such that $p_{ij}(x)$ satisfy the full rank condition: the vectors $V_{ij} = (p_{ij}(\alpha_1), p_{ij}(\alpha_2), \ldots, p_{ij}(\alpha_e)), i \in [e], j \in [a]$ are

linearly independent over $\mathbb{B}$ and the repair bandwidth condition:

$$\begin{aligned}&\sum_{t=e+1}^{n} rank_{\mathbb{B}}(\{p_{ij}(\alpha_t) : i \in [e], j \in [a]\})\\ =&(n-e)e - \frac{e(e-1)(q-1)}{2}.\end{aligned} \tag{2.76}$$

Then, let $\{\eta_1, \eta_2, \ldots, \eta_{\ell/a}\}$ be a set of basis for $\mathbb{F}$ over $\mathbb{E}$; we have the $e\ell$ polynomials as $\{\eta_w p_{ij}(x) : w \in [\ell/a], i \in [e], j \in [a]\}$. Since $\{\eta_1, \eta_2, \ldots, \eta_{\ell/a}\}$ are linearly independent over $\mathbb{E}$ and for any $b_{ijw} \in \mathbb{B}, b_{ijw} p_{ij}(x) \in \mathbb{E}$, we have

$$\sum_{i,j,w} b_{ijw}\eta_w V_{ij} = 0 \iff \sum_{i,j} b_{ijw} V_{ij} = 0, \forall w \in [\frac{\ell}{a}]. \tag{2.77}$$

Also, we know that there does not exist nonzero $b_{ijw} \in \mathbb{B}$ that satisfies $\sum_{i,j} b_{ijw} V_{ij} = 0$, so we have that vectors $\{\eta_w V_{ij},\ w \in [\ell/a], i \in [e], j \in [a]\}$ are also linearly independent over $\mathbb{B}$. So, from Definition 2, we know that we can recover the failed nodes and the repair bandwidth is

$$\begin{aligned}b =&rank_{\mathbb{B}}(\{\eta_1 p_{ij}(x), \ldots, \eta_{\ell/a} p_{ij}(x) : i \in [e], j \in [a]\})\\ =&\frac{\ell}{a} rank_{\mathbb{B}}(\{p_{ij}(x), i \in [e], j \in [a]\})\\ =&\frac{\ell}{a}\left[(n-e)e - \frac{e(e-1)(q-1)}{2}\right].\end{aligned} \tag{2.78}$$

Construction IV: For $s \leq a-1$, consider the polynomials

$$p_{ij}(x) = \delta_i^{q^s-1}\mu_j \prod_{\varepsilon=1}^{q^s-1}\left(x - \left(\alpha_i - w_\varepsilon^{-1}\frac{\mu_j}{\delta_i}\right)\right), j \in [a], \tag{2.79}$$

where $\{\mu_1, \mu_2, \ldots, \mu_a\}$ is the basis for $\mathbb{E}$ over $\mathbb{B}$, $W = \{w_0 = 0, w_1, w_2, \ldots, w_{q^s-1}\}$ is an s-dimensional subspace in $\mathbb{E}$, $s < a, q^s \leq r$, and $\delta_i \in \mathbb{E}, i \in [e]$, are coefficients to be determined.

When $x = \alpha_i$, we have

$$p_{ij}(\alpha_i) = \mu_j^{q^s} \prod_{\varepsilon=1}^{q^s-1} w_\varepsilon^{-1}. \tag{2.80}$$

Since $\prod_{\varepsilon=1}^{q^s-1} w_\varepsilon^{-1}$ is a constant, from Lemma 2 we have

$$rank_{\mathbb{B}}(\{p_{i1}(\alpha_i), p_{i2}(\alpha_i), \ldots, p_{ia}(\alpha_i)\}) = a. \tag{2.81}$$

For $x \neq \alpha_i$, set $x' = \alpha_i - x$, we have

$$\begin{aligned} p_{ij}(x) &= \delta_i^{q^s-1}\mu_j \prod_{\varepsilon=1}^{q^s-1}\left(w_\varepsilon^{-1}\frac{\mu_j}{\delta_i} - x'\right) \\ &= \delta_i^{q^s-1}\mu_j \prod_{\varepsilon=1}^{q^s-1}(w_\varepsilon^{-1}x') \prod_{\varepsilon=1}^{q^s-1}\left(\frac{\mu_j}{\delta_i x'} - w_\varepsilon\right) \\ &= (\delta_i x')^{q^s} \prod_{\varepsilon=1}^{q^s-1}(w_\varepsilon^{-1}) \prod_{\varepsilon=0}^{q^s-1}\left(\frac{\mu_j}{\delta_i x'} - w_\varepsilon\right). \end{aligned} \tag{2.82}$$

By [35, p. 4], $g(y) = \prod_{\varepsilon=0}^{q^s-1}(y - w_\varepsilon)$ is a linear mapping from $\mathbb{E}$ to itself with dimension $a - s$ over $\mathbb{B}$. Since $(\delta_i x')^{q^s} \prod_{\varepsilon=1}^{q^s-1}(w_\varepsilon^{-1})$ is a constant independent of j, we have

$$rank_{\mathbb{B}}(\{p_{i1}(x), p_{i2}(x), \ldots, p_{ia}(x)\}) \leq a - s, \tag{2.83}$$

which means that $p_{ij}(x)$ can be written as

$$p_{ij}(x) = \delta_i^{q^s} \sum_{v=1}^{a-s} \rho_{jv}\lambda_v, \tag{2.84}$$

where $\{\lambda_1, \lambda_2, ..., \lambda_{a-s}\}$ are linearly independent over $\mathbb{B}$, $\rho_{jv} \in \mathbb{B}$, and they are determined by δ_i, μ_j and $x - \alpha_i$.

From Lemma 4, we know that if the matrix S in (2.73) has full rank, then we can recover the e erasures. It is difficult to directly discuss the rank of the matrix, but assume that the polynomials above satisfy the following two conditions:

1. $S_{ii}, i \in [e]$ are identity matrices.
2. For any fixed $i \in [e]$,

$$S_{it} \cdot S_{ty} = \mathbf{0}_{\ell \times \ell}, i > t, y > t. \tag{2.85}$$

Then, it is easy to see that through Gaussian elimination, we can transform the matrix S^T to an upper triangular block matrix, which has identity matrices in the diagonal. Hence, S has full rank.

Here, we choose $\{\xi_1, \xi_2, ..., \xi_\ell\}$ to be the dual basis of $\{\mu_1^{q^s} \prod_{\varepsilon=1}^{q^s-1} w_\varepsilon^{-1}, \mu_2^{q^s} \prod_{\varepsilon=1}^{q^s-1} w_\varepsilon^{-1}, ..., \mu_\ell^{q^s} \prod_{\varepsilon=1}^{q^s-1} w_\varepsilon^{-1}\}$, so

$$tr_{\mathbb{F}/\mathbb{B}}(\xi_m p_{ij}(\alpha_i)) = \begin{cases} 0, m \neq j, \\ 1, m = j. \end{cases} \tag{2.86}$$

Therefore, $S_{ii}, i \in [e]$ are identity matrices. We set $\delta_1 = 1$, and recursively choose δ_i after choosing $\{\delta_1, \delta_2, ..., \delta_{i-1}\}$ to satisfy (2.85). Define $\delta_i' = \delta_i^{q^s}$, and c_{mp} to be the (m, p)-th element in S_{ty} for $m, p \in [a]$. (2.85) can be written as

$$\begin{aligned} &\sum_{m=1}^{a} c_{mp} tr_{\mathbb{F}/\mathbb{B}}(\xi_m p_{ij}(\alpha_t)) \\ =&\sum_{m=1}^{a} c_{mp} \sum_{v=1}^{a-s} b_{jv} tr_{\mathbb{F}/\mathbb{B}}(\xi_m \delta_i' \lambda_v) \\ =&0, \forall j \in [a], \end{aligned} \tag{2.87}$$

where $\lambda_v, v \in [a-s]$, are determined by δ_i, μ_j and $\alpha_t - \alpha_i$. Equation (2.87) is satisfied if

$$\sum_{m=1}^{a} c_{mp} tr_{\mathbb{F}/\mathbb{B}}(\xi_m \delta_i' \lambda_v) = 0, v \in [a-s], p \in [a]. \tag{2.88}$$

As a special case of Lemma 5, we have

$$rank(S_{ty}) = rank_{\mathbb{B}}(\{p_{tj}(\alpha_y), j \in [\ell]\}). \tag{2.89}$$

Then, from (2.83) we know that the rank of S_{ty} is at most $a-s$, which means in (2.88) we only need to consider p corresponding to the independent $a-s$ columns of S_{ty}. So, (2.88) is equivalent to $(a-s)^2$ linear requirements. For $\delta_i' \in \mathbb{E}$, we can view it as a unknowns over $\mathbb{B}$, and we have

$$\frac{(2e-i)(i-1)}{2}(a-s)^2 \leq \frac{e(e-1)}{2}(a-s)^2 \tag{2.90}$$

linear requirements over $\mathbb{B}$ according to (2.85). Also knowing δ_i', we can solve $\delta_i = {\delta_i}^{q^\ell} = {\delta'}_i^{q^{\ell-s}}$. Therefore, we can find appropriate $\{\delta_1, \delta_2, \ldots, \delta_e\}$ to make matrix S full rank when

$$a \geq \frac{e(e-1)}{2}(a-s)^2. \tag{2.91}$$

Then, let $\{\eta_1, \eta_2, \ldots, \eta_{\ell/a}\}$ be a basis for $\mathbb{F}$ over $\mathbb{E}$, we have the $e\ell$ polynomials as $\{\eta_w p_{ij}(x), w \in [\ell/a], i \in [e], j \in [a]\}$. Similar to Construction III, we know that vectors $\{\eta_w V_{ij}, w \in [\ell/a], i \in [e], j \in [a]\}$ are linearly independent over $\mathbb{B}$. Therefore, we can recover the failed nodes and the repair bandwidth is

$$\begin{aligned} b =& rank_{\mathbb{B}}(\{\eta_1 p_{ij}(x), \ldots, \eta_{\ell/a} p_{ij}(x) : i \in [e], j \in [a]\}) \\ =& \frac{\ell}{a} rank_{\mathbb{B}}(\{p_{ij}(x) : i \in [e], j \in [a]\}) \\ \leq& \frac{e\ell}{a}(n-e)(a-s). \end{aligned} \tag{2.92}$$

Thus, the proof is completed. ■

In our scheme, we have constructions for arbitrary a, s, such that $a \mid \ell, s \leq a-1$, while the existing schemes in [17] and [67] mainly considered the special case $\ell = a$. It should be noted that the scheme in [67] can also be used in the case of $s = a-1$ over $\mathbb{E}$ with repair bandwidth $(n-e)e - \frac{e(e-1)(q-1)}{2}$. And, with ℓ/a copies of the code, it can also reach the same repair bandwidth of our scheme. However, by doing so, the code is a vector code, but our scheme constructs a scalar code.

2.4.3 Multiple-erasure repair in multiple cosets

Recall that the scheme in Theorem 5 for a single erasure is a small sub-packetization code with small repair bandwidth for any number of helpers. When there are e erasures and d helpers, $e \leq n-k, k \leq d \leq n-e$, we can recover the erasures one by one using the d helpers. However, inspired by [106], the repaired nodes can be viewed as additional helpers and thus we can reduce the total repair bandwidth. Finally, for every helper, the transmitted information for different failed nodes has some overlap, resulting in a further bandwidth reduction.

The approach we take is similar to that of Section 2.3.3. We take an original code and extend it to a new code with evaluation points as in (2.43). If a helper is in the same coset as any failed node, it transmits naively its entire data; otherwise, it transmits the same amount as the scheme in the original code. After the extension, the new construction decreases the sub-packetization size for fixed n, and the bandwidth is only slightly larger than the original code.

The location of the e erasures are described by $h_i, i \in [e]$, where $0 \leq h_i \leq e, h_1 \geq h_2 \geq ... \geq h_e$, $\sum_{i=1}^{e} h_i = e$. We assume the erasures are located in h_1 cosets, and after removing one erasure in each coset, the remaining erasures are located in h_2 cosets. Then, for the remaining erasures, removing one in each coset, we get the rest of erasures in h_3 cosets, and so on. Figure 2.3 also

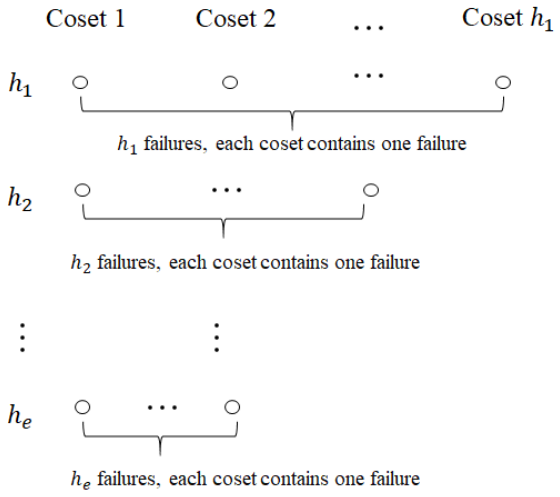

Figure 2.3: Location of the erasures.

shows the erasure locations described above.

In our scheme, we first repair h_1 failures, one from each of the h_1 cosets. Then, for $2 \leq i \leq e$, we repeat the following: After repairing $h_1, h_2, ..., h_{i-1}$ failures, we view these repaired nodes as helpers and repair next h_i failures, one from each of the h_i cosets.

The repair bandwidth of the scheme is showed in the following theorem.

Theorem 8. Let q be a prime number. There exists an $RS(n,k)$ code over $\mathbb{F} = GF(q^\ell)$ for which $\ell = asq_1q_2...q_{\frac{n}{q^a-1}}$, where q_i is the i-th prime number that satisfies $s|(q_i - 1), s = (n-k)!$ and a is an integer. For e erasures and d helpers, $e \leq n-k, k \leq d \leq n-e$, the average repair bandwidth measured in symbols over $\mathbb{B}$ is

$$b \leq \frac{d\ell}{(n-e)}\Bigg[(h_1(q^a-1)-e) + (n-h_1(q^a-1))\sum_{i=1}^{e}\frac{h_i}{d-k+\sum_{v=1}^{i}h_v}\Bigg], \tag{2.93}$$

where $h_i, i \in [e]$ are the parameters that define the location of erasures in Fig. 2.3. For $i \in [e]$, we set $0 \leq h_i \leq e, h_1 \geq h_2 \geq ... \geq h_e$ and $\sum_{i=1}^{e} h_i = e$.

Proof: We first prove the case when a and ℓ' are relatively prime using Lemma 3, the case when a and ℓ' are not necessarily relatively prime are proved in Section 2.8.1. We use the code in [106] as the original code. Let $\mathbb{F}_q(\alpha)$ be the field obtained by adjoining α to the base field $\mathbb{B} = GF(q)$. Similarly let $\mathbb{F}_q(\alpha_1, \alpha_2, \ldots, \alpha_n)$ be the field for adjoining multiple elements. Let α_i be an element of order q_i over $\mathbb{B}$ and h be the number of erasures in the original code. The original code is defined in the field $\mathbb{F}' = GF(q^{\ell'}) = GF(q^{sq_1q_2\cdots q_{n'}})$, which is the degree-s of extension of $\mathbb{F}_q(\alpha_1, \alpha_2, \ldots \alpha_{n'})$. The evaluation points are $A' = \{\alpha_1, \alpha_2, \ldots \alpha_{n'}\}$. The subfield $\mathbb{F}'_{[h]}$ is defined as $\mathbb{F}'_{[h]} = \mathbb{F}_q(\alpha_j, j = h+1, h+2, \ldots, n')$, and $\mathbb{F}'_i$ is defined as $\mathbb{F}_q(\alpha_j, j \neq i, j \in [n'])$.

In the original code, we assume without loss of generality that there are h failed nodes $f'(\alpha_1)$, $f'(\alpha_2), \ldots, f'(\alpha_h)$. Consider the polynomials for failed node $f'(\alpha_i), 1 \leq i \leq h$, as

$$p'_{ij}(x) = x^{j-1} g'_i(x), j \in [s_i], x \in \mathbb{F}', \tag{2.94}$$

where

$$g'_i(x) = \prod_{\alpha \in A'/(R' \cup \{\alpha_i, \alpha_{i+1}, \ldots \alpha_h\})} (x - \alpha), x \in \mathbb{F}', \tag{2.95}$$

for $R' \subseteq A', |R'| = d'$ being the set of helpers. The set of repair polynomials are $\{\eta_{it} p'_{ij}(x), i \in [h], j \in [s_i], t \in [\frac{sq_i}{s_i}]\}$, where $\eta_{it} \in \mathbb{F}'$ are constructed in [106] to ensure that $\{\eta_{it} p'_{i1}(\alpha_i), \eta_{it} p'_{i2}(\alpha_i), \ldots, \eta_{it} p'_{is_i}(\alpha_i)\}$ forms the basis for $\mathbb{F}'$ over $\mathbb{F}'_i$.

Then, the failed nodes are repaired one by one from

$$\begin{aligned} & tr_{\mathbb{F}'/\mathbb{F}'_i}(\upsilon_{\alpha_i} \eta_{it} p'_{ij}(\alpha_i) f'(\alpha_i)) \\ = & - \sum_{\epsilon=1, \epsilon \neq i}^{n} tr_{\mathbb{F}'/\mathbb{F}'_i}(\upsilon_\epsilon \eta_{it} p'_{ij}(\alpha_\epsilon) f'(\alpha_\epsilon)). \end{aligned} \tag{2.96}$$

For $x \notin R' \cup \{\alpha_i, \alpha_{i+1}, \ldots \alpha_h\}$, $p'_{ij}(x) = 0$ and no information is transmitted. Once $f'(\alpha_i)$ is recovered, it is viewed as a new helper for the failures $i+1, i+2, \ldots, h$.

Since $\mathbb{F}'_{[h]} \leq \mathbb{F}'_i$, the information transmitted from the helper α_ϵ can be represented as

$$\begin{aligned}
&tr_{\mathbb{F}'/\mathbb{F}'_i}(\upsilon_\epsilon \eta_{it} p'_{ij}(\alpha_\epsilon) f'(\alpha_\epsilon)) \\
=&tr_{\mathbb{F}'/\mathbb{F}'_i}\left(\xi'_{im} \sum_{m=1}^{q'_i} tr_{\mathbb{F}'_i/\mathbb{F}'_{[h]}}(\upsilon_\epsilon \eta_{it} \xi_{im} p'_{ij}(\alpha_\epsilon) f'(\alpha_\epsilon))\right) \\
=&\sum_{m=1}^{q'_i} \xi'_{im} tr_{\mathbb{F}'/\mathbb{F}'_{[h]}}(\upsilon_\epsilon \eta_{it} \xi_{im} p'_{ij}(\alpha_\epsilon) f'(\alpha_\epsilon)),
\end{aligned} \tag{2.97}$$

where $q'_i = \frac{q_1 q_2 \ldots q_h}{q_i}$, $\{\xi_{i1}, \xi_{i2}, \ldots, \xi_{iq'_i}\}$ and $\{\xi'_{i1}, \xi'_{i2}, \ldots, \xi'_{iq'_i}\}$ are the dual basis for $\mathbb{F}'_i$ over $\mathbb{F}'_{[h]}$. We used the fact that $tr_{\mathbb{F}'/\mathbb{F}'_i}(tr_{\mathbb{F}'_i/\mathbb{F}'_{[h]}}(\cdot)) = tr_{\mathbb{F}'/\mathbb{F}'_{[h]}}(\cdot)$, for $\mathbb{F}'_{[h]} \leq \mathbb{F}'_i \leq \mathbb{F}'$.

The original code satisfies the full rank condition for every $i \in [h]$, and each helper α_ϵ transmits [106]

$$\begin{aligned}
&rank_{\mathbb{F}'_{[h]}}\Bigg(\{\eta_{it}\xi_{im}p'_{ij}(\alpha_\epsilon) : \\
&i \in [h], j \in [s_i], t \in [\frac{sq_i}{s_i}], m \in [q'_i]\}\Bigg) \\
=&rank_{\mathbb{F}'_{[h]}}\Bigg(\{\eta_{it}\xi_{im} : i \in [h], t \in [\frac{sq_i}{s_i}], m \in [q'_i]\}\Bigg) \\
=&\frac{h\ell'}{(d'-k'+h)\prod_{v=h+1}^{n'} p_v}
\end{aligned} \tag{2.98}$$

symbols over $\mathbb{F}'_{[h]}$, which achieves the MSR bound.

In our new code, we extend the field to $\mathbb{F} = GF(q^\ell), \ell = a\ell'$, by adjoining an order-$a$ element α_{n+1} to $\mathbb{F}$. We set $d - k = d' - k'$. The new evaluation points consist of $A = \{\alpha_1\mathbb{E}^*, \alpha_2\mathbb{E}^*, \ldots, \alpha'_n\mathbb{E}^*\}$, $\mathbb{E} = GF(q^a) = \mathbb{F}_q(\alpha_{n+1})$. The subfield $\mathbb{F}_{[h]}$ is defined by adjoining α_{n+1} to $\mathbb{F}'_{[h]}$, and $\mathbb{F}_i$ is defined as $\mathbb{F}_q(\alpha_j, j \neq i, j \in [n+1])$.

Assume first that each coset contains at most one failure, and there are h failures in total. We assume without loss of generality that the evaluation points of the h failed nodes are in $\{\alpha_1\mathbb{E}^*$, $\alpha_2\mathbb{E}^*, \ldots, \alpha_h\mathbb{E}^*\}$, and they are $\alpha_1\gamma_1, \alpha_2\gamma_2, \ldots, \alpha_h\gamma_h$ for some $\gamma_w \in \mathbb{E}, w \in [h]$. Let the set of

helpers be $R \subseteq A$, $|R| = d$. We define the polynomials

$$p_{ij}(x) = x^{j-1} g_i(x), j \in [s_i], x \in \mathbb{F}, \tag{2.99}$$

where

$$g_i(x) = \prod_{\alpha \in A/\{R \cup \{\alpha_i\gamma_i, \alpha_{i+1}\gamma_{i+1}, \ldots \alpha_h\gamma_h\}\}} (x - \alpha), x \in \mathbb{F}. \tag{2.100}$$

The set of repair polynomials are $\{\eta_{it} p_{ij}(x), i \in [h], j \in [s_i], t \in [\frac{sq_i}{s_i}]\}$, where $\eta_{it} \in \mathbb{F}'$ are the same as the original construction. We use field $\mathbb{F}_i$ as the base field for the repair.

$$\begin{aligned} & tr_{\mathbb{F}/\mathbb{F}_i}(\upsilon_{\alpha_i\gamma_i} \eta_{it} p_{ij}(\alpha_i\gamma_i) f(\alpha_i\gamma_i)) \\ =& - \sum_{\alpha \in A, \alpha \neq \alpha_i\gamma_i} tr_{\mathbb{F}/\mathbb{F}_i}(\upsilon_\alpha \eta_{it} p_{ij}(\alpha) f(\alpha)). \end{aligned} \tag{2.101}$$

If $x \in R \cup \{\alpha_i\gamma_i, \alpha_{i+1}\gamma_{i+1}, \ldots \alpha_h\gamma_h\}$, $p_{ij}(x) = 0$ and no information is transmitted. Next, we consider all other nodes.

If $x = \alpha_i\gamma$ for some $\gamma \in \mathbb{E}^*$, we have

$$p_{ij}(x) = \gamma^{j-1} \alpha_i^{j-1} g_i(x). \tag{2.102}$$

Since $\eta_{it}, \alpha_i \in \mathbb{F}'$ and $g_i(x)$ is a constant independent of j, we have

$$\begin{aligned} & rank_{\mathbb{F}_i}(\{\eta_{it} p_{i1}(x), \ldots, \eta_{it} p_{is_i}(x) : t \in [\frac{sq_i}{s_i}]\}) \\ =& rank_{\mathbb{F}_i}(\{\eta_{it}, \ldots, \eta_{it}\alpha_i^{s-1} : t \in [\frac{sq_i}{s_i}]\}) \\ =& rank_{\mathbb{F}'_i}(\{\eta_{it} p'_{i1}(\alpha_i), \ldots, \eta_{it} p'_{is_i}(\alpha_i) : t \in [\frac{sq_i}{s_i}]\}) \end{aligned} \tag{2.103}$$

which indicates the full rank. Note that the last equation follows from Lemma 3. As a result we

can recover the failed nodes and each helper in the cosets containing the failed nodes transmit ℓ symbols in $\mathbb{B}$.

For $x = \alpha_\epsilon \gamma, \epsilon > h$, since $\mathbb{F}_{[h]}$ is a subfield of $\mathbb{F}_i$ and from Lemma 3 we know that $\{\xi_{i1}, \xi_{i2}, \ldots, \xi_{iq'_i}\}$ and $\{\xi'_{i1}, \xi'_{i2}, \ldots, \xi'_{iq'_i}\}$ are also the dual basis for $\mathbb{F}_i$ over $\mathbb{F}_{[h]}$, then, similar to (2.97), we have

$$\begin{aligned}
&tr_{\mathbb{F}/\mathbb{F}_i}(\upsilon_\alpha \eta_{it} p_{ij}(x) f(x)) \\
=&\sum_{m=1}^{q'_i} \xi'_{im} tr_{\mathbb{F}/\mathbb{F}_{[h]}}(\upsilon_\epsilon \eta_{it} \xi_{im} p_{ij}(x) f(x)).
\end{aligned} \tag{2.104}$$

Using the fact that $g_i(x)$ is a constant independent of j, $x \in \mathbb{F}_{[h]}$ and $\eta_{it}\xi_{im} \in \mathbb{F}'$, from Lemma 3 we know that

$$\begin{aligned}
&rank_{\mathbb{F}_{[h]}}\Bigg(\{\eta_{it}\xi_{im} p_{ij}(x) : \\
&i \in [h], j \in [s_i], t \in [\frac{sq_i}{s_i}], m \in [q'_i]\}\Bigg) \\
=&rank_{\mathbb{F}_{[h]}}\Bigg(\{\eta_{it}\xi_{im} : i \in [h], t \in [\frac{sq_i}{s_i}], m \in [q'_i]\}\Bigg) \\
=&rank_{\mathbb{F}'_{[h]}}\Bigg(\{\eta_{it}\xi_{im} : i \in [h], t \in [\frac{sq_i}{s_i}], m \in [q'_i]\}\Bigg) \\
=&rank_{\mathbb{F}'_{[h]}}\Bigg(\{\eta_{it}\xi_{im} p'_{ij}(\alpha_\epsilon) : \\
&i \in [h], j \in [s_i], t \in [\frac{sq_i}{s_i}], m \in [q'_i]\}\Bigg) \\
=&\frac{h\ell'}{(d-k+h)\prod_{v=h+1}^{n'} q_v},
\end{aligned} \tag{2.105}$$

where the last equality follows from (2.98) and $d' - k' = d - k$. So, each helper in the other cosets transmits $\frac{h\ell}{d-k+h}$ symbols over $\mathbb{B}$.

Using the above results, we calculate the repair bandwidth in two steps.

Step 1. We first repair h_1 failures, one from each of the h_1 cosets. From (2.103), we know that in the h_1 cosets containing the failed nodes, we transmit ℓ symbols over $\mathbb{B}$. By (2.105), for each

helper in other cosets, we transmit $\frac{h_1\ell}{d-k+h_1}$ symbols over $\mathbb{B}$.

Step 2. For $2 \leq i \leq e$, repeat the following. After repairing $h_1, h_2, ..., h_{i-1}$ failures, these nodes can be viewed as helpers for repairing next h_i failures, one from each of the h_i cosets. So, we have $d + \sum_{v=1}^{i-1} h_v$ helpers for the h_i failures. For the helpers in the h_1 cosets containing the failed nodes, we already transmit ℓ symbols over $\mathbb{B}$ in Step 1 and no more information needs to be transmitted. For each helper in other cosets, we transmit $\frac{h_i\ell}{d-k+\sum_{v=1}^{i} h_v}$ symbols over $\mathbb{B}$.

Thus, we can repair all the failed nodes. The repair bandwidth can be calculated as (2.93). ■

Suppose that e failures are to be recovered. Compared to the naive strategy which always uses d helpers to repair the failures one by one, our scheme gets a smaller repair bandwidth since the recovered failures are viewed as new helpers and we take advantage of the overlapped symbols for repairing different failures similar to [106].

In the case when $n \gg e(q^a - 1)$, or when we arrange nodes with correlated failures in different cosets, we can assume that all the erasures are in different cosets, $h_1 = e, h_2 = h_3 = ... = h_e = 0$. For example, if correlated failures tend to appear in the same rack in a data center, we can assign each node in the rack to a different coset. Under such conditions, we simplify the repair bandwidth as

$$b \leq \frac{d}{n-e}\frac{e\ell}{d-k+e}(n-e+(d-k)(q^a-2)). \tag{2.106}$$

Indeed, one can examine the expression of (2.93). With the constraint that $\sum_{i=1}^{e} h_i = e$, the first term $h_1(q^a - 1) - e)$ is an increasing function of h_1 and the second term $(n - h_1(q^a - 1)) \sum_{i=1}^{e} \frac{h_i}{d-k+\sum_{v=1}^{i} h_v}$ is a decreasing function of h_1. Under the assumption that n is large, the second term dominates, and increasing h_1 reduces the total repair bandwidth b. Namely, $h_1 = e$ corresponds to the lowest bandwidth for large code length.

Table 2.3: Repair bandwidth of different schemes for e erasures.

	repair bandwidth	number of helpers
Single-erasure repair in one coset (separate repair)	$\frac{e\ell}{a}(n-1)(a-s)$	$n-1$
Multiple-erasure repair in one coset (simultaneous repair)	$\frac{e\ell}{a}(n-e)(a-s)$	$n-e$
Single-erasure repair in multiple cosets (separate repair)	$\frac{e\ell}{n-k}[n-1+(n-k-1)(q^a-2)]$	$n-1$
Multiple-erasure repair in multiple cosets (simultaneous repair)	$\frac{e\ell}{n-k}[n-e+(n-k-e)(q^a-2)]$	$n-e$

In particular, when $d = n - e, h_1 = e$, we have

$$b = \frac{e\ell}{n-k}(n-e) + \frac{e\ell}{n-k}(n-k-e)(q^a-2), \tag{2.107}$$

where the second term is the extra repair bandwidth compared with the MSR bound.

2.4.4 Numerical evaluations and discussions

In this subsection, we compare our schemes for multiple erasures with previous results, including separate repair and schemes in [67] and [106].

We first demonstrate that repairing multiple erasures simultaneously can save repair bandwidth compared to repairing erasures separately. Let us assume e failures happen one by one, and the rest of $n-1$ nodes are available as helpers initially when the first failure occurs. We can either repair each failure separately using $n-1$ helpers, or wait for e failures and repair all of them simultaneously with $n-e$ helpers. Table 2.3 shows the comparison. For our scheme in one coset, separate repair needs a repair bandwidth of $\frac{e\ell}{a}(n-1)(a-s)$ symbols in $\mathbb{B}$, simultaneous a repair requires bandwidth of $\frac{e\ell}{a}(n-e)(a-s)$. For our scheme in multiple cosets, we can repair the failures separately by $n-1$ helpers with the bandwidth of $\frac{e\ell}{n-k}[n-1+(n-k-1)(q^a-2)]$, and with simultaneous repair we can achieve the bandwidth of $\frac{e\ell}{n-k}[n-e+(n-k-e)(q^a-2)]$. One can see that in both constructions, simultaneous repair outperforms separate repair.

Nest we compare our scheme for multiple erasures with the existing schemes. Figure 2.4 shows the

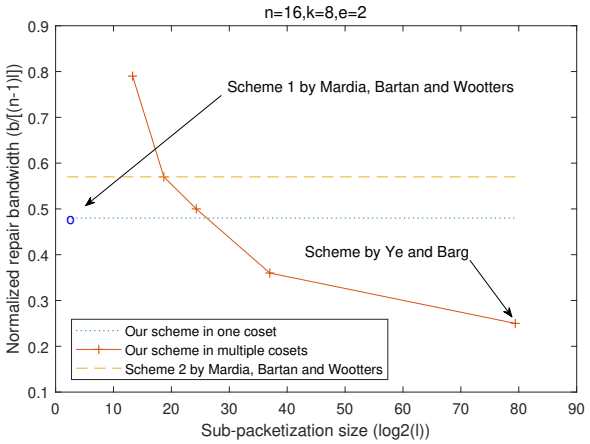

Figure 2.4: Comparison of the schemes, $q = 2, n = 16, k = 8, e = 2$.

Table 2.4: Normalized repair bandwidth($\frac{b}{(n-e)\ell}$) for different schemes when $n = 64, k = 32, e = 2, q = 2$. $\circ$ can be also achieved by Scheme 1 in [67] and $*$ is also achieved by [106].

	$\ell = 6$	$\ell = 7$	$\ell = 8$	$\ell = 9$	...	$\ell = 3.6 \times 10^6$	$\ell = 3.3 \times 10^{11}$	$\ell = 3.9 \times 10^{115}$
Normalized bandwidth for Scheme 1 in [67]	0.42	0.50	0.52	0.52	...	0.52	0.52	0.52
Normalized bandwidth for our scheme in one coset	0.49°	0.49	0.49	0.49	...	0.49	0.49	0.49
Normalized bandwidth for our scheme in multiple cosets						0.52	0.48	0.0625^*

normalized repair bandwidth for different schemes when $n = 16, k = 8, e = 2, q = 2$. Table 2.4 shows the comparison when $n = 64, k = 32, e = 2, q = 2$. We make the following observations:

1. For fixed (n, k) and our scheme with multiple cosets, we use the paremeter a to adjust the sub-packetization size. From Theorem 8, we know that $\ell \approx a \cdot (\frac{n}{q^a-1})^{(\frac{n}{q^a-1})}$, which means that increasing a will decrease the sub-packetization ℓ. In our schemes with one coset and two cosets, the parameter a is determined by the code length n, so increasing ℓ will not change a or the normalized repair bandwidth. When $q = 2$, our code with $a = 1$ coincides with that of [106].
2. For small ℓ and full-length code ($\ell = \log_2 n$), the scheme in [67] has the smallest normalized repair bandwidth. (Our scheme in one coset also achieves the same point as Scheme 1 in [67] when $\ell = \log_2 n$.)
3. When ℓ grows larger ($4 < \ell < 2.1 \times 10^7$ in Figure 2.4, $6 < \ell < 3.3 \times 10^{11}$ in Table 2.4),

our scheme in one coset has the smallest repair bandwidth.

4. For extremely large ℓ ($\ell \geq 2.1 \times 10^7$ in Figure 2.4, $\ell \geq 3.3 \times 10^{11}$ in Table 2.4), our scheme in multiple cosets has the smallest repair bandwidth.
5. The scheme in [106] also achieves one point in both Figure 2.4 and Table 2.4, which can be viewed as a special case of our scheme in multiple cosets.

2.5 Repair Algorithm for $RS(n, k)$ Codes

In this section, we provide a detailed algorithm and the necessary lookup tables to implement the general ideas in [61] to repair $RS(n, k)$ over $GF(2^\ell)$. The computation complexity is analyzed in terms of the number of required finite field operations and required memories. Moreover, for $\ell = 8, 16, 32, 64$, which are frequently used sub-packetization sizes in practice, we provide an efficient computation of the trace function.

First, we describe the repair algorithm using the example of the $RS(14, 10)$ code. The formulas for the general case will be provided at the end.

The $RS(14, 10)$ code has $n = 14$ codeword symbols and $k = 10$ information symbols. In a storage system, different symbols are stored in different nodes. The symbols are over the finite field $\mathbb{F} = GF(2^8) = \{0, 1, \beta, \beta^2, ..., \beta^{254}\}$, where β is the primitive element of $\mathbb{F}$. As a standard choice, β is the root of $1 + x^2 + x^3 + x^4 + x^8 = 0$. For the information symbols $u_j \in \mathbb{F}, j = 0, 1, \cdots, 9$, let f be the polynomial $f(x) = \sum_{j=0}^{9} u_j x^j$. RS codeword symbols are evaluations of the polynomial f and erasures can be corrected using interpolation. In this section, we apply the general construction in [61, Thm. 1] to this specific code. By restricting the evaluation points to a subfield, one obtains a low repair bandwidth. Consider the subfield $\mathbb{E} = GF(2^4) = \{0, 1, \beta^{17}, \beta^{17\cdot 2}, ..., \beta^{17\cdot 14}\}$ of $\mathbb{F}$. Let us choose the set of evaluation points from $\mathbb{E}$, and denote it by $A = \{\alpha_1, \alpha_2, ..., \alpha_{14}\} = \{1, \beta^{17}, \beta^{17\cdot 2}, \beta^{17\cdot 3}, ..., \beta^{17\cdot 13}\}$. Then, the 14 codeword symbols, in the 14 nodes, are $\{N_m =$

$f(\alpha_m) = \sum_{j=0}^{9} u_j {\alpha_m}^j : \alpha_m \in A\}$.

We use the *trace function* to map symbols of $\mathbb{F}$ to sub-symbols of the base field $\mathbb{B} = GF(2)$:

$$tr_{\mathbb{F}/\mathbb{B}}(x) = x + x^2 + x^{2^2} + x^{2^3} + ... + x^{2^7}, \tag{2.108}$$

where for every $x \in \mathbb{F}$, $tr_{\mathbb{F}/\mathbb{B}}(x) \in \mathbb{B}$, i.e., the result of Eq. (2.108) is 0 or 1. Any symbol $x \in \mathbb{F}$ can be reconstructed from the 8 bits evaluated using the trace function as:

$$x = \sum_{i=1}^{8} \gamma_i' tr_{\mathbb{F}/\mathbb{B}}(\gamma_i x) = \sum_{i=1}^{8} \gamma_i' b_i', \tag{2.109}$$

where $\{\gamma_1, \gamma_2, ..., \gamma_8\}$ is a basis for $\mathbb{F}$ over $\mathbb{B}$ and $\{\gamma_1', \gamma_2', ..., \gamma_8'\}$ is its dual basis.

Let us assume the node $N_* = f(\alpha_*)$ fails, $1 \leq * \leq 14$. Then, the transmitted symbols from the other nodes (called helpers) are related to the failed node symbol through the dual codewords $\{c_{mi} : m \in [14]\}$ of $RS(14, 10)$, for $i \in [8]$:

$$c_{*i} N_* = \sum_{m \neq *} c_{mi} N_m. \tag{2.110}$$

8 dual codewords are required for each failure. To transmit symbols of the base field $\mathbb{B}$, we take the trace function on both sides of (2.110). It is required that the 8 symbols $\{c_{*i} : i \in [8]\}$ form a basis for $\mathbb{F}$ over $\mathbb{B}$, then we can treat them as $\{\gamma_i : i \in [8]\}$ and use (2.109) to repair the failed node.

The dual codeword symbols are chosen in [61, Thm. 1] as

$$\{c_{mi} : i \in [8]\} = \{\upsilon_m \eta_t p_{j,*}(\alpha_m) : t \in [2], j \in [4]\}, \tag{2.111}$$

where $m \in [14]$ and the subscript i is indexed by t and j as $i = 4 \times (t-1) + j$. Here $\{\upsilon_m, m \in [14]\} = \{\beta^{51}, \beta^{136}, 1, \beta^{51}, \beta^{221}, \beta^{34}, \beta^{238}, \beta^{136}, \beta^{238}, \beta^{221}, \beta^{102}, \beta^{102}, \beta^{34}, 1\}$ are the col-

umn multipliers, $\{\eta_1, \eta_2\} = \{1, \beta\}$, and the polynomial $p_{j,*}(x)$ is

$$p_{j,*}(x) = \xi_j \prod_{w \in W} (x - \alpha_* + w^{-1}\xi_j), \tag{2.112}$$

where $\xi_j = \beta^{17(j-1)}, W = \{1, \beta^{17}, \beta^{68}\}$.

Let $rank_{\mathbb{B}}(\{a_1, a_2, ..., a_i\})$ be the cardinality of a maximal subset of $\{a_1, a_2, ..., a_i\}$ that is linearly independent over $\mathbb{B}$. As analyzed in [61, Thm. 1], we have

$$rank_{\mathbb{B}}(\{c_{mi} : i \in [8]\}) = \begin{cases} 8, \text{ if } m = *, \\ 4, \text{ if } m \neq *, \end{cases} \tag{2.113}$$

and $\{c_{mi} : i \in [8]\}, m \neq *$ lie in two subspaces of dimension 2 spanned by $\{\upsilon_m \eta_t p_{j,*}(\alpha_m) : j \in [4]\}, t \in [2]$. Now, we have $\{c_{*i} : i \in [8]\}$ form a basis for $\mathbb{F}$ over $\mathbb{B}$, and we can recover the failed node with only 4 bits from each helper. Specifically, for helper m, we represent the basis for the subspace spanned by $\{c_{mi} : i \in [8]\}$ as $\{\epsilon_{m,t,z} : t \in [2], z \in [2]\}$, where$\{\epsilon_{m,t,z} : z \in [2]\}$ are the first two independent elements (hense the basis) of $\{\upsilon_m \eta_t p_{j,*}(\alpha_m) : j \in [4]\}$, for $t \in [2]$. Denote $D_{m,v} = tr_{\mathbb{F}/\mathbb{B}}(\epsilon_{m,t,z} N_m), v = 2(t-1) + z$. Since the trace function is a linear function, $\{D_{m,v} : v \in [4]\}$ can reconstruct $tr_{\mathbb{F}/\mathbb{B}}(c_{mi} N_m), i \in [8]$. Hence the trace function of (2.110) can be evaluated.

For the general case, the $RS(n, k)$ code over $\mathbb{F} = GF(2^\ell)$ is defined by evaluations points in the subfield $\mathbb{E} = GF(2^a)$, for some $a|\ell$. Eq. (2.111) is replaced by

$$\{c_{mi} : i \in [\ell]\} = \{\upsilon_m \eta_t p_{j,*}(\alpha_m) : t \in [\frac{\ell}{a}], j \in [a]\}, \tag{2.116}$$

where $m \in [n]$ and the subscript i is indexed by t and j as $i = a \times (t-1) + j$. Here $\{\upsilon_m, m \in [n]\}$ are the column multipliers given by $\upsilon_m = \prod_{j \in [n], j \neq m} (\alpha_m - \alpha_j)$ [65, Thm. 4, Ch.10], and $\{\eta_t : t \in [\frac{\ell}{a}]\}$

Algorithm 1 Repair algorithm for $RS(n,k)$ over $GF(2^\ell)$.

Input: The failed node index $*$ and the remaining nodes $N_m, m \in [n], m \neq *$.
Output: The failed node N_*.
Preprocessing Steps:
Step 1: Construct **dual basis table** and **dual codeword table** (e.g. Tables 2.5 and 2.6): Find the dual basis of $\{c_{*i} : i \in [\ell]\}$, denoted by $\{\gamma'_1, \gamma'_2, ..., \gamma'_\ell\}$. For each helper node m, find the ℓ dual codewords $c_{mi}, i \in [\ell]$ in (2.111).
Step 2: Construct **subspace basis table** and **representation table** (e.g. Tables 2.7 and 2.8): Find the basis $\{\epsilon_{m,t,z}, t \in [\frac{\ell}{a}], z \in [a-s]\}$ of the subspace spanned by $\{c_{mi} : i \in [\ell]\}$, and the representation of $tr_{\mathbb{F}/\mathbb{B}}(c_{mi}N_m)$ for each helper m.
Repair Steps:
Step 3: Calculate and download the binary values $D_{m,v} = tr_{\mathbb{F}/\mathbb{B}}(\epsilon_{m,t,z}N_m)$ from each helper, $v \in [\frac{\ell}{a}(a-s)]$.
Step 4: Represent $tr_{\mathbb{F}/\mathbb{B}}(c_{mi}N_m)$ from $D_{m,v}$ and apply (2.110):

$$b'_i = tr_{\mathbb{F}/\mathbb{B}}(c_{*i}N_*) = \sum_{m=1, m\neq *}^{n} tr_{\mathbb{F}/\mathbb{B}}(c_{mi}N_m), i \in [\ell]. \tag{2.114}$$

Step 5: Reconstruct the failed node by

$$N_* = \sum_{i=1}^{\ell} \gamma'_i b'_i. \tag{2.115}$$

is the basis for $\mathbb{F}$ over $\mathbb{E}$. The polynomial $p_{j,*}(x)$ is

$$p_{j,*}(x) = \xi_j \prod_{w \in W} (x - \alpha_* + w^{-1}\xi_j), \tag{2.117}$$

where $\xi_j = (\beta^{\frac{2^\ell - 1}{2^a - 1}})^{j-1}, j \in [a]$, and W is an s-dimensional subspace spanned by $\{1, \beta^{\frac{2^\ell - 1}{2^a - 1}}, ..., (\beta^{\frac{2^\ell - 1}{2^a - 1}})^{s-1}\}$ except $\{0\}$, $2^s \leq n - k$. Parameters $\epsilon_{m,t,z}, D_{m,v}$, $v = (a - s) \times (t - 1) + z, z \in [a - s], t \in [\frac{\ell}{a}]$, are defined similar to the case of $RS(14, 10)$. Moreover, it can be shown that $\frac{\ell}{a}(a - s)$ bits are downloaded from each helper. For details see [61].

The detailed steps are provided in Algorithm 1. Due to space limitation, Tables 2.6, 2.7, and 2.8 are only shown when Node 1 fails.

Complexity. In what follows, we analyze the space and computation complexity of our algorithm.

During preprocessing in Steps 1 and 2, Tables 2.5, 2.6, 2.7, and 2.8 need to be calculated only once. In addition, the symbols in Table 2.6 are intermediate values and do not need to be stored. Thus, each node stores one corresponding column of Table 2.5, and one corresponding column of 13 different variations of Tables 2.7 and 2.8, where a variation is for one potential failure. Note that $tr_{\mathbb{F}/\mathbb{B}}(c_{mi}N_m), i \in [8]$ lie in two spaces of dimension 2 spanned by $\{D_{m,1}, D_{m,2}\}$ and $\{D_{m,3}, D_{m,4}\}$, hence each symbol in Table 2.8 is 2 bits. The storage overhead per node is 8 symbols of $GF(2^8)$ in Table 2.5, $4 \times 13 = 52$ symbols of $GF(2^8)$ in Table 2.7 and $8 \times 13 = 104$ symbols of 2 bits in Table 2.8. In total, we need 688 bits per node. For the general case, similar calculations show that the storage overhead per node is $\ell + \frac{\ell}{a}(n - 1)(a - s)$ symbols of $GF(2^\ell)$ and $(n - 1)\ell$ symbols of $a - s$ bits. Moreover, this storage overhead is amortized over the total storage size of the node.

Next, we analyze the computation complexity in the repair steps (Steps 3, 4, and 5). We show that some steps do not require general operations over $\mathbb{F}$. For the commonly used cases of $\ell = 8, 16, 32, 64$, we present a lemma to calculate the trace function.

Lemma 6. Let $\mathbb{F} = GF(2^8), GF(2^{16}), GF(2^{32}), GF(2^{64})$ and $\mathbb{B} = GF(2)$. The trace function $tr_{\mathbb{F}/\mathbb{B}}(x)$ is equal to one single bit of x, for $x \in \mathbb{F}$.

Proof: We prove for the case of $\mathbb{F} = GF(2^8)$, the proof for other fields have similar steps and we only show the results. From [63, Thm 2.25], we know that $tr_{\mathbb{F}/\mathbb{B}}(x) = 0$ if and only if $x = \delta^2 + \delta$ for some $\delta \in \mathbb{F}$. Let us write x as

$$x = x_0 + x_1\beta + x_2\beta^2 + ... + x_7\beta^7, \tag{2.118}$$

where $x_i \in \mathbb{B}, i = 0, 1, ..., 7$ are the 8-bit presentation of x, and β is a root of the primitive polynomial $1 + x^2 + x^3 + x^4 + x^8 = 0$. Similarly, we write δ as $\delta = \delta_0 + \delta_1\beta + \delta_2\beta^2 + ... + \delta_7\beta^7$, where $\delta_i \in \mathbb{B}, i = 0, 1, ..., 7$.

Then, we have $tr_{\mathbb{F}/\mathbb{B}}(x) = 0$ if and only if

$$\begin{aligned} x = \delta^2 + \delta &= \delta_4 + \delta_6 + \delta_7 + (\delta_1 + \delta_7)\beta \\ &+ (\delta_1 + \delta_2 + \delta_4 + \delta_5 + \delta_6)\beta^2 \\ &+ (\delta_3 + \delta_4 + \delta_6)\beta^3 + (\delta_2 + \delta_5 + \delta_7)\beta^4 \\ &+ (\delta_3 + \delta_5)\beta^6 + (\delta_6 + \delta_7)\beta^7. \end{aligned} \tag{2.119}$$

In the above equation, we use the fact that $1 + \beta^2 + \beta^3 + \beta^4 + \beta^8 = 0$ and $\delta_i = {\delta_i}^2$ because δ_i is binary.

Because $\{1, \beta, \beta^2, ..., \beta^7\}$ are linearly independent over $\mathbb{B}$, from (2.118) and (2.119), we get

$tr_{\mathbb{F}/\mathbb{B}}(x) = 0$ if and only if $x_5 = 0$. Thus, $tr_{\mathbb{F}/\mathbb{B}}(x)$ is the same as the 6-th bit of x.

The results for other field sizes are tabulated below:

ℓ	primitive polynomial	bit index
8	$x^8 + x^4 + x^3 + x^2 + 1$	6
16	$x^{16} + x^{14} + x^{10} + x^8 + x^3 + x + 1$	14
32	$x^{32} + x^{22} + x^2 + x + 1$	32
64	$x^{64} + x^4 + x^3 + x^2 + 1$	62

■

In Step 3, we need to perform 4 multiplications in $GF(2^8)$ to calculate $\epsilon_{m,t,z}N_m, t \in [2], z \in [2]$ and 4 trace functions to calculate $D_{m,v}$ from each helper. From Lemma 6, the cost of the trace function is just checking the 6-th bit and can be ignored. In Step 4, as shown in Table 2.8, the representation needs only 2 additions: $D_{m,1} + D_{m,2}$ and $D_{m,3} + D_{m,4}$, and Eq. (2.114) needs 8×13 additions in $GF(2)$. This step is done in the failed node. In Step 5, since $b'_i, i \in [8]$ are binary symbols, at most 7 additions in $GF(2^8)$ are needed. Therefore, in total we need 4 multiplications in $GF(2^8)$ at each helper, plus 130 additions in $GF(2)$ and 7 additions in $GF(2^8)$ at the failed node. For general parameters, we need $\frac{\ell}{a}(a-s)$ multiplications in $GF(2^\ell)$ at each helper, plus $(n-1)\ell \min\left\{\frac{1}{a}\left(2^{a-s} - (a-s) - 1\right) + 1, a - s + 1\right\}$ additions in $GF(2)$, and $\ell - 1$ additions in $GF(2^\ell)$ at the failed node.

2.6 Comparison

For $RS(14, 10)$ in $GF(2^8)$, our algorithm requires a repair bandwidth of 52 bits for one failure, which is 35% better than the naive repair [68] that requires 80 bits. The full-length code in [40], [18] requires 78 bits using the general scheme. For the special case of $RS(14, 10)$ in $GF(2^8)$, [40] found a method that needs at most 64 bits. The MSR scheme in [103] and the asymptotic MSR scheme in [94] do not provide a solution for small fields like $GF(2^8)$. For the special case of $RS(14, 10)$, [25] provides three different repair schemes that require 60, 56, and 54 bits.

Table 2.5: Dual basis table for $RS(14, 10)$ over $GF(2^8)$, $a = 4, s = 2$. β is a root of the primitive polynomial $1 + x^2 + x^3 + x^4 + x^8$.

Failed node	1	2	3	4	5	6	7	8	9	10	11	12	13	14
γ'_1	β^{203}	β^{118}	β^{254}	β^{203}	β^{33}	β^{220}	β^{16}	β^{118}	β^{16}	β^{33}	β^{152}	β^{152}	β^{220}	β^{254}
γ'_2	β^{152}	β^{67}	β^{203}	β^{152}	β^{237}	β^{169}	β^{220}	β^{67}	β^{220}	β^{237}	β^{101}	β^{101}	β^{169}	β^{203}
γ'_3	β^{84}	β^{254}	β^{135}	β^{84}	β^{169}	β^{101}	β^{152}	β^{254}	β^{152}	β^{169}	β^{33}	β^{33}	β^{101}	β^{135}
γ'_4	β^{16}	β^{186}	β^{67}	β^{16}	β^{101}	β^{33}	β^{84}	β^{186}	β^{84}	β^{101}	β^{220}	β^{220}	β^{33}	β^{67}
γ'_5	β^{187}	β^{102}	β^{238}	β^{187}	β^{17}	β^{204}	1	β^{102}	1	β^{17}	β^{136}	β^{136}	β^{204}	β^{238}
γ'_6	β^{136}	β^{51}	β^{187}	β^{136}	β^{221}	β^{153}	β^{204}	β^{51}	β^{204}	β^{221}	β^{85}	β^{85}	β^{153}	β^{187}
γ'_7	β^{68}	β^{238}	β^{119}	β^{68}	β^{153}	β^{85}	β^{136}	β^{238}	β^{136}	β^{153}	β^{17}	β^{17}	β^{85}	β^{119}
γ'_8	1	β^{170}	β^{51}	1	β^{85}	β^{17}	β^{68}	β^{170}	β^{68}	β^{85}	β^{204}	β^{204}	β^{17}	β^{51}

Table 2.6: Dual codeword table. It shows the symbols $c_{mi} = \upsilon_m \eta_t p_{j,*}(\alpha_m), i = 4(t-1)+j$, for $RS(14, 10)$ when Node $* = 1$ fails.

helper m	2	3	4	5	6	7	8	9	10	11	12	13	14
$i = 1(t = 1, j = 1)$	β^{17}	1	0	1	β^{34}	β^{187}	β^{102}	β^{238}	β^{17}	β^{17}	β^{119}	0	β^{119}
$i = 2(t = 1, j = 2)$	β^{17}	β^{119}	β^{68}	0	β^{68}	β^{68}	β^{204}	β^{85}	β^{153}	β^{51}	0	β^{102}	β^{238}
$i = 3(t = 1, j = 3)$	β^{17}	1	β^{204}	0	β^{170}	0	β^{68}	0	β^{17}	β^{51}	β^{238}	β^{136}	β^{17}
$i = 4(t = 1, j = 4)$	β^{119}	β^{119}	0	β^{119}	β^{34}	β^{187}	β^{204}	0	β^{51}	β^{51}	β^{17}	β^{238}	β^{17}
$i = 5(t = 2, j = 1)$	β^{18}	β	0	β	β^{35}	β^{188}	β^{103}	β^{239}	β^{18}	β^{18}	β^{120}	0	β^{120}
$i = 6(t = 2, j = 2)$	β^{18}	β^{120}	β^{69}	0	β^{69}	β^{69}	β^{205}	β^{86}	β^{154}	β^{52}	0	β^{103}	β^{239}
$i = 7(t = 2, j = 3)$	β^{18}	β	β^{205}	0	β^{171}	0	β^{69}	0	β^{18}	β^{52}	β^{239}	β^{137}	β^{18}
$i = 8(t = 2, j = 4)$	β^{120}	β^{120}	0	β^{120}	β^{35}	β^{188}	β^{205}	0	β^{52}	β^{52}	β^{18}	β^{239}	β^{18}

Our approach can be applied to any RS code as long as there is an integer a such that $a|l$ and $2^a > n+1$. For the $RS(11, 8)$ in Yahoo Object Store [69], we can set $n = 11, k = 8, a = 4, s = 1$. When applied to $GF(2^8)$, the repair bandwidth of our algorithm is 60 bits, which is 6% better than the naive scheme. For the $RS(12, 8)$ in Baidu's Atlas cloud storage [56], we can set $n = 12, k = 8, a = 4, s = 2$. When applied to $GF(2^8)$, we can achieve a repair bandwidth of 44 bits, which is 31% smaller than the naive scheme. In both cases, the full-length code's repair bandwidth [40], [18] is higher than that of the naive scheme.

Table 2.7: Subspace basis table. It shows $\epsilon_{m,t,z}$ for $RS(14, 10)$ when Node $* = 1$ fails.

helper m	2	3	4	5	6	7	8	9	10	11	12	13	14
$\epsilon_{m,1,1}$	β^{17}	1	β^{68}	1	β^{34}	β^{187}	β^{102}	β^{238}	β^{17}	β^{17}	β^{119}	β^{102}	β^{119}
$\epsilon_{m,1,2}$	β^{119}	β^{119}	β^{204}	β^{119}	β^{68}	β^{68}	β^{204}	β^{85}	β^{153}	β^{51}	β^{238}	β^{136}	β^{238}
$\epsilon_{m,2,1}$	β^{18}	β	β^{69}	β	β^{35}	β^{188}	β^{103}	β^{239}	β^{18}	β^{18}	β^{120}	β^{103}	β^{120}
$\epsilon_{m,2,2}$	β^{120}	β^{120}	β^{205}	β^{120}	β^{69}	β^{69}	β^{205}	β^{86}	β^{154}	β^{52}	β^{239}	β^{137}	β^{239}

Table 2.8: Representation table. It represents $tr_{\mathbb{F}/\mathbb{B}}(c_{mi}N_m)$ by $D_{m,v} = tr_{\mathbb{F}/\mathbb{B}}(\epsilon_{m,t,z}N_m)$, $v = 2(t-1)+z$, for $RS(14,10)$ when Node 1 fails.

helper	2	3	4	5	6	7	8	9	10	11	12	13	14
$i=1$	$D_{2,1}$	$D_{3,1}$	0	$D_{5,1}$	$D_{6,1}$	$D_{7,1}$	$D_{8,1}$	$D_{9,1}$	$D_{10,1}$	$D_{11,1}$	$D_{12,1}$	0	$D_{14,1}$
$i=2$	$D_{2,1}$	$D_{3,2}$	$D_{4,1}$	0	$D_{6,2}$	$D_{7,2}$	$D_{8,2}$	$D_{9,2}$	$D_{10,2}$	$D_{11,2}$	0	$D_{13,1}$	$D_{14,2}$
$i=3$	$D_{2,1}$	$D_{3,1}$	$D_{4,2}$	0	$D_{6,1}+D_{6,2}$	0	$D_{8,1}+D_{8,2}$	0	$D_{10,1}$	$D_{11,2}$	$D_{12,2}$	$D_{13,2}$	$D_{14,1}+D_{14,2}$
$i=4$	$D_{2,2}$	$D_{3,2}$	0	$D_{5,2}$	$D_{6,1}$	$D_{7,1}$	$D_{8,2}$	0	$D_{10,1}+D_{10,2}$	$D_{11,2}$	$D_{12,1}+D_{12,2}$	$D_{13,1}+D_{13,2}$	$D_{14,1}+D_{14,2}$
$i=5$	$D_{2,3}$	$D_{3,3}$	0	$D_{5,3}$	$D_{6,3}$	$D_{7,3}$	$D_{8,3}$	$D_{9,3}$	$D_{10,3}$	$D_{11,3}$	$D_{12,3}$	0	$D_{14,1}$
$i=6$	$D_{2,3}$	$D_{3,4}$	$D_{4,3}$	0	$D_{6,4}$	$D_{7,4}$	$D_{8,4}$	$D_{9,4}$	$D_{10,4}$	$D_{11,4}$	0	$D_{13,3}$	$D_{14,2}$
$i=7$	$D_{2,3}$	$D_{3,3}$	$D_{4,4}$	0	$D_{6,3}+D_{6,4}$	0	$D_{8,3}+D_{8,4}$	0	$D_{10,3}$	$D_{11,4}$	$D_{12,4}$	$D_{13,4}$	$D_{14,1}+D_{14,2}$
$i=8$	$D_{2,4}$	$D_{3,4}$	0	$D_{5,4}$	$D_{6,3}$	$D_{7,3}$	$D_{8,4}$	0	$D_{10,3}+D_{10,4}$	$D_{11,4}$	$D_{12,3}+D_{12,4}$	$D_{13,3}+D_{13,4}$	$D_{14,1}+D_{14,2}$

2.7 Conclusion

In this chapter, we designed three Reed-Solomon code repair schemes to provide a tradeoff between the sub-packetization size and the repair bandwidth. Our schemes choose the evaluation points of the Reed-Solomon code from one, two, or multiple cosets of the multiplicative group of the underlying finite field. For a single erasure, when the sub-packetization size is large, the scheme in multiple cosets has better performance, it approaches the MSR bound. When sub-packetization size is small, the scheme in one coset has advantages in repair bandwidth. The scheme in two cosets has smaller repair bandwidth with certain parameters in between the other two cases. For multiple erasures, our scheme in one coset has constructions for arbitrary redundancy $n-k$ and our scheme in multiple cosets reduced the sub-packetization size of an MSR code. The two schemes together provided a set of tradeoff points and we observe similar tradeoff characteristics as in the single erasure case. In spite of several tradeoff points we provided in this chapter, the dependence of the sub-packetization size versus the repair bandwidth is still an open question.

2.8 Detailed Proofs

2.8.1 Proof of schemes for the case of arbitrary a and ℓ'.

In this section, we first introduce a lemma similar to Lemma 3 that does not require a and ℓ' to be relatively prime. By applying this lemma, our constructions in multiple cosets for single and multiple erasures can be generalized when a and ℓ' are arbitrary integers.

We note that a finite field $\mathbb{F} = GF(q^\ell)$ is also a vector space over $GF(q)$. Let $\mathbb{E}$ be a subspace of $\mathbb{F}$. Define the subspace spanned by a set of elements $\{\gamma_1, \gamma_2, \ldots, \gamma_i\} \subseteq \mathbb{F}$ over $\mathbb{E}$ as $\text{span}_{\mathbb{E}}\{\gamma_1, \gamma_2, \ldots, \gamma_i\} \triangleq \{\sum_{j=1}^{i} b_j\gamma_j : b_j \in \mathbb{E}\}$. The rank $rank_{\mathbb{E}}(\{\gamma_1, \gamma_2, ..., \gamma_i\})$ is defined to be the cardinality of a maximal subset of $\{\gamma_1, \gamma_2, ..., \gamma_i\}$ that is linearly independent over $\mathbb{E}$.

Lemma 7. Let $\mathbb{B} = GF(q), \mathbb{F}' = GF(q^{\ell'}), \mathbb{F} = GF(q^\ell), \ell = a\ell'$, and q be any power of a prime number. Define the subspace $\mathbb{E} = \text{span}_{\mathbb{B}}\{\beta_1, \beta_2, \ldots, \beta_a\}$, where $\{\beta_1, \beta_2, \ldots, \beta_a\}$ is a basis for $\mathbb{F}$ over $\mathbb{F}'$. For any set of $\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\} \subseteq \mathbb{F}' \leq \mathbb{F}$, we have

$$\begin{aligned} &rank_{\mathbb{E}}(\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\}) \\ =&rank_{\mathbb{B}}(\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\}). \end{aligned} \tag{2.120}$$

Proof: Assume $rank_{\mathbb{B}}(\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\}) = c$ and without loss of generality, $\{\gamma_1, \gamma_2, ..., \gamma_c\}$ are linearly independent over $\mathbb{B}$. Then, we can construct $\{\gamma'_{c+1}, \gamma'_{c+2}, ..., \gamma'_{\ell'}\} \subseteq \mathbb{F}'$ to make $\{\gamma_1, \gamma_2, ..., \gamma_c, \gamma'_{c+1}, \gamma'_{c+2}, ..., \gamma'_{\ell'}\}$ form a basis for $\mathbb{F}'$ over $\mathbb{B}$.

Since $\{\beta_1, \beta_2, \ldots, \beta_a\}$ is the basis for $\mathbb{F}$ over $\mathbb{F}'$, we know that $\{\beta_i\gamma_1, \beta_i\gamma_2, ..., \beta_i\gamma_c, \beta_i\gamma'_{c+1}, \beta_i\gamma'_{c+2}, ..., \beta_i\gamma'_{\ell'} : i \in [a]\}$ is the basis for $\mathbb{F}$ over $\mathbb{B}$. Then, we have $\mathbb{F} = \text{span}_{\mathbb{E}}\{\gamma_1, \gamma_2, ..., \gamma_c, \gamma'_{c+1}, \gamma'_{c+2}, ..., \gamma'_{\ell'}\}$, namely, $\{\gamma_1, \gamma_2, ..., \gamma_c, \gamma'_{c+1}, \gamma'_{c+2}, ..., \gamma'_{\ell'}\}$ is a basis for $\mathbb{F}$ over $\mathbb{E}$, hence $\{\gamma_1, \gamma_2, ..., \gamma_c\}$ are

linearly independent over $\mathbb{E}$,

$$\begin{aligned} & rank_{\mathbb{E}}(\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\}) \\ \geq & c \\ = & rank_{\mathbb{B}}(\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\}). \end{aligned} \tag{2.121}$$

Since $\mathbb{B} \subseteq \mathbb{E}$, we also have

$$\begin{aligned} & rank_{\mathbb{E}}(\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\}) \\ \leq & rank_{\mathbb{B}}(\{\gamma_1, \gamma_2, ..., \gamma_{\ell'}\}). \end{aligned} \tag{2.122}$$

The proof is completed. ■

For the schemes in multiple cosets when a and ℓ' are not relatively prime, we just use the subspace $\mathbb{E} = \text{span}_{\mathbb{B}}\{\beta_1, \beta_2, ..., \beta_a\}$ to replace the subfield $GF(q^a)$. We denote by $\mathbb{E}^* = \mathbb{E}\backslash\{0\}$ for the subspace $\mathbb{E}$. The evaluation points of the new code are $\{\gamma\mathbb{E}^* : \gamma \in A'\}$ where $A' \subseteq \mathbb{F}'$ is the set of evaluation points for the original code. In the proofs, we use Lemma 7 instead of Lemma 3. For example, from Lemma 7 we know that the new evaluation points are all distinct if the elements in A' are linearly independent over $\mathbb{B}$.

2.8.2 Proof of Theorem 6

In this section, we prove the equivalence of Definitions 1 and 2. We first show that the dual code scheme in Definition 2 reduces to a linear repair scheme as in Definition 1 in Lemma 8. Then, we show that Definition 1 reduces to Definition 2 in Lemma 9 and Lemma 10.

Lemma 8. The dual code scheme can be reduced to the linear repair scheme in Definition 1.

Proof: In the dual code scheme, we repair nodes $[e]$ from the linearly independent equations

$$\sum_{v=1}^{e} tr_{\mathbb{F}/\mathbb{B}}(C'_{ijv}C_v) = -\sum_{t=e+1}^{n} tr_{\mathbb{F}/\mathbb{B}}(C'_{ijt}C_t), i \in [e], j \in [\ell]. \tag{2.123}$$

Here, C'_{ijt} can be written as

$$C'_{ijt} = \sum_{m=1}^{\ell} \xi'_m tr_{\mathbb{F}/\mathbb{B}}(\xi_m C'_{ijt}), \tag{2.124}$$

where $\{\xi_1, \xi_2, \ldots, \xi_\ell\}$ and $\{\xi'_1, \xi'_2, \ldots, \xi'_\ell\}$ are the dual basis for $\mathbb{F}$ over $\mathbb{B}$. Then, we can rewrite (2.123) in matrix form as

$$\begin{aligned} &\sum_{v=1}^{e} S_{iv} \begin{bmatrix} tr_{\mathbb{F}/\mathbb{B}}(\xi'_1 C_v) \\ tr_{\mathbb{F}/\mathbb{B}}(\xi'_2 C_v) \\ \vdots \\ tr_{\mathbb{F}/\mathbb{B}}(\xi'_\ell C_v) \end{bmatrix} \\ &= -\sum_{t=e+1}^{n} S_{it} \begin{bmatrix} tr_{\mathbb{F}/\mathbb{B}}(\xi'_1 C_v) \\ tr_{\mathbb{F}/\mathbb{B}}(\xi'_2 C_v) \\ \vdots \\ tr_{\mathbb{F}/\mathbb{B}}(\xi'_\ell C_v) \end{bmatrix}, i \in [e], \end{aligned} \tag{2.125}$$

where $S_{it} \in \mathbb{B}^{\ell \times \ell}$ is called the repair matrix defined as

$$S_{it} = \begin{bmatrix} tr_{\mathbb{F}/\mathbb{B}}(\xi_1 C'_{i1t}) & \cdots & tr_{\mathbb{F}/\mathbb{B}}(\xi_\ell C'_{i1t}) \\ tr_{\mathbb{F}/\mathbb{B}}(\xi_1 C'_{i2t}) & \cdots & tr_{\mathbb{F}/\mathbb{B}}(\xi_\ell C'_{i2t}) \\ \vdots & \ddots & \vdots \\ tr_{\mathbb{F}/\mathbb{B}}(\xi_1 C'_{i\ell t}) & \cdots & tr_{\mathbb{F}/\mathbb{B}}(\xi_\ell C'_{i\ell t}) \end{bmatrix}, \tag{2.126}$$

Let

$$X_w \triangleq \begin{bmatrix} tr_{\mathbb{F}/\mathbb{B}}(\xi_1' C_w) \\ tr_{\mathbb{F}/\mathbb{B}}(\xi_2' C_w) \\ \vdots \\ tr_{\mathbb{F}/\mathbb{B}}(\xi_\ell' C_w) \end{bmatrix}, w \in [n]. \tag{2.127}$$

We want to solve $X_i, i \in [e]$, which can be used to recover the e failed nodes $C_1, C_2, \ldots, C_e$. Define matrix S as

$$S = \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1e} \\ S_{21} & S_{22} & \cdots & S_{2e} \\ \vdots & \vdots & \ddots & \vdots \\ S_{e1} & S_{e2} & \cdots & S_{ee} \end{bmatrix}. \tag{2.128}$$

Then, (2.125) can be represented as

$$S \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_e \end{bmatrix} = - \sum_{t=e+1}^{n} \begin{bmatrix} S_{1t} \\ S_{2t} \\ \vdots \\ S_{et} \end{bmatrix} X_t. \tag{2.129}$$

Thus, from Lemma 4 we know that if the full rank condition satisfies[1], S has full rank so we can solve $X_i, i \in [e]$. Then, $C_i, i \in [e]$ can be repaired from

$$C_i = \sum_{m=1}^{\ell} \xi_m tr_{\mathbb{F}/\mathbb{B}}(\xi_m' C_i), \tag{2.130}$$

[1]We use Lemma 4 while in Lemma 4 we use the polynomials $p_{ij}(x)$ as part of the elements in the defined matrix S_{it}. However, in Lemma 4 we just view the polynomials $p_{ij}(x)$ as a symbol, change them to the dual codeword symbol C'_{ijt} will not have effects on the results of the lemma.

Now, set $\mu_{im} = \xi_m$, we get $\lambda_{im} = tr_{\mathbb{F}/\mathbb{B}}(\xi'_m C_i)$, which can be solved from (2.129). Note that the right side of (2.129) is equal to the right side of (2.123), we get the queries $Q_t = \{C'_{ijt}, i \in [e], j \in [\ell]\}$ and the coefficients $\beta_{im\gamma t}$ come from matrix S.

Then, we can get the repair bandwidth condition:

$$b = \sum_{t=e+1}^{n} rank_{\mathbb{B}}(\{C'_{ijt} : i \in [e], j \in [\ell]\}) = \sum_{t=e+1}^{\ell} rank_{\mathbb{B}}(Q_t). \tag{2.131}$$

■

Lemma 9. A linear repair scheme in Definition 1 can be represented in the form below:

$$\mu'_{ij} C_i = \sum_{t=e+1}^{n} \theta_{ijt} C_t, i \in [e], j \in [\ell], \tag{2.132}$$

where $\{\mu'_{i1}, \mu'_{i2}, ..., \mu'_{i\ell}\}$ is the dual basis of $\{\mu_{i1}, \mu_{i2}, ..., \mu_{i\ell}\}$, and $\theta_{ijt} \in \text{span}_{\mathbb{B}}(Q_t)$, $e+1 \leq t \leq n, i \in [e], j \in [\ell]$ are some coefficients in $\mathbb{F}$. The repair bandwidth is

$$b = \sum_{t=e+1}^{n} rank_{\mathbb{B}}(\{\theta_{ijt} : i \in [e], j \in [\ell]\}). \tag{2.133}$$

Proof: By (2.66) and (2.67) we have

$$\sum_{t=e+1}^{n} \sum_{\gamma \in Q_t} tr_{\mathbb{F}/\mathbb{B}}(\beta_{ij\gamma t} \cdot \gamma C_t) = \lambda_{ij} = tr_{\mathbb{F}/\mathbb{B}}(\mu'_{ij} C_i). \tag{2.134}$$

Set $\theta_{ijt} = \sum_{\gamma \in Q_t} \beta_{ij\gamma t} \cdot \gamma$. Then, we have $\theta_{ijt} \in \text{span}_{\mathbb{B}}(Q_t)$. Hence,

$$\begin{aligned} &\sum_{t=e+1}^{n} tr_{\mathbb{F}/\mathbb{B}}(\theta_{ijt} C_t) \\ =&\sum_{t=e+1}^{n} tr_{\mathbb{F}/\mathbb{B}}(\sum_{\gamma \in Q_t} \beta_{ij\gamma t} \cdot \gamma C_t) \\ =&tr_{\mathbb{F}/\mathbb{B}}(\mu'_{ij} C_i). \end{aligned} \tag{2.135}$$

Equations (2.134) and (2.135) hold for all $f \in \mathbb{F}[x]$. Since the RS code is a linear code, they also hold for $\delta_m \cdot f \in \mathbb{F}[x]$ for all $\delta_m \in \mathbb{F}$. In particular, let $\delta_m, m \in [\ell]$, be a basis for $\mathbb{F}$ over $\mathbb{B}$, Then,

$$tr_{\mathbb{F}/\mathbb{B}}(\delta_m \cdot \mu'_{ij} C_i) = tr_{\mathbb{F}/\mathbb{B}}(\delta_m \cdot \sum_{t=e+1}^{n} \theta_{ijt} C_t), \forall m \in [\ell], \tag{2.136}$$

which in turn implies that θ_{ijt} also satisfies (2.132). ■

Note that the repair bandwidth (2.133) also satisfies

$$\begin{aligned} b &= \sum_{t=e+1}^{n} rank_{\mathbb{B}}(\theta_{ijt} : i \in [e], j \in [\ell]) \\ &\leq \sum_{t=e+1}^{n} rank_{\mathbb{B}}(Q_t), \end{aligned} \tag{2.137}$$

since $\theta_{ijt} \in \text{span}_{\mathbb{B}}(Q_t)$. However, for any linear scheme L in Definition 1, if (2.137) holds with strict inequality, we can improve the linear scheme L by setting Q_t such that $\text{span}_{\mathbb{B}}(Q_t) = \text{span}_{\mathbb{B}}(\{\theta_{ijt}, i \in [e], j \in [\ell]\})$, for all $e+1 \leq t \leq n$. Hence, the linear scheme L and the scheme in Lemma 9 have identical bandwidth.

Lemma 10. The scheme in Lemma 9 can be represented by the dual code scheme in Definition 2.

Proof: By (2.132), $(0, \ldots, \mu'_{ij}, \ldots, 0, \theta_{ije+1}, \ldots, \theta_{ijn})$ is a dual codeword, where μ'_{ij} is the i-th entry. Then, for $j \in [\ell]$, we set C'_{ijt} such that $C'_{iji} = -\mu'_{ij}$, $C'_{ijv} = 0, v \in [e], v \neq i$ and $C'_{ijt} = \theta_{ijt}, e+1 \leq t \leq n$. The full rank condition follows because $\{\mu'_{i1}, \mu'_{i2}, ..., \mu'_{i\ell}\}$ is the basis for $\mathbb{F}$ over $\mathbb{B}$, and the repair bandwidth condition follows from (2.137). Thus, we obtain the dual code scheme in Definition 2.

■

2.8.3 Proof of Lemma 4

Proof: Vectors $V_{ij}, i \in [e], j \in [\ell]$ are linearly independent over $\mathbb{B}$ is equivalent to that there is no nonzero $b_{ij} \in \mathbb{B}, i \in [e], j \in [\ell]$ that satisfy

$$\sum_{i,j} b_{ij} p_{ij}(\alpha_v) = 0, \forall v \in [e]. \tag{2.138}$$

Here, $p_{ij}(x)$ can be written as

$$p_{ij}(x) = \sum_{m=1}^{\ell} \xi'_m tr_{\mathbb{F}/\mathbb{B}}(\xi_m p_{ij}(x)), \tag{2.139}$$

where $\{\xi_1, \xi_2, \ldots, \xi_\ell\}$ and $\{\xi'_1, \xi'_2, \ldots, \xi'_\ell\}$ are the dual basis for $\mathbb{F}$ over $\mathbb{B}$. So, it is equivalent to that there is no nonzero $b_{ij} \in \mathbb{B}, i \in [e], j \in [\ell]$ that satisfy

$$\sum_{i,j} b_{ij} \sum_{m=1}^{\ell} \xi'_m tr_{\mathbb{F}/\mathbb{B}}(\xi_m p_{ij}(\alpha_v)) = 0, \forall v \in [e]. \tag{2.140}$$

Since $\{\xi'_1, \xi'_2, \ldots, \xi'_\ell\}$ are linearly independent over $\mathbb{B}$. Therefore, there is no nonzero $b_{ij} \in \mathbb{B}, i \in [e], j \in [\ell]$ that satisfy

$$\sum_{i,j} b_{ij} tr_{\mathbb{F}/\mathbb{B}}(\xi_m p_{ij}(\alpha_v)) = 0, \forall v \in [e], m \in [\ell], \tag{2.141}$$

which is equivalent to S has full rank. ■

2.8.4 Proof of Lemma 5

Proof: Assume $rank_{\mathbb{B}}(\{p_{ij}(\alpha_t), i \in [e], j \in [\ell]\}) = c$ and $\{p_{ij}(\alpha_t), (i,j) \in I\}$ are linearly independent over $\mathbb{B}, |I| = c$. Define $S_{it}(j)$ as the vector for the j-th row in S_{it}: $S_{it}(j) = (tr_{\mathbb{F}/\mathbb{B}}(\xi_1 p_{ij}(\alpha_t)),\ tr_{\mathbb{F}/\mathbb{B}}(\xi_2 p_{ij}(\alpha_t)), ..., tr_{\mathbb{F}/\mathbb{B}}(\xi_\ell p_{ij}(\alpha_t)))$. We first prove $\{S_{it}(j), (i,j) \in I\}$ are

linearly independent and then prove $S_{i't}(j'), i' \in [e], j' \in [\ell], (i', j') \notin I$ can be represented as $\mathbb{B}$-linear combinations of $\{S_{it}(j), (i,j) \in I\}$.

If $\{S_{it}(j), (i,j) \in I\}$ are linearly dependent over $\mathbb{B}$, then there exists some nonzero $b_{ij} \in \mathbb{B}, (i,j) \in I$ that satisfies

$$\sum_{(i,j)\in I} b_{ij} S_{it}(j) = 0, \tag{2.142}$$

and we have

$$\sum_{(i,j)\in I} b_{ij} tr_{\mathbb{F}/\mathbb{B}}(\xi_m p_{ij}(\alpha_t)) = 0, \forall m \in [\ell]. \tag{2.143}$$

Multiplying the above equation by ξ'_m and summing over all $m \in [\ell]$ result in

$$\sum_{(i,j)\in I} \sum_{m=1}^{\ell} b_{ij} \xi'_m tr_{\mathbb{F}/\mathbb{B}}(\xi_m p_{ij}(\alpha_t)) = 0. \tag{2.144}$$

Then, from (2.139) we know that b_{ij} satisfies

$$\sum_{(i,j)\in I} b_{ij} p_{ij}(\alpha_t) = 0, \tag{2.145}$$

which is contradictory to the statement that $\{p_{ij}(\alpha_t), (i,j) \in I\}$ are linearly independent over $\mathbb{B}$. Therefore, $\{S_{it}(j), (i,j) \in I\}$ are linearly independent over $\mathbb{B}$.

Let us assume $p_{i'j'}(\alpha_t), i' \in [e], j' \in [\ell], (i', j') \notin I$ can be represented as

$$p_{i'j'}(\alpha_t) = \sum_{(i,j)\in I} b'_{ij} p_{ij}(\alpha_t), \text{ for some } b'_{ij} \in \mathbb{B}. \tag{2.146}$$

Then, for $m \in [\ell]$,

$$tr_{\mathbb{F}/\mathbb{B}}(\xi_m p_{i'j'}(\alpha_t)) = tr_{\mathbb{F}/\mathbb{B}}\left(\xi_m \sum_{(i,j)\in I} b'_{ij} p_{ij}(\alpha_t)\right)$$
$$= \sum_{(i,j)\in I} b'_{ij} tr_{\mathbb{F}/\mathbb{B}}(\xi_m p_{ij}(\alpha_t)), \tag{2.147}$$

which means that for $i' \in [e], j' \in [\ell], (i', j') \notin I$,

$$S_{i't}(j') = \sum_{(i,j)\in I} b'_{ij} S_{it}(j) \tag{2.148}$$

is the $\mathbb{B}$-linear combination of $\{S_{it}(j), (i,j) \in I\}$. ■

Chapter 3

Storage Codes with Flexible Number of Nodes

3.1 Introduction

In distributed systems, error-correcting codes are ubiquitous to achieve high efficiency and reliability. However, most of the codes have a fixed redundancy level, while in practical systems, the number of failures varies over time. When the number of failures is smaller than the designed redundancy level, the redundant storage nodes are not used efficiently. In this chapter, we present *flexible storage codes* that make it possible to recover the entire information through accessing a flexible number of nodes.

An (n, k, ℓ) *(array) code* over a finite field $\mathbb{F}$ is denoted by $(C_1, C_2, ..., C_n), C_i = (C_{1,i}, C_{2,i}, \ldots, C_{\ell,i})^T \in \mathbb{F}^\ell$, where n is the codeword length, k is the dimension, and ℓ is the size of each node (or codeword symbol) and is called the *sub-packetization* size. For an (n, k, ℓ) code, assume we can recover the entire information by downloading all the symbols from any R nodes. We define the download time of the slowest node among the R nodes as the *data access latency*. In practical

systems, the number of available nodes might be different over time and the latency of each node can be modelled as a random variable [62]. Waiting for downloading all ℓ symbols from exactly R nodes may result in a large delay. Hence, it is desirable to be able to adjust R and ℓ according to the number of failures. Motivated by reducing the data access latency, we propose flexible storage codes below.

A *flexible storage codes* is an (n, k, ℓ) code that is parameterized by a given integer a and a set of tuples $\{(R_j, k_j, \ell_j) : 1 \leq j \leq a\}$ that satisfies

$$k_j \ell_j = k\ell, 1 \leq j \leq a, k_1 > k_2 > ... > k_a = k, \ell_a = \ell, \tag{3.1}$$

and if we take ℓ_j particular coordinates of each codeword symbol, denoted by $(C_{m_1,i}, C_{m_2,i}, \ldots, C_{m_{\ell_j},i})^T \in \mathbb{F}^{\ell_j}$, $i \in [n]$, where $[n]$ is the set of integers smaller or equal to n, we can recover the entire information from any R_j nodes.

For example, *flexible maximum distance separable (MDS) codes* are codes satisfying the singleton bound for each k_j, namely, $R_j = k_j$, $1 \leq j \leq a$. Fig. 3.1 shows an example. $C_{1,1}, C_{1,2}, C_{1,3}$, $C_{2,1}, C_{2,2}, C_{2,3}$ are the 6 information symbols. $W_1 = C_{1,1} + C_{1,2} + C_{1,3}, W_1' = C_{1,1} + 2C_{1,2} + 3C_{1,3}$ are the parities for $C_{1,1}, C_{1,2}, C_{1,3}$, and $W_2 = C_{2,1} + C_{2,2} + C_{2,3}, W_2' = C_{2,1} + 2C_{2,2} + 3C_{2,3}$ are the parities for $C_{2,1}, C_{2,2}, C_{2,3}$. The accessed symbols in each scenario are marked as red. $W_3' = W_1' + W_2', W_4' = W_1' + 2W_2'$ are the parities of W_1' and W_2'. In Scenario 1, all the information symbols are accessed, we obtain the entire information directly. In Scenario 2, W_1' and W_2' are also the parities in Rows 1 and 2, respectively. Thus, we obtain 3 symbols in the first two rows, and the entire information can be decoded.

It is easy to see that the flexible code in the above example has a better expected latency than a fixed code with either $k = 2$ or 3. In particular, each node can read and then send its three symbols one by one to the decoder (in practice, each symbol can be viewed as, for example, several Megabytes when multiple copies of the same code are applied). The flexible decoder can wait until 2 symbols

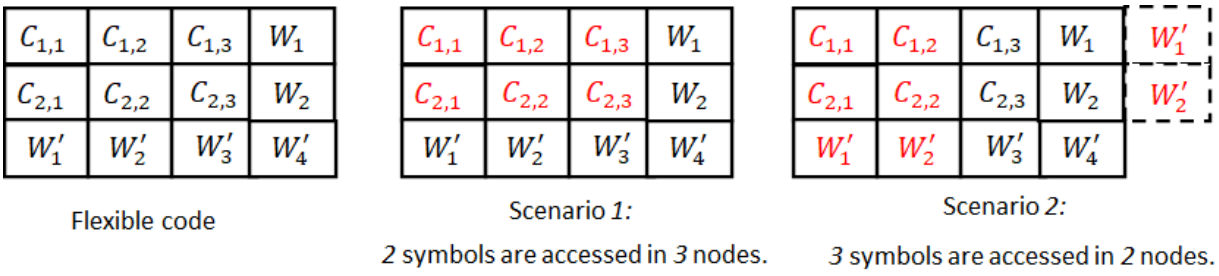

Figure 3.1: Example of a $(4, 2, 3)$ flexible MDS code over $GF(5)$.

from any 3 nodes, or 3 symbols from any 2 nodes are delivered, whose latency is the minimum of the two fixed codes.

Several constructions of flexible MDS codes exist in the literature, though intended for different application scenarios, including error-correcting codes [95], universally decodable matrices [30, 76], secrete sharing [44], and private information retrieval [5]. However, for other important types of storage codes, such as codes that efficiently recover from a single node failure, or codes that correct mixed types of node and symbol failures, flexible constructions remain an open problem. In this chapter, we provide a framework that can produce flexible storage codes for different code families. The main contributions of the chapter are summarized below.

• **A framework for flexible codes** is proposed that can generate flexible storage codes given a construction of fixed (non-flexible) storage code.

• **Flexible LRC (locally recoverable) codes** allow information reconstruction from a variable number of available nodes while maintaining the locality property, providing efficient single node recovery. For an (n, k, ℓ, r) flexible LRC code parametrized by $\{(R_j, k_j, \ell_j) : 1 \leq j \leq a\}$ that satisfies (3.1) and $R_j = k_j + \frac{k_j}{r} - 1$, each single node failure can be recovered from a subset of r nodes, while the total information is reconstructed by accessing ℓ_j symbols in R_j nodes. We provide code constructions based on the optimal LRC code construction [92].

• **Flexible PMDS (partial MDS) codes** are designed to tolerate a flexible number of *node failures*

and a given number of extra *symbol failures*, desirable for solid-state drives due to the presence of mixed types of failures. We provide an (n, k, ℓ, s) with a set of $\{(R_j, k_j, \ell_j) : 1 \leq j \leq a\}$ satisfying (3.1) and $R_j = k_j$ such that when ℓ_j symbols are accessed in each node, we can tolerate $n - R_j$ failures and s extra symbol failures. We construct flexible codes from the PMDS code [13].

• **Flexible MSR (minimum storage regenerating) codes** are type of flexible MDS codes such that a single node failure is recovered by downloading the minimum amount of information from the available nodes. Both vector and scalar codes are obtained by applying our flexible code framework to the MSR codes in [104] and [94].

• **Latency analysis** is carried out for flexible storage codes. It is demonstrated that our flexible storage codes always have a lower latency compared to the corresponding fixed codes. Also, applying our flexible codes to the matrix-vector multiplication scenario, we show simulation results from Amazon clusters that we can improve 6% for $n = 8, R_1 = 5, R_2 = 4, \ell_1 = 12, \ell_2 = 15$ and matrix size of 1500×1500.

Related work. The flexibility idea was first proposed in [45] to minimize a cost function such as a linear combination of bandwidth, delay or the number of hops. Flexible MDS codes were first proposed in [95]. In [95], one can recover the entire information by downloading ℓ_j symbols from any k_j nodes. However, each of the k_j nodes needs to first read *all* the ℓ symbols and then calculate and transmit the ℓ_j symbols required for decoding. The aim of [95] is to reduce the bandwidth instead of the number of accessed symbols. Universally decodable matrices (UDM) [30, 76] can also be used for the flexible MDS problem. UDM is a generalization of flexible MDS code where the decoder can obtain different number of symbols from the nodes. In particular, from the first v_i symbols from node C_i, for any $v_i, 1 \leq i \leq n$ such that $\sum_{i=1}^{n} v_i \geq k\ell$, the entire information can be recovered. Flexibility problems are also considered for secret sharing [44, 96, 112, 79] and private information retrieval [5, 89, 4, 19, 90, 91], such that the number of available nodes is flexible. The constructions in [44] and [5] are equivalent to each other and they achieved optimal decoding bandwidth while keeping secrecy or privacy from other parties. When we remove the secrecy or

privacy requirement, these constructions become flexible MDS codes. All of [95, 30, 76, 44, 5] achieve the optimal field size of $|\mathbb{F}| = n$.

There are several works on latency and flexibility in the literature in distributed coded computing [57, 73, 102, 100]. Specifically, fixed MDS codes are well studied [57], [73], where the computing task is distributed to n server nodes and the task can be completed with the results from the fastest k nodes. In [57], [73], the authors studied the optimal dimension k under exponential latency of each node. Moreover, flexible MDS codes are applied to the distributed computing problem in [102, 70, 101]. However, it is assumed that we know the set of available nodes before we start computing, which is not the case in our setup.

The chapter is organized as follows: In Section 3.2, we present the definition and the construction of our flexible storage codes. We present the flexible LRC, PMDS, and MSR codes in Sections 3.3.1, 3.3.2, and 3.3.3, respectively. In Section 3.4, we analyze the latency of data access using our flexible codes and compare it with those of fixed codes. The conclusion is made in Section 3.5.

Notation. For any integer $a \geq 1$, $[a]$ denotes the set $\{1, 2, \ldots, a\}$. For a matrix A over $\mathbb{F}$, let $rank(A)$ denote its rank. For a set of matrices $A_1, A_2, \ldots, A_n$ of size $x \times y$, denote $diag(A_1, A_2, \ldots, A_n)$ the corresponding diagonal matrix of size $nx \times ny$. For a finite field $\mathbb{F}$, denote by $\mathbb{F}^* = \mathbb{F}\backslash\{0\}$.

3.2 The Framework for Flexible Codes

In this section, we define flexible storage codes and provide the framework for flexible codes to convert a fixed (non-flexible) code construction into a flexible one. For ease of exposition, ideas are illustrated through flexible MDS code examples in this section. Other types of code constructions are shown in Section 3.3.

First, we define flexible storage codes. In our illustrations, the codeword is represented by an $\ell \times n$ array over $\mathbb{F}$, denoted as $C \in \left(\mathbb{F}^{\ell}\right)^{n}$, where n is called the code length, and ℓ is called the sub-packetization. Each column corresponds to a storage node. We choose some fixed integers $j \in [a], \ell_j \in [\ell]$, and *recovery thresholds* $R_j \in [n]$. Let the *decoding columns* $\mathcal{R}_j \subseteq [n]$ be a subset of R_j columns, and the *decoding rows* $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_{R_j} \subseteq [\ell]$ be subsets of rows each with size ℓ_j. Denote by $C\ |_{\mathcal{R}_j:\mathcal{I}_1,\mathcal{I}_2,\ldots,\mathcal{I}_{R_j}}$ the $\ell_j \times R_j$ subarray of C that takes the rows $\mathcal{I}_1$ in the first column of $\mathcal{R}_j$, the rows $\mathcal{I}_2$ in the second column of $\mathcal{R}_j, \ldots$, and the rows $\mathcal{I}_k$ in the last column of $\mathcal{R}_j$. The information will be reconstructed from this subarray. For flexible MDS codes, flexible MSR codes, and flexible PMDS codes, we have

$$R_j = k_j.$$

Notation. For the above types of codes, we simply omit the parameter R_j.

For flexible LRC codes, we require

$$R_j = k_j + \frac{k_j}{n - k_j} + 1,$$

since the minimum distance is lower bounded by $n - k_j - \frac{k_j}{n-k_j} + 2$ [33].

Definition 3. The (n, k, ℓ) flexible storage code is parameterized by (R_j, k_j, ℓ_j), $j \in [a]$, for some positive integer a, such that $k_j \ell_j = k\ell, 1 \leq j \leq a, k_1 > k_2 > ... > k_a = k, \ell_a = \ell$. It encodes $k\ell$ information symbols over a finite filed $\mathbb{F}$ into n nodes, each with ℓ symbols. The code satisfies the following reconstruction condition for all $j \in [a]$: from any R_j nodes, each node accesses a set of ℓ_j symbols, and we can reconstruct all the information symbols, for any $j \in [a]$. That is, the code is defined by

- an encoding function $\mathcal{E} : \left(\mathbb{F}^{\ell}\right)^{k} \to \left(\mathbb{F}^{\ell}\right)^{n}$,
- decoding functions $\mathcal{D}_{\mathcal{R}_j} : \left(\mathbb{F}^{\ell_j}\right)^{R_j} \to \left(\mathbb{F}^{\ell}\right)^{k}$, for all $\mathcal{R}_j \subseteq [n], |\mathcal{R}_j| = R_j$, and
- decoding rows $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_{R_j} \subseteq [\ell]$, $|\mathcal{I}_1| = |\mathcal{I}_2| = \cdots = |\mathcal{I}_{R_j}| = \ell_j$, which are dependent

on the choice of the decoding columns $\mathcal{R}_j$.

The functions are chosen such that any information $U \in \left(\mathbb{F}^{\ell}\right)^{k}$ can be reconstructed from the nodes in $\mathcal{R}_j$:

$$\mathcal{D}_{\mathcal{R}_j}\left(\mathcal{E}(U) \mid_{\mathcal{R}_j:\mathcal{I}_1,\mathcal{I}_2,\ldots,\mathcal{I}_{R_j}}\right) = U.$$

A *flexible MDS code* is defined as a flexible storage code as in Definition 3, such that $R_j = k_j$. We first examine the example in Fig. 3.1

Lemma 11. Fig. 3.1 is an $(n, k, \ell) = (4, 2, 3)$ flexible MDS code parameterized by $(k_j, \ell_j) \in \{(3, 2), (2, 3)\}$.

Proof: The encoding function is clear. We have encoded $k\ell = 6$ information symbols over $\mathbb{F}$ to a code with $n = 4, \ell = 3, k = 2$.

Then, we present the decoding. From any $k_1 = 3$ nodes, each node accesses the first $\ell_1 = 2$ symbols: The first 2 rows form a single parity-check $(4, 3, 2)$ MDS code, and thus we can easily get the information symbols from any 3 out of 4 symbols in each row. From any $k_2 = 2$ nodes, each node accesses all the $\ell_2 = 3$ symbols: We can first decode W_1' and W_2' in the last row since the last row is a $(4, 2, 1)$ MDS code. Then, $(C_{1,1}, C_{1,2}, C_{1,3}, W_1, W_1')$ and $(C_{2,1}, C_{2,2}, C_{2,3}, W_2, W_2')$ form two $(5, 3, 1)$ MDS codes. We can decode all the information symbols from W_1', W_2' and any 2 columns of the first 2 rows. ■

Code overview. The main idea of the general code construction is similar to that of Fig. 3.1. The construction is based on a set of $(n + k_j - k_a, k_j, \ell_j - \ell_{j-1})$ codes, each code called a *layer*, such that $k_j\ell_j = k\ell, j \in [a]$, $k_1 > k_2 > ...k_a = k, \ell_a = \ell, \ell_0 = 0$. The first layer is encoded from the original information symbols and other layers are encoded from the "extra parities". The intuition for the flexible reconstruction is that after accessing symbols from some layers, we can decode the corresponding information symbols, which is in turn extra parity symbols in an upper layer. Therefore, the decoder can afford accessing less codeword symbols in the upper layer, resulting in

a smaller recovery threshold.

Table 3.1: Construction of multiple-layer codes

Storage nodes				Extra parities				
$C_{1,1}$	$C_{1,2}$	$\cdots$	$C_{1,n}$	$C'_{1,1}$	$\cdots$	$\cdots$	$\cdots$	C'_{1,k_1-k_a}
$C_{2,1}$	$C_{2,2}$	$\cdots$	$C_{2,n}$	$C'_{2,1}$	$\cdots$	$\cdots$	C'_{2,k_2-k_a}	
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		
$C_{a-1,1}$	$C_{a-1,2}$	$\cdots$	$C_{a-1,n}$	$C'_{a-1,1}$	$\cdots$	$C'_{a-1,k_{a-1}-k_a}$		
$C_{a,1}$	$C_{a,2}$	$\cdots$	$C_{a,n}$					

Construction 1. In Table 3.1, we construct (n, k, ℓ) flexible storage codes parameterized by $\{(k_j, \ell_j) : 1 \leq j \leq a\}$, such that $k_j\ell_j = k\ell$, $k_1 > k_2 > ...k_a = k, \ell_a = \ell$.

Each column is a node. Note that only the first n columns under storage nodes are stored, and the extra parities are auxiliary. Set $\ell_0 = 0$. We have a layers, and Layer $j, j \in [a]$, is an $(n + k_j - k_a, k_j, \ell_j - \ell_{j-1})$ code

$$[C_{j,1}, C_{j,2}, \ldots, C_{j,n}, C'_{j,1}, C'_{j,2}, \ldots, C'_{j,k_j-k_a}],$$

where $C_{j,i} = [C_{j,1,i}, C_{j,2,i}, ..., C_{j,\ell_j-\ell_{j-1},i}]^T \in \mathbb{F}^{\ell_j-\ell_{j-1}}$, $i \in [n]$, are actually stored, and $C'_{j,i} = [C'_{j,1,i}, C'_{j,2,i}, ..., C'_{j,\ell_j-\ell_{j-1},i}]^T \in \mathbb{F}^{\ell_j-\ell_{j-1}}$, $i \in [k_j - k_a]$, are the auxiliary extra parities. The $(n + k_1 - k_a, k_1, \ell_1)$ code in the first layer is encoded from the $k_1\ell_1 = k\ell$ information symbols over $\mathbb{F}$, and the $(n + k_j - k_a, k_j, \ell_j - \ell_{j-1})$ code in Layer $j, j \geq 2$, is encoded from extra parities $C'_{j',i}$, for $j' \in [j], k_j - k_a + 1 \leq i \leq k_{j-1} - k_a$. As a sanity check, $\sum_{j'=1}^{j}(k_{j-1} - k_j)(\ell_{j'} - \ell_{j'-1}) = (k_{j-1} - k_j)(\ell_{j-1} - \ell_0) = k_j(\ell_j - \ell_{j-1})$ extra parities over $\mathbb{F}$ are encoded into Layer j, which matches the code dimension of that layer. Here we used $\ell_0 = 0$, and $k_{j-1}\ell_{j-1} = k_j\ell_j$.

Construction 1 can be applied to different kinds of codes. We start with MDS codes to show how to use Construction 1 with a family of storage codes. For an (n, k, ℓ) *flexible MDS code* parametrized by $\{(R_j, k_j, \ell_j) : 1 \leq j \leq a\}$ satisfying Definition 3, we have $R_j = k_j$. That is, we can recover the entire information from any k_j nodes, each node accessing its first ℓ_j symbols.

Theorem 9. With a set of $(n + k_j - k_a, k_j, \ell_j - \ell_{j-1}), j \in [a], \ell_0 = 0$ MDS codes over $\mathbb{F}$, Construction 1 is an (n, k, ℓ) flexible MDS code parametrized by $\{(R_j, k_j, \ell_j) : 1 \leq j \leq a\}$ satisfying Definition 3 and $R_j = k_j$.

Proof: Encoding: As described in Construction 1, we encode the $k\ell$ information symbols into an $(n+k_1-k_a, k_1, \ell_1)$ MDS code, and $(n+k_j-k_a, k_j, \ell_j-\ell_{j-1}), 2 \leq j \leq a$ MDS codes are encoded from the extra parities.

Decoding: Fix $j \in [a]$. Assume from any k_j nodes, each node accesses its first ℓ_j symbols over $\mathbb{F}$. We want to show that all the information symbols can be recovered.

We prove by induction that we are able to decode Layer 1, which contains all the information symbols.

Base case: For Layer $j' = j$, it is obvious since Layer j is an MDS code with dimension k_j.

Induction step: Suppose that Layers $j'+1, j'+2, ..., j$ are decoded. Then, for Layer j', as shown in Construction 1 from the decoded layers we get the $k_{j'} - k_j$ extra parities $C'_{j',i}, k_j - k_a + 1 \leq i \leq k_{j'} - k_a$. Together with the k_j nodes we have accessed in Layer j', we get enough dimensions to decode Layer j'. ■

We note that one can choose any family of MDS codes for the above theorem, e.g., Reed-Solomon codes [80], and vector codes [10]. In the case of vector codes, the codeword symbols of the MDS codes are from a vector space rather than a finite field.

3.3 Constructions

In this section, we show how to apply Construction 1 to LRC (locally recoverable) codes, PMDS (partial maximum distance separable) codes, and MSR (minimum storage regenerating) codes.

These codes provide a flexible reconstruction mechanism for the entire information, and either can reduce the single-failure repair cost, i.e., the number of helper nodes and the amount of transmitted information, or can tolerate mixed types of failures. Applications include failure protection in distributed storage systems and in solid-state drives.

3.3.1 Flexible LRC

An (n, k, ℓ, r) LRC code is defined as a code with length n, dimension k, and sub-packetization size ℓ. For any single node failure or erasure, there exists a group of at most r available nodes (called helpers) such that the failure can be recovered from them [33, 29, 32, 111, 9]. The minimum Hamming distance of an (n, k, ℓ, r) LRC code is lower bounded in [33] as

$$d_{\min} \geq n - k - \lceil \frac{k}{r} \rceil + 2, \tag{3.2}$$

and LRC codes achieving the bound are called optimal LRC codes. For simplicity, we use (n, k, r) LRC codes to present (n, k, ℓ, r) LRC codes with $\ell = 1$. Tamo and Barg [92] constructed a family of optimal (n, k, r) LRC codes that encode the k information symbols into $C = [C_{1,1}, C_{1,2}, \ldots, C_{1,r+1}, ..., C_{\frac{n}{r+1},1}, C_{\frac{n}{r+1},2}, ..., C_{\frac{n}{r+1},r+1}]$, where each group $\{C_{m,i} : i \in [r+1]\}, m \in [\frac{n}{r+1}]$, is an MDS code with dimension r and the whole code C has a minimum distance of $n - k - \frac{k}{r} + 2$, i.e., we can decode all the information symbols from any $k + \frac{k}{r} - 1$ nodes. If an optimal LRC code has the above structure with groups, we say it is an optimal LRC code *by groups*.

We define the (n, k, ℓ, r) ***flexible LRC code*** parameterized by $\{(R_j, k_j, \ell_j) : 1 \leq j \leq a\}$ as a flexible storage code as in Definition 3, such that all the symbols of any node can be recovered by reading at most r other nodes, and

$$R_j = k_j + \frac{k_j}{n - k_j} + 1.$$

The above R_j matches the minimum distance lower bound (3.2). As a result, our definition of flexible LRC code implies optimal minimum Hamming distance when we consider all symbols at each node.

Code overview. The flexible LRC code is based on Construction 1, where, first, *extra groups* are generated in each row. Then, r extra parities are chosen from each extra group and encoded into lower layers. During information reconstruction, extra parities and hence extra groups are recovered from lower layers, leading to a smaller number of required access.

Example 5. Table 3.2 shows an example of $(n = 12, k = 4, \ell = 3, r = 2)$ flexible LRC code. In this code, Rows 1 and 2 are $(n = 12, k = 6, r = 2)$ LRC codes encoded from the information, and 1 extra group is generated in each row. We take 4 extra parities from the extra groups, which are encoded into $(n = 12, k = 4, r = 2)$ LRC code in Row 3. In this example, we have 12 nodes and they are evenly divided into 4 groups. Any single failed node can be recovered from the other 2 nodes in the same group. To recover the entire information, we require either any 8 nodes, each accessing the first 2 symbols, or any 5 nodes, each accessing all 3 symbols. The details of this code are shown in Theorem 10 and Example 6.

Table 3.2: Construction of $(n = 12, k = 4, \ell = 3, r = 2)$ flexible LRC code

	group 1			$\cdots$	group 4		
Layer 1	$C_{1,1,1}$	$C_{1,1,2}$	$C_{1,1,3}$	$\cdots$	$C_{1,1,10}$	$C_{1,1,11}$	$C_{1,1,12}$
	$C_{1,2,1}$	$C_{1,2,2}$	$C_{1,2,3}$	$\cdots$	$C_{1,2,10}$	$C_{1,2,11}$	$C_{1,2,12}$
Layer 2	$C_{2,1,1}$	$C_{2,1,2}$	$C_{2,1,3}$	$\cdots$	$C_{2,1,10}$	$C_{2,1,11}$	$C_{2,1,12}$

In the following, we apply the optimal LRC codes by groups to Construction 1 and show how to construct an (n, k, ℓ, r) ***flexible LRC code*** parametrized by $\{(R_j, k_j, \ell_j) : 1 \leq j \leq a\}$ satisfying Definition 3. We assume n is divisible by $r + 1$ and all k_j's are divisible by r here. The code is defined in $\mathbb{F}$ of size at least $n + (k_1 - k_a)\frac{r+1}{r}$. The resulting code turns out to be an (n, k_j, ℓ_j, r) LRC code when ℓ_j symbols are accessed at each node. That is, for any single node failure, there exists a group of at most r helpers such that the failure can be recovered from them.

Theorem 10. Let n be divisible by $r+1$ and all $k_j, j \in [a]$ be divisible by r. With a set of $(n+(k_j-k_a)\frac{r+1}{r}, k_j, r), j \in [a], \ell_0 = 0$ optimal LRC codes by groups over $\mathbb{F}$, Construction 1 results in the flexible LRC codes with locality r and $\{(R_j, k_j, \ell_j) : 1 \leq j \leq a\}$ satisfying Definition 3.

Proof: Encoding: In Layer j, we apply an $(n+(k_j-k_a)\frac{r+1}{r}, k_j, r), j \in [a], \ell_0 = 0$ optimal LRC code to each row. As described in Construction 1, we encode the $k\ell$ information symbols in the ℓ_1 rows of Layer 1, and the remaining rows are encoded from the extra parities.

Next, we show how to choose the n stored symbols and the $k_j - k_a$ extra parities in each row. In the $(n+(k_j-k_a)\frac{r+1}{r}, k_j, r)$ LRC code, we have $\frac{n}{r+1} + \frac{k_j-k_a}{r}$ groups. We first pick $\frac{n}{r+1}$ groups, containing n symbols, as the stored symbols. Thus, the n stored symbols in each row form an $(n, k_j, r), j \in [a], \ell_0 = 0$ optimal LRC code. Then, in the remaining $\frac{k_j-k_a}{r}$ groups, we pick r nodes in each group, which contains $k_j - k_a$ nodes, as the extra parities.

Decoding: Since all the information symbols are encoded in Layer 1, we can decode the information symbols if we get enough dimensions to decode Layer 1.

We prove by induction that we can decode all information symbols from any $R_j = k_j + \frac{k_j}{r} - 1, j \in [a]$ nodes, each node accesses the first ℓ_j symbols.

Base case: From Layer j, since each row of it is part of the $(n+(k_j-k_a)\frac{r+1}{r}, k_j, r)$ optimal LRC code, we can decode this layer from R_j nodes by the property of the optimal LRC codes.

Induction step: Let $1 < j' \leq j$ be given and suppose that Layers $j', j'+1, ..., j$ are decoded. From Construction 1, we know that all the extra parities in Layer $j'-1$ are included as the information symbols in Layers $j', j'+1, ..., j$ and are decoded. Also, we know from the encoding part that the extra parities in Layer $j'-1$ consist of the r parity symbols in each group of the $(n+(k_{j'-1}-k_a)\frac{r+1}{r}, k_{j'-1}, r)$ optimal LRC codes. Thus, according to the locality, the remaining symbol in all $\frac{k_{j'-1}-k_j}{r}$ groups in each row can be reconstructed. Therefore, we get additional $(k_{j'-1}-k_j)\frac{r+1}{r}$

symbols in each row of Layer $j'-1$ from the extra parities. Together with the R_j nodes we accessed in each row of Layer $j'-1$, we get $R_{j'-1}$ symbols and, we are able to decode Layer $j'-1$.

Locality: Since each row is encoded as a LRC code with locality r, every layer and the entire code also have locality r.

The proof is completed. ■

Example 6. We set $(n,k,l,r)=(12,4,3,2)$, $(R_1,k_1,\ell_1)=(8,6,2)$, $(R_2,k_2,\ell_2)=(5,4,2)$. The code is defined over $\mathbb{F}=GF(2^4)=\{0,1,\alpha,...,\alpha^{14}\}$, where α is a primitive element of the field. Totally we have $k\ell=12$ information symbols and we assume they are $u_{1,0},u_{1,1},...,u_{1,5}$, $u_{2,0},u_{2,1},...,u_{2,5}$. The example is based on the optimal LRC code constructions in [92].

The construction is shown below, each column is a node with 3 symbols:

$$\begin{bmatrix} C_{1,1,1} & C_{1,1,2} & \cdots & C_{1,1,12} \\ C_{1,2,1} & C_{1,2,2} & \cdots & C_{1,2,12} \\ C_{2,1,1} & C_{2,1,2} & \cdots & C_{2,1,12} \end{bmatrix}, \tag{3.3}$$

where every entry in Row m will be constructed as $f_m(x)$ for some polynomial $f_m(\cdot)$ and some field element x as below, $m=1,2,3$.

The evaluation points are divided into 4 groups as $A=\{A_1=\{1,\alpha^5,\alpha^{10}\},A_2=\{\alpha,\alpha^6,\alpha^{11}\}$, $A_3=\{\alpha^2,\alpha^7,\alpha^{12}\}$, $A_4=\{\alpha^3,\alpha^8,\alpha^{13}\}\}$. We also set $A_5=\{\alpha^4,\alpha^9,\alpha^{14}\}$ as the evaluation points group for the extra parities.

According to [92], we define $g(x)=x^3$, and one can check $g(x)$ is a constant for each group A_i,

$i \in [5]$. Then, the first 2 rows are encoded with

$$f_m(x) = \big(u_{m,0} + u_{m,1}g(x) + u_{m,2}g^2(x)\big) + x\big(u_{m,3} + u_{m,4}g(x) + u_{m,5}g^2(x)\big), m = 1, 2. \tag{3.4}$$

The last row is encoded with

$$f_3(x) = \big(f_1(\alpha^4) + f_1(\alpha^9)g(x)\big) + x\big(f_2(\alpha^4) + f_2(\alpha^9)g(x)\big). \tag{3.5}$$

Since $g(x)$ is a constant for each group, $f_m(x), m \in [3]$ can be viewed as a polynomial of degree 2. Any single failure can be recovered from the other 2 available nodes evaluated by the points in the same group. The locality $r = 2$ is achieved.

Noticing that $f_1(x)$ and $f_2(x)$ are polynomials of degree 7, all information symbols can be reconstructed from the first $\ell_1 = 2$ rows of any $R_1 = 8$ available nodes.

Moreover, $f_3(x)$ has degree 4. With $R_2 = 5$ available nodes, we can first decode $f_1(\alpha^4)$, $f_1(\alpha^9)$, $f_2(\alpha^4)$, $f_2(\alpha^9)$ in row 3. Then, $f_1(\alpha^{14})$, $f_2(\alpha^{14})$ can be decoded due to the locality $r = 2$. At last, together with the 5 other evaluations of $f_1(x)$ and $f_2(x)$ obtained in Rows 1 and 2, we are able to decode all information symbols.

3.3.2 Flexible PMDS codes

PMDS codes are first introduced in [11] to overcome mixed types of failures in Redundant Arrays of Independent Disks (RAID) systems using solid-state drives (SSDs). A code consisting of an $\ell \times n$ array is an (n, k, ℓ, s) PMDS code if it can tolerate $n - k$ node or column failures and s additional arbitrary symbol failures in the code.

Let $\ell_0 = 0$ and $\{(k_j, \ell_j) : 1 \le j \le a\}$ satisfy (3.1). We define an (n, k, ℓ, s) *flexible PMDS code*

parameterized by $\{(k_j, \ell_j) : 1 \leq j \leq a\}$ such that any row in $[\ell_{j-1}+1, \ell_j]$ is an (n, k_j) MDS code, and from the first ℓ_j rows, we can reconstruct the entire information if there are up to $n - k_j$ node failures and up to s additional arbitrary symbol failures, $1 \leq j \leq a$. As mentioned, for PMDS codes, $R_j = k_j$. Note that different from Definition 3, the number of information symbols for a flexible PMDS code is at most $k\ell - s \triangleq K$.

Example 7. Consider the example of a $(5, 3, 4, 2)$ flexible PMDS code with $\{(k_1, \ell_1), (k_2, \ell_2)\} = \{(4, 3), (3, 4)\}$ in Table 3.3. If we only have "$*$" as failures, we can use the first 4 nodes to decode, each node accessing the first 3 symbols. If both "$*$" and "$\triangle$" are failures, we can decode from Nodes $1, 3, 4$, each node accessing 4 symbols. In both cases, the remaining $K = k\ell - s = 10$ symbols are independent and sufficient to reconstruct the entire information. The details of the encoding and decoding for this construction are presented in Theorem 11.

Table 3.3: An example of $(5, 3, 4, 2)$ flexible PMDS code.

$C_{1,1,1}$	$\triangle$	$C_{1,1,3}$	$*$	$*$
$C_{1,2,1}$	$\triangle$	$C_{1,2,3}$	$C_{1,2,4}$	$*$
$C_{1,3,1}$	$\triangle$	$*$	$C_{1,3,4}$	$*$
$C_{2,1,1}$	$\triangle$	$C_{2,1,3}$	$C_{2,1,4}$	$*$

A general construction of PMDS codes is proposed in [13] for any k and s using Gabidulin codes. In this section, we first introduce the construction in [13] and then show how to apply it to flexible PMDS codes.

Code overview. To tolerate additional symbol failures, the fixed PMDS code in [13] uses Gabidulin code to encode the information into auxiliary symbols, which are evenly allocated to each row. Then, an MDS code is applied to the auxiliary symbols in each row, ensuring the protection against column failures. Our flexible PMDS code encodes the information using Gabidulin code into auxiliary symbols, which are allocated to each layer according to k_j. MDS codes with different dimensions are then applied to each row, thus ensuring flexible information reconstruction.

An (N, K) Gabidulin code over the finite field $\mathbb{F} = GF(q^L), L \geq N$ is defined by the polynomial

$f(x) = \sum_{i=0}^{K-1} u_i x^{q^i}$, where $u_i \in \mathbb{F}, i = 0, 1, ..., K-1$ is the information symbol. The N codeword symbols are $f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_N)$ where the N evaluation points $\{\alpha_1, ..., \alpha_N\}$ are linearly independent over $GF(q)$. From any K independent evaluation points over $GF(q)$, the information can be recovered.

In [13, Construction 1], the (n, k, ℓ, s) codeword is an $\ell \times n$ matrix over $\mathbb{F} = GF(q^{k\ell})$ shown below:

$$\begin{bmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,n} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ C_{\ell,1} & C_{\ell,2} & \cdots & C_{\ell,n} \end{bmatrix}, \tag{3.6}$$

where each column is a node. Set $K = \ell k - s$. Here, $C_{m,i} \in \mathbb{F}, m \in [\ell], i \in [k]$ are the $K + s$ codeword symbols from a $(K + s, K)$ Gabidulin code, and for each row m, $m \in [\ell]$,

$$[C_{m,k+1}, ..., C_{m,n}] = [C_{m,1}, ..., C_{m,k}] G_{\text{MDS}}, \tag{3.7}$$

where G_{MDS} is the $k \times (n - k)$ encoding matrix of an (n, k) systematic MDS code over $GF(q)$ that generates the parity.

It is proved in [13, Lemma 2] that t_m symbols in row $m, m \in [\ell]$, is equivalent to evaluations of $f(x)$ with $\sum_{m=1}^{\ell} \min(t_m, k)$ evaluation points that are linearly independent over $GF(q)$. Thus, with any $n - k$ node failures and s symbol failures, we have $t_m \leq k$ and

$$\sum_{m=1}^{\ell} \min(t_m, k) = \sum_{m=1}^{\ell} t_m = \ell k - s = K. \tag{3.8}$$

Then, with the K linearly independent evaluations of $f(x)$, we can decode all information symbols.

Next, we show how to construct flexible PMDS codes. Rather than generating extra parities as in

Construction 1, the main idea here is that we divide our code into multiple layers, and each layer applies a construction similar to that of (3.6) with a different dimension.

Theorem 11. We can construct an (n, k, ℓ, s) flexible PMDS code over $GF(q^N)$ parameterized by $\{(k_j, \ell_j) : 1 \leq j \leq a\}$ satisfying (3.1), with an (N, K) Gabidulin code over $GF(q^N)$, $N = \sum_{j=1}^{a} k_j(\ell_j - \ell_{j-1})$, $K = \ell k - s$, and a set of (n, k_j) systematic MDS codes over $GF(q)$.

Proof: Encoding: Denote $C_{j,m_j,i}$ the symbol in the m_j-th row of Layer j, and in the i-th node, $j \in [a], m_j \in [\ell_j - \ell_{j-1}], i \in [n]$. We first encode the K information symbols using the (N, K) Gabidulin code. Then, we set the first k_j codeword symbols in each row: $C_{j,m_j,i}, j \in [a], m_j \in [\ell_j - \ell_{j-1}], i \in [k_j]$, as the codeword symbols in the (N, K) Gabidulin code. The remaining $n - k_j$ codeword symbols in each row are

$$[C_{j,m_j,k_j+1}, ..., C_{j,m_j,n}] = [C_{j,m_j,1}, ..., C_{j,m_j,k_j}]G_{n,k_j},$$

where G_{n,k_j} is the encoding matrix (to generate the parity check symbols) of the (n, k_j) systematic MDS code over $GF(q)$.

Decoding: For $n - k_J$ failures, we access the first ℓ_J rows (the first J layers) from each node. The code structure in each layer is similar to the general PMDS code in [13, Construction 1], from [13, Lemma 2] we know that for a union of t_{m_j} symbols in Row m_j of Layer j, $j \leq J$, they are equivalent to evaluations of $f(x)$ with $\sum_{j=1}^{J} \sum_{m_j=1}^{\ell_j - \ell_{j-1}} \min(t_{m_j}, k_j)$ linearly independent points over $GF(q)$ in $GF(q^N)$. Thus, with $n - k_J$ node failures and s symbol failures, we have $t_{m_j} \leq k_J \leq k_j$ for $j \in [J]$, and

$$\sum_{j=1}^{J} \sum_{m_j=1}^{\ell_j - \ell_{j-1}} \min(t_{m_j}, k_j) = \sum_{j=1}^{J} \sum_{m_j=1}^{\ell_j - \ell_{j-1}} t_{m_j} = \ell_J k_J - s = K.$$

Then, the information symbols can be decoded from K linearly independent evaluations of $f(x)$.

■

3.3.3 Flexible MSR codes

In this section, we study flexible MSR codes. In the following, the number of parity nodes is denoted by $r = n - k$ [1]. The *repair bandwidth* is defined as the amount of transmission required to repair a single node erasure, or failure, from all remaining nodes (called helper nodes), normalized by the size of the node. For an (n, k) MDS code, the repair bandwidth is bounded by the minimum storage regenerating (MSR) bound [20] as

$$b \geq \frac{n-1}{n-k}. \tag{3.9}$$

An MDS code achieving the MSR bound is called an MSR code. MSR vector codes are well studied in [104, 77, 71, 93, 99, 78, 34, 105], where each symbol is a vector. As one of the most popular codes in practical systems, Reed-Solomon (RS) code and its repair is studied in [94, 40, 17, 106, 61], where each symbol is a scalar.

We have shown in Theorem 9 that using a set of MDS codes, Construction 1 can recover the information symbols by any pair (k_j, ℓ_j), which means that for the first ℓ_j symbols in each node, our code is an (n, k_j, ℓ_j) MDS code. In addition, we require the optimal repair bandwidth property for *flexible MSR codes*. A *flexible MSR code* is defined to be a flexible storage code as in Definition 3, such that $R_j = k_j$, and a single node failure is recovered using a repair bandwidth satisfying the MSR bound (2.1).

Code overview. Our codes in this section are similar to Construction 1, with additional restrictions on the parity check matrices and the extra parities. The key point here is that the extra parities and the information symbols in lower layers are exactly the same and they also share the same parity check sub-matrix. To repair the failed symbol with smallest bandwidth, the extra parities are viewed as additional helpers and the required information can be obtained *for free* from the repair of the lower layers.

[1] Notice that r was used for a different meaning (locality) in LRC codes.

We will first show an illustrating example with 2 layers and then present our constructions based on vector and scalar MSR codes, respectively.

Example 8. We construct an $(n, k, \ell) = (4, 2, 3)$ flexible MSR code parameterized by $(k_1, \ell_1) = (3, 2)$ and $(k_2, \ell_2) = (2, 3)$. The reconstruction of the entire information and the repair bandwidth are proved in Lemma 12.

Let $\mathbb{F} = GF(2^2) = \{0, 1, \beta, \beta^2 = 1 + \beta\}$, where β is a primitive element of $GF(2^2)$. Our construction is based on the following $(4, 2, 2)$ MSR vector code over $\mathbb{F}^2$ with parity check matrix

$$H = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & h_{1,4} \\ h_{2,1} & h_{2,2} & h_{2,3} & h_{2,4} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.10}$$

where each $h_{i,j}$ is a 2×2 matrix over $\mathbb{F}$. Namely, a codeword symbol c_i is in $\mathbb{F}^2$, $i = 1, 2, 3, 4$, and the codeword $[c_1^T, c_2^T, c_3^T, c_4^T]^T \in (\mathbb{F}^2)^4$ is in the null space of H. One can check that it is a $(4, 2)$ MDS code, i.e., any two codeword symbols suffice to reconstruct the entire information. The repair matrix is defined as

$$S_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, S_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, S_3 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}, S_4 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.11}$$

It is easy to check that

$$rank\left(S_* \begin{bmatrix} h_{1,i} \\ h_{2,i} \end{bmatrix} \right) = \begin{cases} 2, i = * \\ 1, i \neq * \end{cases}. \tag{3.12}$$

When node $* \in \{1, 2, 3, 4\}$ fails, we can repair node c_* by equations $S_* \times H \times [c_1^T, c_2^T, c_3^T, c_4^T]^T = 0$.

In particular, helper i, $i \neq *$, transmits

$$S_* \begin{bmatrix} h_{1,i} \\ h_{2,i} \end{bmatrix} c_i,$$

which is 1 symbol in $\mathbb{F}$, achieving an optimal total repair bandwidth of 3 symbols in $\mathbb{F}$.

For our flexible MSR code, every entry in the code array is a vector in $\mathbb{F}^2$. The code array is shown as below, each column being a node:

$$\begin{bmatrix} C_{1,1,1} & C_{1,1,2} & C_{1,1,3} & C_{1,1,4} \\ C_{1,2,1} & C_{1,2,2} & C_{1,2,3} & C_{1,2,4} \\ C_{2,1,1} & C_{2,1,2} & C_{2,1,3} & C_{2,1,4} \end{bmatrix}. \tag{3.13}$$

The code has 2 layers, where $C_{1,m_1,i} \in \mathbb{F}^2$ are in Layer 1 and $C_{2,m_2,i}$ are in Layer 2 with $m_1 = 1, 2, m_2 = 1, i \in [4]$. Each $C_{j,m_j,i}$ is the vector $[c_{j,m_j,i,1}, c_{j,m_j,i,2}]^T$ with elements in $\mathbb{F}$. The code totally contains 48 bits with 24 information bits, and each node contains 12 bits. We define the code with the 3 parity check matrices shown below. Let

$$H_1 = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & h_{1,4} & h_{1,1} \\ h_{2,1} & h_{2,2} & h_{2,3} & h_{2,4} & \beta h_{2,1} \end{bmatrix}, \tag{3.14}$$

$$H_2 = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & h_{1,4} & h_{1,2} \\ h_{2,1} & h_{2,2} & h_{2,3} & h_{2,4} & \beta h_{2,2} \end{bmatrix}, \tag{3.15}$$

$$H_3 = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & h_{1,4} \\ \beta h_{2,1} & \beta h_{2,2} & h_{2,3} & h_{2,4} \end{bmatrix}. \tag{3.16}$$

The code is defined by

$$H_1 \times [C_{1,1,1}^T, C_{1,1,2}^T, C_{1,1,3}^T, C_{1,1,4}^T, C_{2,1,1}^T]^T = 0, \tag{3.17}$$

$$H_2 \times [C_{1,2,1}^T, C_{1,2,2}^T, C_{1,2,3}^T, C_{1,2,4}^T, C_{2,1,2}^T]^T = 0, \tag{3.18}$$

$$H_3 \times [C_{2,1,1}^T, C_{2,1,2}^T, C_{2,1,3}^T, C_{2,1,4}^T]^T = 0. \tag{3.19}$$

Lemma 12. Example 8 is an $(n, k, \ell) = (4, 2, 3)$ flexible MSR code parameterized by $(k_j, \ell_j) \in \{(3, 2), (2, 3)\}$.

Proof: It is easy to check that the code defined by H_1 or H_2 is an $(5, 2)$ MDS code, and H_3 defines an $(4, 2)$ MDS code. Thus, the construction in Example 8 is the same as Construction 1, and the flexible reconstruction of the entire information is shown in Theorem 9.

Let $* \in \{1, 2, 3, 4\}$ be the index of the failed node. For the repair, we first note that

$$rank\left(S_* \begin{bmatrix} h_{1,i} \\ h_{2,i} \end{bmatrix}\right) = rank\left(S_* \begin{bmatrix} h_{1,i} \\ \beta h_{2,i} \end{bmatrix}\right) = \begin{cases} 2, i = * \\ 1, i \neq * \end{cases}. \tag{3.20}$$

for $i = 1, 2$.

Then, we use the same repair matrix S_* in (3.11) to repair the failed node $*$:

$$S_* \times H_1 \times [C_{1,1,1}^T, C_{1,1,2}^T, C_{1,1,3}^T, C_{1,1,4}^T, C_{2,1,1}^T]^T = 0, \tag{3.21}$$

$$S_* \times H_2 \times [C_{1,2,1}^T, C_{1,2,2}^T, C_{1,2,3}^T, C_{1,2,4}^T, C_{2,1,2}^T]^T = 0, \tag{3.22}$$

$$S_* \times H_3 \times [C_{2,1,1}^T, C_{2,1,2}^T, C_{2,1,3}^T, C_{2,1,4}^T]^T = 0. \tag{3.23}$$

For helper $i \in [4]$, $i \neq *$, it transmits

$$S_* \begin{bmatrix} h_{1,i} \\ h_{2,i} \end{bmatrix} C_{1,1,i}, \tag{3.24}$$

$$S_* \begin{bmatrix} h_{1,i} \\ h_{2,i} \end{bmatrix} C_{1,2,i}, \tag{3.25}$$

$$S_* \begin{bmatrix} h_{1,i} \\ \overline{\beta} h_{2,i} \end{bmatrix} C_{2,1,i}, \tag{3.26}$$

where $\overline{\beta} = \beta$ if $i = 1, 2$ and $\overline{\beta} = 1$ if $i = 3, 4$. Note that to repair the failed node, in Equations (3.21) and (3.22), we also require $S_* \begin{bmatrix} h_{1,1} \\ \beta h_{2,1} \end{bmatrix} C_{2,1,1}$ and $S_* \begin{bmatrix} h_{1,2} \\ \beta h_{2,2} \end{bmatrix} C_{2,1,2}$, which can be either obtained from (3.26) or solved from Equation (3.23).

Then, from (3.12) and (3.20) we have that for any failed node, we only need 1 symbol from each of the remaining $C_{j,m_j,i}$, which meets the MSR bound. ■

Remark. Notice that in this example, we do not require the codes in the first layer defined by (3.14) and (3.15) to be MSR codes, thus resulting in a smaller field. However, the rank condition (3.20) guarantees the optimal repair bandwidth for the entire code. Also, in our general constructions, we do not require the codes in Layers 1 to $a - 1$ to be MSR codes.

In the following, we show that by applying Construction 1 to the vector MSR code [104] and the RS MSR code [94], we can construct flexible MSR codes.

Flexible MSR codes with parity check matrices

Below we present codes defined by parity check matrices similar to Example 8. We show in Theorem 12 that with certain choices of the parity check matrices, one obtains a flexible MSR

code.

Construction 2. The code is defined in some $\mathbb{F}^L$ parameterized by $(k_j, \ell_j), j \in [a]$ such that $k_j\ell_j = k\ell$, $k_1 > k_2 > ...k_a = k, \ell_a = \ell$. We define the parity check matrix for the m_j-th row in Layer $j \in [a]$ as:

$$H_{j,m_j} = \begin{bmatrix} h_{j,m_j,1} & \cdots & h_{j,m_j,n} & g_{j,m_j,1} & \cdots & g_{j,m_j,k_j-k_a} \end{bmatrix}, \tag{3.27}$$

where each $h_{j,m_j,i}, g_{j,m_j,i}$ is an $rL \times L$ matrix with elements in $\mathbb{F}$. The $(n + k_j - k_a, k_j)$ MDS code in the m_j-th row of Layer j is defined by

$$H_{j,m_j} \times [{C_{j,m_j,1}}^T, {C_{j,m_j,2}}^T, \cdots, {C_{j,m_j,n}}^T, {C'_{j,m_j,1}}^T, \cdots, {C'_{j,m_j,k_j-k_a}}^T]^T = 0, \tag{3.28}$$

where $C_{j,m_j,i}$ are the stored codeword symbols and $C'_{j,m_j,i}$ are the extra parities. In this construction, when we encode the extra parities into lower layers, we set the codeword symbols and the corresponding parity check matrix entries exactly the same. Specifically, for Layers $j < j' \leq a$, we set

$$g_{j,x,y} = h_{j',x',y'}, \tag{3.29}$$

$$C'_{j,x,y} = C_{j',x',y'}. \tag{3.30}$$

Here, for $x \in [l_j - l_{j-1}]$, $k_{j'} - k_a + 1 \leq y \leq k_{j'-1} - k_a$, we have $g_{j,x,y}$ corresponds to $h_{j',x',y'}$ in Layer j', and

$$x' = \lfloor \frac{x(k_{j'-1} - k_{j'}) + y}{k_{j'}} \rfloor, \tag{3.31}$$

$$y' = (x(k_{j'-1} - k_{j'}) + y) \bmod k_{j'}, \tag{3.32}$$

where "mod" denotes the modulo operation.

For instance, in Example 8, the 2 extra parities in Layer 1 are exactly the same as the first 2 symbols in Layer 2 with $C'_{1,1,1} = C_{2,1,1}, g_{1,1,1} = h_{2,1,1}$ and $C'_{1,2,1} = C_{2,1,2}, g_{1,2,1} = h_{2,1,2}$.

Theorem 12. Assume the parity check matrices of Construction 2 in (3.27) satisfy

1). [MDS condition.] The codes defined by (3.27) are $(n + k_j - k_a, k_j)$ MDS codes.

2). [Rank condition.] The same repair matrices $S_*, * \in [n]$ can be used for every parity check matrix such that

$$rank(S_* h_{j,m_j,i}) = \begin{cases} L, i = * \\ \frac{L}{r}, i \neq * \end{cases}, \ i \in [n]. \tag{3.33}$$

Then, the code defined by Construction 2 is a flexible MSR code.

Proof: 1). If the MDS property is satisfied, Construction 2 is the same as Construction 1 by defining the MDS codes with parity check matrices. The flexible reconstruction of the entire information is presented in Theorem 9.

2). For repair, assume node $*, * \in [n]$ is failed. We use the repair matrix S_* in each row to repair it:

$$S_* \times H_{j,m_j} \times [{C_{j,m_j,1}}^T, {C_{j,m_j,2}}^T, \cdots, {C_{j,m_j,n}}^T, {C'_{j,m_j,1}}^T, \cdots, {C'_{j,m_j,k_j-k_a}}^T]^T = 0. \tag{3.34}$$

Notice that $C'_{j,m_j,1}, \cdots, C'_{j,m_j,k_j-k_a}$ are also the information symbols in the lower layers with the same corresponding parity check sub-matrices and can be retrieved from the lower layers. Thus, the failed node can be repaired from $n - 1$ helpers.

Clearly from (3.33), we only need L/r symbols from each helper and the optimal repair bandwidth is achieved. ■

We will now take Ye and Barg's construction [104] to show how to construct the flexible MSR codes satisfying conditions in Theorem 12. The code structure in one row is similar to [37].

Assume the field size $|\mathbb{E}| > rn$ and $\lambda_{i,j} \in \mathbb{E}, i \in [n], j = 0, 1, ..., r-1$ are rn distinct elements. The parity check matrix for the (n, k) MSR code in [104] can be represented as:

$$H = \begin{bmatrix} I & I & \cdots & I \\ A_1 & A_2 & \cdots & A_n \\ \vdots & \vdots & \ddots & \vdots \\ A_1^{r-1} & A_2^{r-1} & \cdots & A_n^{r-1} \end{bmatrix}, \tag{3.35}$$

where I is the $L \times L$ identity matrix and $A_i = \sum_{z=0}^{L-1} \lambda_{i,z_i} e_z {e_z}^T$. e_z is a vector of length $L = r^n$ with all elements 0 except the z-th element which is equal to 1. We write the r-ary expansion of z as $z = (z_n z_{n-1} \dots z_1)$, where $0 \leq z_i \leq r-1$ is the i-th digit from the right and $z = \sum_{i=0}^{r-1} z_i r^i$. Clearly, A_i is an $L \times L$ diagonal matrix with elements λ_{i,z_i}. The $L \times rL$ repair matrix $S_*, * \in [n]$ are also defined in [104] and [37, Sec. IV-A]:

$$S_* = \text{Diag}(D_*, D_*, ..., D_*) \tag{3.36}$$

with $\frac{L}{r} \times L$ matrix D_*, and it is shown that

$$rank\left(S_* \begin{bmatrix} I \\ A_i \\ \vdots \\ A_i^{r-1} \end{bmatrix} \right) = rank \begin{pmatrix} D_* \\ D_* A_i \\ \vdots \\ D_* A_i^{r-1} \end{pmatrix} = \begin{cases} L, i = * \\ \frac{L}{r}, i \neq * \end{cases}. \tag{3.37}$$

Here, for $0 \leq x \leq r^{n-1} - 1, 0 \leq y \leq r^n - 1$, the (x, y)-th entry of D_* equals 1 if the r-ary expansion of x and y satisfies $(x_{n-1}, x_{n-1}, \dots, x_1) = (y_n, y_{n-1}, \dots, y_{i+1}, y_{i-1}, \dots, y_1)$, and otherwise it equals 0.

Consider an extended field $\mathbb{F}$ from $\mathbb{E}$ and denote $\mathbb{F}^* \triangleq \mathbb{F}\backslash\{0\}$, $\mathbb{E}^* \triangleq \mathbb{E}\backslash\{0\}$. Then $\mathbb{F}^*$ can be partitioned to $t \triangleq \frac{|\mathbb{F}^*|}{|\mathbb{E}^*|}$ cosets: $\{\beta_1\mathbb{E}^*, \beta_2\mathbb{E}^*, ..., \beta_t\mathbb{E}^*\}$, for some elements $\beta_1, \beta_2, \ldots, \beta_t$ in $\mathbb{F}$ [61, Lemma 1]. Now, we define for the storage nodes (the first n nodes)

$$h_{j,m_j,i} = \begin{bmatrix} I \\ \beta_{j,m_j} A_i \\ \beta_{j,m_j}^2 A_i^2 \\ \vdots \\ \beta_{j,m_j}^{r-1} A_i^{r-1} \end{bmatrix}, \tag{3.38}$$

where β_{j,m_j} is chosen from $\{\beta_1, \beta_2, \ldots, \beta_t\}$. We say β_{j,m_j} is the *additional coefficient*. Then, the extra parity entries $g_{j,m_j,i}$ can be obtained accordingly from (3.31) and (3.32). Also, notice that A_i might show in H_{j,m_j} several times since the extra parity matrices are the same as the information symbols in lower layers. We choose the additional coefficients as below.

Condition 1. In each H_{j,m_j}, the additional coefficients for the same A_i are distinct.

Corollary 3. With parity check matrices defined by (3.38) and Condition 1, Construction 2 is a flexible MSR code.

Proof: We will prove the construction is flexible MSR using Theorem 12, for any given $j \in [a], m_j \in [k_j - k_a]$.

1) [MDS condition.] For the codeword $(c_1^T, c_2^T, ..., c_{n+k_j-k_a}^T)$ defined by the parity check matrix H_{j,m_j}, we write each codeword symbol as $c_i = (c_{i,1}, c_{i,2}, ..., c_{i,L})^T$. Since A_i is a diagonal matrix, for any $z = 0, 1, ..., L-1$, we have

$$\begin{bmatrix} 1 & \cdots & 1 & 1 & \cdots & 1 \\ \beta_{j,m_j}\lambda_{1,z_1} & \cdots & \beta_{j,m_j}\lambda_{n,z_n} & \alpha_1\gamma_1 & \cdots & \alpha_{k_j-k_a}\gamma_{k_j-k_a} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ (\beta_{j,m_j}\lambda_{1,z_1})^{r-1} & \cdots & (\beta_{j,m_j}\lambda_{n,z_n})^{r-1} & (\alpha_1\gamma_1)^{r-1} & \cdots & (\alpha_{k_j-k_a}\gamma_{k_j-k_a})^{r-1} \end{bmatrix} \begin{bmatrix} c_{1,z} \\ c_{2,z} \\ \vdots \\ c_{n+k_j-k_a,z} \end{bmatrix} = 0. \tag{3.39}$$

Here, $\beta_{j,m_j}, \alpha_1, \alpha_2, ..., \alpha_{k_j-k_a}$ are additional coefficients satisfying Condition 1. For $y \in [k_j - k_a]$, denote $\gamma_y \triangleq \lambda_{y',z_{y'}}$, corresponding to $g_{j,m_j,y} = h_{j',x',y'}$, where x', y' are computed from (3.31) and (3.32) with $x = m_j$. Next, we show (3.39) corresponds to a Vandermonde matrix, i.e., $(c_{1,z}, c_{2,z}, ..., c_{n+k_j-k_a,z})^T$ forms an $(n + k_j - k_a, k_j)$ Reed-Solomon code. Consider two entries in the second row of the $r \times (n + k_j - k_a)$ matrix in (3.39). Notice that each entry is the product of an additional coefficient and a λ variable (or a γ variable). There are three cases. 1) If the λ or the γ values are identical, by Condition 1, their additional coefficients differ. So, these two entries are distinct. 2) If the λ or the γ values are distinct, and the additional coefficients are identical, then the two entries are distinct. 3) The λ or the γ values are distinct, and the additional coefficients are distinct. Noticing λ and γ belong to $\mathbb{E}^*$, distinct additional coefficients implies that the two entries are in distinct cosets.

After we combine all $z = 0, 1, \ldots, L-1$ together, $(c_1^T, c_2^T, ..., c_{n+k_j-k_a}^T)^T$ is an $(n + k_j - k_a, k_j)$ MDS vector code.

2) [Rank condition.] Multiplying the row of a matrix by a constant does not change the rank. So, by (3.37) and (3.38),

$$rank(S_* h_{j,m_j,i}) = rank \begin{pmatrix} D_* \\ D_*\beta A_i \\ \vdots \\ D_*\beta^{r-1}A_i^{r-1} \end{pmatrix} = rank \begin{pmatrix} D_* \\ D_* A_i \\ \vdots \\ D_* A_i^{r-1} \end{pmatrix} = \begin{cases} L, i = * \\ \frac{L}{r}, i \neq * \end{cases}. \tag{3.40}$$

Since the code satisfies the above two conditions, using Theorem 12, it is a flexible MSR code. ■

To calculate the required field size, we study how many additional coefficients are required for our flexible MSR codes satisfying Condition 1. In the following, we propose 2 possible coefficient assignments. It should be noticed that one might find better assignments with smaller field sizes.

The simplest coefficient assignment assigns different additional coefficients to different rows, i.e., β_{j,m_j} to Row m_j in Layer j for the storage nodes (the first n nodes). By doing so, the parity check matrix $\beta_{j,m_j}A_i, j \in [a], m_j \in [\ell_j - \ell_{j-1}]i \in [n]$ will show at most twice in Construction 2, i.e., in Layer j corresponding to storage Node i, and in Layer j' corresponding to an extra parity, for some $j > j'$. Hence, the same A_i will correspond to different additional coefficients in the same row and Condition 1 is satisfied. In this case, we need a field size of $\ell|\mathbb{E}|$.

In the second assignment, we assign different additional coefficients in different layers for the storage nodes (the first n nodes), but for different rows in the same layer, we might use the same additional coefficient. For a given row, the storage nodes will not conflict with the extra parities since the latter correspond to the storage nodes in other layers. Also, the extra parities will not conflict with each other if they correspond to the storage nodes in different layers. Then, we only need to check the extra parities in the same row corresponding to storage nodes in the same layer. For the extra parities/storage nodes $g_{j,x,y} = h_{j',x',y'}$, given j, x, j', y', the additional coefficients should be different. In this case $k_{j'} - k_a + 1 \leq y \leq k_{j'-1} - k_a$, and there will be at most $\lceil \frac{k_{j'-1}-k_{j'}}{k_{j'}} \rceil$ that make y' a constant in (3.32). As long as we assign $\lceil \frac{k_{j'-1}-k_{j'}}{k_{j'}} \rceil$ number of β in Layer $j', j' \geq 2$ (in Layer 1 we only need one β), Condition 1 is satisfied.

The total number of required additional coefficients is $1 + \sum_{j=2}^{a} \lceil \frac{k_{j-1}-k_j}{k_j} \rceil \triangleq t$. Notice that $(k_{j-1} - k_j)\ell_{j-1} = k_j(\ell_j - \ell_{j-1})$, we have

$$t = 1 + \sum_{j=2}^{a} \lceil \frac{k_{j-1} - k_j}{k_j} \rceil = 1 + \sum_{j=2}^{a} \lceil \frac{\ell_j - \ell_{j-1}}{\ell_{j-1}} \rceil \leq 1 + \sum_{j=2}^{a} (\ell_j - \ell_{j-1}) \leq \ell. \tag{3.41}$$

Moreover, in the best case when we have $k_{j-1} - k_j \leq k_j$ for all j, the number of additional coefficients is a, and $|\mathbb{F}| \geq a|\mathbb{E}|$.

Here, we briefly compare our construction with another flexible MSR construction in [95]. In our code, each node is in $\mathbb{F}^{\ell(n-k)^n}$, where $|\mathbb{F}| \geq t(n-k)n$. Namely, each node requires $\ell(n-k)^n \log_2(t(n-k)n)$ bits. Tamo, Ye and Barg also considered the optimal repair of flexible codes in [95] under their setting, i.e., the downloaded symbols instead of the accessed symbols in each node is flexible to reconstruct the entire information. Their nodes are elements in $\mathbb{F}^{s(n-k)^n}$, and $|\mathbb{F}| \geq s(n-k)n$, where s is defined such that $s_j/s = \ell_j/\ell$ fraction of information are downloaded in each node, where s is the least common multiple of $s_1, s_2, ..., s_a$. Without loss of generality, we can choose $\ell = s$ in our construction. Hence, for Eq. (3.41), the required field size of our construction is better than that of the construction in [95].

Flexible RS MSR codes

In this section, we introduce the construction of Reed-Solomon (RS) MSR codes.

An $RS(n,k)$ code over the finite field $\mathbb{F}$ is defined as

$$RS(n,k) = \{(f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n)) : f \in \mathbb{F}[x], \deg(f) \leq k-1\},$$

where the evaluation points are defined as $\{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq \mathbb{F}$, and $\deg()$ denotes the degree of a polynomial. The encoding polynomial $f(x) = u_0 + u_1 x + \cdots + u_{k-1}x^{k-1}$, where $u_i \in \mathbb{F}, i = 0, 1, \ldots, k-1$ are the information symbols. Every evaluation symbol $f(\alpha_i), i \in [n]$ is called a codeword symbol. RS codes are MDS codes, namely, from any k codeword symbols, the information can be recovered.

Let $\mathbb{B}$ be the base field of $\mathbb{F}$ such that $\mathbb{F} = \mathbb{B}^L$. For repairing RS codes, [40] and [61] shows that any linear repair scheme for a given $RS(n,k)$ over the finite field $\mathbb{F} = \mathbb{B}^L$ is equivalent to finding

a set of repair polynomials $p_{*,v}(x)$ such that for the failed node $f(\alpha_*)$, $* \in [n]$,

$$rank_{\mathbb{B}}(\{p_{*,v}(\alpha_*) : v \in [L]\}) = L, \tag{3.42}$$

where the rank $rank_{\mathbb{B}}(\{\gamma_1, \gamma_2, ..., \gamma_i\})$ is defined as the cardinality of a maximum subset of $\{\gamma_1, \gamma_2, ..., \gamma_i\}$ that is linearly independent over $\mathbb{B}$.

The transmission from helper $f(\alpha_i)$ is

$$Tr_{\mathbb{F}/\mathbb{B}}(p_{*,v}(\alpha_i)f(\alpha_i)), v \in [L], \tag{3.43}$$

where the trace function $Tr_{\mathbb{F}/\mathbb{B}}(x)$ is a linear function such that for all $x \in \mathbb{F}$, $Tr_{\mathbb{F}/\mathbb{B}}(x) \in \mathbb{B}$ [63]. The repair bandwidth for the i-th helper is

$$b_i = rank_{\mathbb{B}}(\{p_{*,v}(\alpha_i) : v \in L\}) \tag{3.44}$$

symbols in $\mathbb{B}$.

The flexible RS MSR code construction is similar to Construction 2 based on parity check matrices, as presented below.

Construction 4. We define a code in $\mathbb{F} = GF(q^L)$ with a set of pairs $(k_j, \ell_j), j \in [a]$ such that $k_j\ell_j = k\ell$, $k_1 > k_2 > ...k_a = k, \ell_a = \ell$, $r = n - k$. In the m_j-th row in Layer $j \in [a]$, the codeword symbols $C_{j,m_j,i}, i \in [n]$ are defined as:

$$C_{j,m_j,i} = f_{j,m_j}(\alpha_{j,m_j,i}), \tag{3.45}$$

and the extra parities $C'_{j,m_j,i}, i \in [k_j - k_a]$ are defined as

$$C'_{j,m_j,i} = f_{j,m_j}(\alpha_{j,m_j,i+n}), \tag{3.46}$$

where $\{f_{j,m_j}(\alpha_{j,m_j,i}), i \in [n + k_j - k_a]\}$ is an $RS(n + k_j - k_a, k_j)$ code. We next define the encoding polynomial $f_{j,m_j}(x)$ and the evaluation point $\alpha_{j,m_j,i}$.

In this construction, we set the extra parities and the corresponding evaluation points exactly the same as the information symbols in lower layers, and we arrange the extra parities the same way as in Construction 2. Specifically, for $C'_{j,x,y}$ in Layer j, $x \in [l_j - l_{j-1}]$, when $k_j - k_{j'-1} + 1 \leq y \leq k_j - k_{j'}$ for $j + 1 \leq j' \leq a$, it is encoded to Layer j' with $\alpha_{j,x,y+n} = \alpha_{j',x',y'}$ and $C'_{j,x,y} = C_{j',x',y'}$, with x', y' in (3.31) (3.32). The encoding polynomial $f_{j',m_{j'}}(x) \in \mathbb{F}$ in Layer j' is defined by the $k_{j'}$ evaluation points and the codeword symbols from the extra parities.

Theorem 13. For Construction 4 is a flexible MSR RS code, if it satisfies:

1) [MDS condition.] In Row m_j of Layer j, $\alpha_{j,m_j,i}, i \in [n + k_j - k_a]$ are distinct elements in $\mathbb{F}$.

2) [Rank conditions.] The same set of repair polynomials $p_{*,v}(x), * \in [n], v \in [L]$, can be used in each row such that:

$$rank_{\mathbb{B}}(\{p_{*,v}(\alpha_{j,m_j,*}) : v \in [L]\}) = L, \tag{3.47}$$

$$b_i = rank_{\mathbb{B}}(\{p_{*,v}(\alpha_{j,m_j,i}) : v \in [L]\}) = L/r, i \in [n]\backslash\{*\}. \tag{3.48}$$

Proof: 1). In the case when $\alpha_{j,m_j,i}, i \in [n + k_j - k_a]$ are distinct elements in $\mathbb{F}$, $\{f_{j,m_j}(\alpha_{j,m_j,i}), i \in [n + k_j - k_a]\}$ is $RS(n + k_j - k_a, k_j)$. Moreover, Layer j' is encoded from the $k_{j'}$ extra parities in Layers $1, 2, \ldots, j' - 1$. Thus, Construction 4 is the same as Construction 1 by using the RS codes as the MDS codes. The flexible reconstruction property is shown in Theorem 9.

2). For the repair, since the extra parities share the same codeword symbols and evaluation points with the storage nodes in lower layers, from (3.43) we know that the transmission for repair is also the same. Thus, we only transmit them once when they are shown as storage nodes.

From (3.48) we know that in each row, each helper transmits L/r symbols, which is optimal. ■

We take the construction in [61] as the $RS(n + k_j - k_a, k_j), j \in [a]$ codes in Construction 4 to show how to construct flexible MSR RS codes.

In [61, Theorem 5], the RS code is defined in $\mathbb{F}$ with evaluation points chosen from $\{\beta_1\alpha_i, \beta_2\alpha_i, ..., \beta_t\alpha_i, i \in [n]\}$ such that $t = \frac{|\mathbb{F}^*|}{|\mathbb{E}^*|}$ for a subfield $\mathbb{E} = GF(q^L)$ of $\mathbb{F}$, and $\alpha_i \in \mathbb{E}$, $i \in [n]$. Here $\beta_1, \ldots, \beta_t$ correspond to elements in $\mathbb{F}$ such that $\{\beta_1\mathbb{E}^*, \ldots, \beta_t\mathbb{E}^*\}$ forms a partition of $\mathbb{F}^*$ [61, Lemma 1]. For the repair polynomials $p_{*,v}(x)$ in [61],

$$rank_{\mathbb{B}}(\{p_{*,v}(\beta\alpha_i) : v \in [L]\}) = \begin{cases} L, i = * \\ \frac{L}{r}, i \neq * \end{cases} \tag{3.49}$$

for all β chosen from $\{\beta_1, ..., \beta_t\}$. The required subfield size in [61] is $|\mathbb{E}| \approx n^n$.

For Construction 4, we assign the evaluation points in the storage nodes as $\alpha_{j,m_j,i} = \beta_{j,m_j}\alpha_i \in \mathbb{F}$, $i \in [n], j \in [a], m_j \in [\ell_j - \ell_{j-1}]$, where β_{j,m_j} is chosen from $\{\beta_1, \ldots, \beta_t\}$. The evaluation points of the extra parities are given by the storage nodes as in (3.31) and (3.32). We assign the additional coefficient β to satisfy Condition 1. Similar to Construction 2, we guarantee that in each row, the $n + k_j - k_a$ evaluation points are distinct and the total number of required β required is $t = 1 + \sum_{j=2}^{a} \lceil \frac{k_{j-1}-k_j}{k_j} \rceil$. In the best case when we have $k_{j-1} - k_j \leq k_j$ for all j, the number of β we required is a. The required field size is $a|\mathbb{E}|$.

Corollary 5. With the RS code in [61], Construction 4 is a flexible MSR RS code.

Proof: We use Theorem 13 to prove that the code is a flexible MSR RS code.

1) [MDS condition.] We have assigned the evaluation points in each row as distinct elements in $\mathbb{F}$.

2) [Rank conditions.] We know from (3.49) that the rank conditions in Theorem 13 are satisfied.

■

3.4 Latency

In this section, we analyze the latency of obtaining the entire information using our codes with flexible number of nodes.

One of the key properties of the flexible storage codes presented in this chapter is that the decoding rows are the first ℓ_j rows if we have R_j available nodes. As a result, the decoder can simply download symbols one by one from each node, and symbols of Layer j can be used for Layers $j, j+1, \ldots, a$.

For one pair of (R_j, ℓ_j), define a random variable T_j associated with the time for the first R_j nodes transmitting the first ℓ_j symbols. T_j is called the *latency* for the j-th layer. Instead of predetermining a fixed pair (R, ℓ) for the system, flexible storage codes allow us to use all possible pairs $(R_j, \ell_j), j \in [a]$. The decoder downloads symbols from all n nodes and as long as it obtains ℓ_j symbols from R_j nodes, the download is complete. For flexible codes with Layers $1, 2, ..., a$, we use $T_{1,2,\ldots,a} = \min(T_j, j \in [a])$ to represent the *latency*.

Notice that for the fixed code with the same failure tolerance level, i.e., $R = R_a, \ell = \ell_a$, its latency is T_a. Since

$$T_{1,2,\ldots,a} = \min(T_j, j \in [a]) \leq T_a, \tag{3.50}$$

we reach the following remark.

Remark 3. Given the storage size per node ℓ, the number of nodes n, and recovery threshold $R = R_a$, the flexible storage code can reduce the latency of obtaining the entire information compared to any fixed array code.

Assume the probability density function (PDF) of T_j is $p_{R_j,\ell_j}(t)$. We calculate the expected delay

as

$$E(T_j) = \int_0^\infty \tau_j p_{R_j,\ell_j}(\tau_j) d\tau_j. \tag{3.51}$$

If a fixed code is adopted, one can optimize the expected latency and get an optimal pair (R^*, ℓ^*) for a given distribution [57], [73]. However, a flexible storage code still outperforms such an optimal fixed code in latency due to Remark 3. Moreover, in practice the choice of (n, k, R, ℓ) depends on the system size and the desired failure tolerance level and is not necessarily optimized for latency.

Next, we take the *Hard Disk Drive* (HDD) storage system as an example to calculate the latency of our flexible storage codes and show how much we can save compared to a fixed MDS code. In this part, we compute the overall latency of a flexible code with (R_1, ℓ_1), (R_2, ℓ_2), and length n. We compare it with the latency of fixed codes with (n, R_1, ℓ_1) and (n, R_2, ℓ_2), respectively.

The HDD latency model is derived in [82], where the overall latency consists of the *positioning time* and the *data transfer time*. The positioning time measures the latency to move the hard disk arm to the desired cylinder and rotate the desired sector to under the disk head. As the accessed physical address for each node is arbitrary, we assume the positioning time is a random variable uniformly distributed, denoted by $U(0, t_{\text{pos}})$, where t_{pos} is the maximum latency required to move through the entire disk. The data transfer time is simply a linear function of the data size, and we assume the transfer time for a single symbol in our code is t_{trans}. Therefore, the overall latency model is $X + \ell \cdot t_{\text{trans}}$, where $X \sim U(0, t_{\text{pos}})$ and ℓ is the number of accessed symbols.

Consider an (n, R, ℓ) fixed code. When R nodes finish the transmission of ℓ symbols, we get all the information. The corresponding latency is called the R-th order statistics. For n independent random variables satisfying $U(0, t_{\text{pos}})$, the R-th order statistics for the positioning time, denoted by U_R, satisfies a *beta distribution* [49]:

$$U_R \sim \text{Beta}(R, n + 1 - R, 0, t_{\text{pos}}). \tag{3.52}$$

with expectation $E[U_R] = \frac{R}{n+1} t_{\text{pos}}$. For a random variable $Y \sim \text{Beta}(\alpha, \beta, a, c)$, the probability density function (pdf) is defined as

$$f(Y = y; \alpha, \beta, a, c) = \frac{(y-a)^{\alpha-1}(c-y)^{\beta-1}}{(c-a)^{\alpha+\beta-1} B(\alpha, \beta)}, \tag{3.53}$$

where

$$B(\alpha, \beta) = \int_{t=0}^{1} t^{\alpha-1}(1-t)^{\beta-1} dt \tag{3.54}$$

is the *Beta function*.

The expectation of overall latency for an (n, R_1, ℓ_1) fixed code, denoted by T_1, is

$$E(T_1) = \frac{R_1}{n+1} t_{\text{pos}} + \ell_1 t_{\text{trans}}. \tag{3.55}$$

Similarly, the expected overall latency $E(T_2)$ for the fixed (n, R_2, ℓ_2) code is

$$E(T_2) = \frac{R_2}{n+1} t_{\text{pos}} + \ell_2 t_{\text{trans}}. \tag{3.56}$$

Now, consider our flexible code with 2 layers. The difference of the positioning times U_{R_1} and U_{R_2} is

$$\Delta U = U_{R_1} - U_{R_2} \sim \text{Beta}(R_1 - R_2, n + 1 - (R_1 - R_2), 0, t_{\text{pos}}). \tag{3.57}$$

Thus, we can get the expectation of the overall latency for our flexible code, denoted by $T_{1,2}$, as

$$\begin{aligned} E(T_{1,2}) &= E(min(T_1, T_2)) \\ &= E(T_1|T_1 - T_2 \leq 0)P(T_1 - T_2 \leq 0) + E(T_2|T_1 - T_2 > 0)P(T_1 - T_2 > 0) \\ &= E(T_1) - E(T_1 - T_2|T_1 - T_2 > 0)P(T_1 - T_2 > 0) \\ &= \frac{R_1}{n+1} t_{\text{pos}} + \ell_1 t_{\text{trans}} - \int_{(\ell_2-\ell_1)t_{\text{trans}}}^{t_{\text{pos}}} [\Delta U - (\ell_2 - \ell_1)t_{\text{trans}}] f(\Delta U) d\Delta U, \end{aligned} \tag{3.58}$$

where the last term is the saved latency compared to an (n, R_1, ℓ_1) code. The saved latency can be calculated as:

$$\begin{aligned} E(T_1 - T_{1,2}) &= \int_{(\ell_2-\ell_1)t_{\text{trans}}}^{t_{\text{pos}}} [\Delta U - (\ell_2 - \ell_1)t_{\text{trans}}] f(\Delta U) d\Delta U \\ &= \frac{a t_{\text{pos}}}{a+b} I_{1-x}(b, a+1) - (\ell_2 - \ell_1)t_{\text{trans}} I_{1-x}(b, a), \end{aligned} \tag{3.59}$$

where $x = \frac{\ell_2-\ell_1}{t_{\text{pos}}} t_{\text{trans}}, a = R_1 - R_2, b = n - (R_1 - R_2) + 1$, and $I_x(a, b)$ is the *regularized incomplete beta function*:

$$I_x(a, b) = \frac{B(x; a, b)}{B(a, b)}, \tag{3.60}$$

with *incomplete beta function*

$$B(x; a, b) = \int_{t=0}^{x} t^{a-1}(1-t)^{b-1} dt. \tag{3.61}$$

Using the fact that $I_x(b, a+1) = I_x(b, a) + \frac{x^b(1-x)^a}{aB(b,a)}$, we have

$$E(\Delta T_1) = (E(T_1) - E(T_2)) I_{1-x}(b, a) + t_{\text{pos}} \frac{R_1 - R_2}{n+1} \frac{x^a(1-x)^b}{aB(a, b)}. \tag{3.62}$$

Similarly, the saved latency compared to an (n, k_2, ℓ_2) code is

$$E(T_2 - T_{1,2}) = (E(T_2) - E(T_1))I_x(a,b) + t_{\text{pos}} \frac{R_1 - R_2}{n+1} \frac{x^a(1-x)^b}{aB(a,b)}. \tag{3.63}$$

From (3.55) and (3.56) we can see that the latency of a fixed MDS code is a function of $n, R, \ell, t_{\text{pos}}$, and t_{trans}. One can optimize the code reconstruction threshold R^* similar to [57] and [73] based on other parameters. However, the system parameters might change over time and one "optimal" R^* cannot provide low latency in all situations. For example, with fixed n, ℓ and the total information size, larger t_{trans} results in a larger R^* while larger t_{pos} results in a smaller R^*. In our flexible codes, we can always pick the best R_j over all $j \in [a]$, thus provide a lower latency.

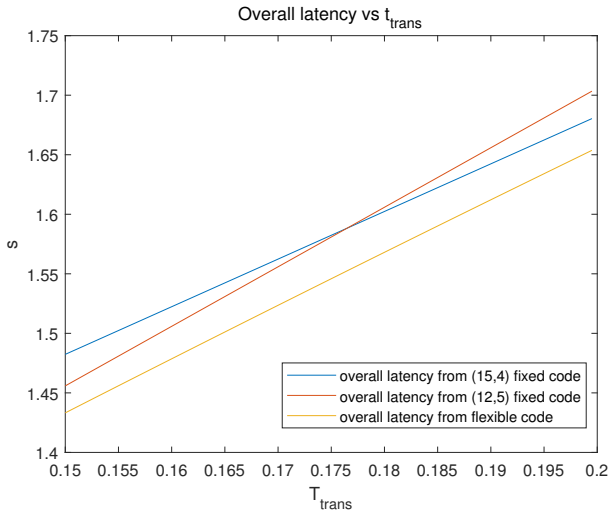

Figure 3.2: Overall latency of fixed codes and flexible codes.

Fig. 3.2 shows the overall latency of fixed codes and flexible recoverable codes with $n = 16, R_1 = 15, R_2 = 12, \ell_1 = 4, \ell_2 = 5.t_{\text{pos}} = 1.$. We fix other parameters and change the unit data transfer time t_{trans}. For fixed codes, a smaller R provides a lower latency with a smaller t_{trans}, and when t_{trans} grows, a larger R is preferred. However, our flexible code always provides a smaller latency, and can save $2\% \sim 5\%$ compared to the better of the two fixed codes.

Our flexible codes can also be applied to distributed computing systems for matrix-vector multiplications [57]. The matrix is divided row-wisely and encoded to n servers using our codes. Each

server is assigned ℓ computation tasks. If any R_j servers complete ℓ_j tasks, we can obtain the final results. Simulation is carried out on Amazon clusters with $n = 8$ servers (m1.small instances). And each task is a multiplication of a square matrix and a vector. The results are shown in Fig. 3.3. We can see a similar trend as that of Fig. 3.2. Our flexible code improves the latency by about 6% compared to the better of the two fixed codes when the matrix size is 1500×1500.

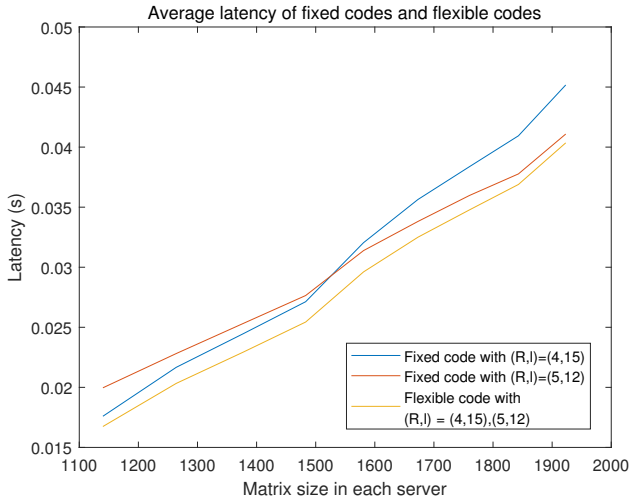

Figure 3.3: Overall latency of fixed codes and flexible codes for matrix-vector multiplication in Amazon cluster. $n = 8, R_1 = 5, R_2 = 4, \ell_1 = 12, \ell_2 = 15$..

3.5 Conclusion

In this chapter, we proposed flexible storage codes and investigated the construction of such codes under various settings. Our analysis shows the benefit of our codes in terms of latency. Open problems include flexible codes for distributed computed problems other than matrix-vector multiplications, code constructions with a smaller finite field size and smaller sub-packetization, and storage codes utilizing partial data transmission from each node similar to universally decodable matrices.

Chapter 4

Flexible Constructions for Distributed Matrix Multiplication

4.1 Introduction

Distributed matrix multiplication has received wide interest because of the huge amount of data computing required by many popular applications like machine learning. In particular, the following basic distributed matrix multiplication is considered: A master wishes to obtain the product of two massive input matrices $A \in \mathbb{F}^{\lambda \times \kappa}$ and $B \in \mathbb{F}^{\kappa \times \mu}$, where $\mathbb{F}$ is some finite field. Each matrix is encoded into N *shares* and distributed to N servers. Each server performs computation on its own shares and sends the *results* to the master. After collecting enough results, the master can decode the desired product AB. To reduce the overall system latency caused by stragglers (servers that fail to respond or respond after the master executes the reconstruction), distributed matrix computing schemes with straggler tolerance are provided in [57, 109, 24, 21, 110, 107, 81, 58, 22, 23, 108, 46, 3, 88, 98, 66, 97, 85, 41, 87, 47, 54, 72, 60, 14, 50, 26, 53, 1, 48, 15, 6, 8, 7]. Among the state-of-the-art schemes, some are based on matrix partitioning such as Polynomial codes [109],

MatDot codes and PolyDot codes [24], Generalized PolyDot codes [21] and Entangled Polynomial (EP) codes [110], and others are based on batch processing such as Lagrange Coded Computing [107] and Cross Subspace Alignment codes [48]. The majority of the literature assumes a fixed number of stragglers, i.e., the data is distributed to N servers and after any R of them complete their computing, the final product can be obtained by the master. Here R is predetermined and called the *recovery threshold*. However, when the number of stragglers is smaller than $N - R$, the master still only uses the results from R servers, and the results of other servers are wasted. In [6, 8, 7, 27, 42, 52, 43, 2, 70], the authors consider a setting in which the number of stragglers is not known as a priori and design schemes that can cope with this setting. References [27, 2, 70] focus on the task scheduling for general distributed computing or distributed learning. The matrix-vector multiplication setting is considered in [6, 8]. Reference[7, 52, 42, 43] consider matrix-matrix multiplication, but they can only handle a special partitioning, i.e., A is split row-wisely and B is split column-wisely. Arbitrary partitioning of input matrices is important in massive matrix multiplication since it enables different utilization of system resources (e.g., the required amount of storage at each server and the amount of communication from servers to the master). When the number of stragglers is fixed, EP codes [110] provide an elegant solution for arbitrary partitioning by encoding the input matrix blocks into a carefully designed polynomial.

This chapter proposes flexible distributed matrix multiplication in order to achieve low latency. The desired product AB can be decoded from collecting the results of a flexible number of servers. As long as the master collects enough results from servers, the computing is completed. This idea of multi-message is also considered in [52, 2, 70]. A naive solution to achieve flexibility is simply applying the EP code [110] with a recovery threshold of RK, where each server gets K pairs of shares instead of one pair of shares. The master can calculate the final results with any RN out of the KN computing results. Thus, each server only needs to compute RK/N results when there is no straggler, and in general the number of results computed in each server can be adjusted based on the number of stragglers. However, by doing so, the computation needs to be done in a field with minimum size of KN, and multiplication in a larger field results in a much bigger delay for

each multiplication [31].

To obtain a smaller field size, we propose the following solution. The main idea is that non-stragglers can finish more tasks to compensate for the effect of the stragglers without knowing the pattern of the stragglers a priori. Specifically, the computation is divided into 2 layers, where the first layer has a larger recovery threshold and the second layer has a smaller recovery threshold. Each server keeps calculating and sending results to the master until enough servers send results to the master, which can be either a larger number of servers for the first layer or a smaller number of servers for both 2 layers. The remaining servers are viewed as stragglers. Our construction only requires a field size of less than $2N$. The computation load of each server can be reduced when there are fewer stragglers than $N - R$. Since computation load is one of the main reasons of delay, our scheme performs better than fixed EP codes with respect to delay, as shown in Fig. 4.1. In Fig. 4.1 we show the Example 1 of Section 4.3. We assume $\lambda = \kappa = \mu = 6U$, for some integer U, and the computation delay for multiplication of two $U \times U$ matrices in each server satisfy the exponential distribution with parameter 0.1. The latency of the EP code is the delay of the 3rd quickest server, and the slowest 2 servers are viewed as stragglers. For the flexible construction, the computation is completed in the cases of 5 servers complete 1 task (no straggler), or 4 servers complete 2 tasks (1 straggler), or of 3 servers complete 3 tasks (2 stragglers). The overall latency is the smallest latency of these 3 cases. The expected latency is 10.79 for EP code, and 8.20 for the flexible construction, hence we save 24%.

Notation: We use calligraphic characters to denote sets. For positive integer N, $[N]$ stands for the set $\{1, 2, \ldots, N\}$. For a matrix M, $|M|$ denotes its cardinality and when M is partitioned into blocks, $M_{(i,j)}$ denotes the block in the i-th row and the j-th column.

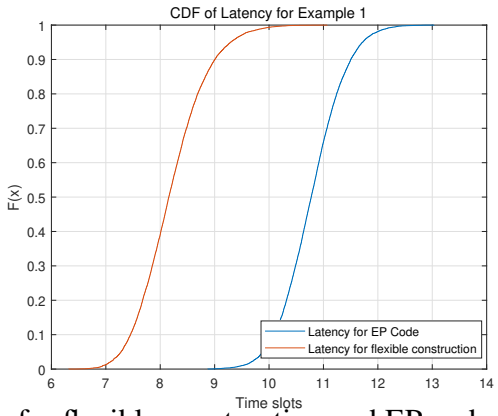

Figure 4.1: CDF of latency for flexible construction and EP code in Example 1 of Section 4.3.

4.2 Problem Statement

We consider a problem of matrix multiplication with two input matrices $A \in \mathbb{F}^{\lambda \times \kappa}$ and $B \in \mathbb{F}^{\kappa \times \mu}$, for some integers λ, κ, μ and a field $\mathbb{F}$. We are interested in computing the product $S = AB$ in a distributed computing environment with 2 sources, a master, and N servers. Sources 1 and 2 hold matrices A and B, respectively. It is assumed that there are up to $N - R$ stragglers among the servers. In non-flexible distributed matrix multiplication, R is called the *recovery threshold*. Given the flexibility parameters R_1, R_2, where $N \geq R_1 > R_2 = R$, the shares (coded matrix sets) $\widetilde{\mathcal{A}}_i$ and $\widetilde{\mathcal{B}}_i$ are generated by sources for Server $i, i \in [N]$. Each share has $R_1 - R_2 + 1$ coded matrices, which are divided into 2 layers. The first layer contains the first coded matrix, denoted by $\widetilde{A}_{i,1}$ or $\widetilde{B}_{i,1}$, and the second layer contains the remaining $R_1 - R_2$ coded matrices, denoted by $\{\widetilde{A}_{i,2}, \cdots, \widetilde{A}_{i,R_1-R_2+1}\}$, or $\{\widetilde{B}_{i,2}, \cdots, \widetilde{B}_{i,R_1-R_2+1}\}$. For $i \in [N]$, the shares and the encoding functions are

$$\widetilde{\mathcal{A}}_i = \{\widetilde{A}_{i,j} \mid j \in [R_1 - R_2 + 1]\} = f_i(A), \tag{4.1}$$

$$\widetilde{\mathcal{B}}_i = \{\widetilde{B}_{i,j} \mid j \in [R_1 - R_2 + 1]\} = g_i(B). \tag{4.2}$$

Then $\widetilde{\mathcal{A}}_i$ and $\widetilde{\mathcal{B}}_i$ are sent to Server i from the sources before the computation starts. Each server is with a storage capacity C [1]. To satisfy the storage constraint, for each Server $i, i \in [N]$,

[1]The maximum storage size C is usually smaller than $|A| + |B|$, otherwise the sources can send A and B to the servers.

$\sum_{M\in\widetilde{\mathcal{A}}_i\cup\widetilde{\mathcal{B}}_i}|M|\leq C$.

Server i computes R_1-R_2+1 tasks in order:

$$\widetilde{S}_{i,j}=h\left(\widetilde{A}_{i,j},\widetilde{B}_{i,j}\right)=\widetilde{A}_{i,j}\cdot\widetilde{B}_{i,j},j\in[R_1-R_2+1],$$

and sends $\widetilde{S}_{i,j}$ to the master once its computation is finished. Since the results are computed in order, the master receives $\widetilde{S}_{i,j_1}$ before $\widetilde{S}_{i,j_2}$ for $\forall i\in[N],j_1<j_2$. Denote $\widetilde{S}_{i,[j]}=\left\{\widetilde{S}_{i,t}\mid t\in[j]\right\}$ and $\widetilde{S}_{\mathcal{K},[j]}=\left\{\widetilde{S}_{i,[j]}\mid i\in\mathcal{K}\right\},\forall\mathcal{K}\subset[N]$.

The decoding function $d_{\mathcal{K},[j]}$ of the master for recovering S satisfies

$$\begin{aligned}&S=d_{\mathcal{K},[j]}\left(\widetilde{S}_{\mathcal{K},[j]}\right),\\&\forall R_2\leq|\mathcal{K}|=R^*\leq R_1,j=R_1-R^*+1.\end{aligned}\tag{4.3}$$

The function set $\{f_i,g_i,h,d_{\mathcal{K},[j]}\mid 1\leq i\leq N,R_2\leq|\mathcal{K}|=R^*\leq R_1,j=R_1-R^*+1\}$ is called the *flexible constructions for distributed matrix multiplication*.

In other words, the sources send all R_1-R_2+1 coded matrices to each server. Then, each server keeps calculating and sending results to the master until the master obtains enough results – either when the quickest R_1 servers complete the first task, or when the quickest R^* servers complete the first R_1-R^*+1 tasks, $R_2\leq R^*<R_1$. The remaining servers are viewed as stragglers. The *latency* is defined as the time required for the master to collect enough results from the start of the computation. For simplicity, in the analysis of this chapter, we assume a small failure probability at each server and a constant time for a unit computation at each server if it is not a straggler.

We want to find flexible constructions with the *storage capacity* C and the *computation load* (i.e., the number of multiplications) at each server as small as possible.

4.3 Construction

In this section, we present our flexible constructions. We start from a motivating example.

Example 9. Consider the matrix multiplication of A and B, for $A \in \mathbb{F}^{\lambda\times\kappa}, B \in \mathbb{F}^{\kappa\times\mu}$, using $N = 5$ servers with at most $N-R = 2$ stragglers. Assume A is partitioned column-wisely and B is partitioned row-wisely: $A = [A_1, A_2], B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$, and the master requires $AB = A_1B_1+A_2B_2$. Applying the EP code [110], server $i, i \in [5]$ receives coded matrices $A_1 + \alpha_i A_2$ and $\alpha_i B_1 + B_2$, and calculates

$$\begin{aligned} &(A_1 + \alpha_i A_2) \cdot (\alpha_i B_1 + B_2) \\ =&A_1B_2 + \alpha_i(A_1B_1 + A_2B_2) + \alpha_i^2 A_2B_1, \end{aligned} \tag{4.4}$$

which is a degree 2 polynomial with respect to α_i. Thus $A_1B_1 + A_2B_2$ can be calculated by 3 distinct evaluations from $\{\alpha_i \mid i \in [5]\}$ using Lagrange interpolation. The total computation load of directly multiplying A and B is $L = \lambda\kappa\mu$, and with EP code the computation load of each server is $L/2$. However, when there is no straggler, the computation of 2 servers are wasted.

Alternatively, we can use a flexible scheme to calculate AB, such that any R^* available servers can complete the computation, $3 = R_2 \leq R^* \leq R_1 = 5$. First, we partition the matrices and get $A = [A_1, A_2, A_3], B = [B_1^T, B_2^T, B_3^T]^T$, and thus the master requires $AB = A_1B_1 + A_2B_2 + A_3B_3$. Let $\{\alpha_i | i \in [7]\}$ be distinct elements in $\mathbb{F}$. The calculation will be divided into 2 layers.

Layer 1: server $i, i \in [5]$, calculates

$$\begin{aligned}
&(A_1 + \alpha_i A_2 + \alpha_i^2 A_3) \cdot (\alpha_i^2 B_1 + \alpha_i B_2 + B_3) \\
=&A_1 B_3 + \alpha_i (A_2 B_3 + A_1 B_2) \\
+&\alpha_i^2 (A_1 B_1 + A_2 B_2 + A_3 B_3) \\
+&\alpha_i^3 (A_2 B_1 + A_3 B_2) + \alpha_i^4 A_3 B_1.
\end{aligned} \tag{4.5}$$

It is a degree 4 polynomial with respect to α_i, and the final product can be obtained from all 5 servers. If there is no straggler, we stop here. In this layer, matrices A, B are divided into smaller pieces compared to fixed EP code and the computation load of each server is $L/3$. If there are stragglers, the servers continue the calculation in Layer 2.

Layer 2: We set $A_{\alpha_i} = (A_1 + \alpha_i A_2 + \alpha_i^2 A_3), B_{\alpha_i} = (\alpha_i^2 B_1 + \alpha_i B_2 + B_3)$ and we further partition them into 2 parts,

$$A_{\alpha_i} = [A_{\alpha_i,1}, A_{\alpha_i,2}], B_{\alpha_i} = \begin{bmatrix} B_{\alpha_i,1} \\ B_{\alpha_i,2} \end{bmatrix}. \tag{4.6}$$

The calculation of each server is shown in Table 4.1.

Since Layer 2 has a similar structure as (4.4), from any 3 of the servers, we can get $A_{\alpha_6} \cdot B_{\alpha_6}$ and/or $A_{\alpha_7} \cdot B_{\alpha_7}$. If there is one straggler, the master obtains $A_{\alpha_6} \cdot B_{\alpha_6}$ from Layer 2, which causes the additional computation load of $L/6$ in a server. If there are 2 stragglers, the master obtains both $A_{\alpha_6} \cdot B_{\alpha_6}$ and $A_{\alpha_7} \cdot B_{\alpha_7}$, which causes the computation load of $L/3$ in Layer 2 for each server.

Note that in this example, there are $R_1 - R_2 + 1 = 3$ coded matrices in a share. That is, $\widetilde{A}_{i,1} = A_{\alpha_i}, \widetilde{A}_{i,2} = A_{\alpha_6,1} + \alpha_i A_{\alpha_6,2}, \widetilde{A}_{i,3} = A_{\alpha_7,1} + \alpha_i A_{\alpha_7,2}$, $\widetilde{B}_{i,1} = B_{\alpha_i}, \widetilde{B}_{i,2} = B_{\alpha_6,1} + \alpha_i B_{\alpha_6,2}, \widetilde{B}_{i,3} = B_{\alpha_7,1} + \alpha_i B_{\alpha_7,2}$, for $i \in [N]$. Server i needs to store $\widetilde{\mathcal{A}}_i$ and $\widetilde{\mathcal{B}}_i$ before the computation steps. Each server computes the $R_1 - R_2 + 1 = 3$ tasks in order independent of the progress of the other

Table 4.1: Calculation tasks in each server for Example 1.

	Server 1	Server 2	Server 3	Server 4	Server 5
Layer 1	$A_{\alpha_1} \cdot B_{\alpha_1}$	$A_{\alpha_2} \cdot B_{\alpha_2}$	$A_{\alpha_3} \cdot B_{\alpha_3}$	$A_{\alpha_4} \cdot B_{\alpha_4}$	$A_{\alpha_5} \cdot B_{\alpha_5}$
Layer 2	$(A_{\alpha_6,1} + \alpha_1 A_{\alpha_6,2})$ $\cdot(\alpha_1 B_{\alpha_6,1} + B_{\alpha_6,2})$, $(A_{\alpha_7,1} + \alpha_1 A_{\alpha_7,2})$ $\cdot(\alpha_1 B_{\alpha_7,1} + B_{\alpha_7,2})$	$(A_{\alpha_6,1} + \alpha_2 A_{\alpha_6,2})$ $\cdot(\alpha_2 B_{\alpha_6,1} + B_{\alpha_6,2})$, $(A_{\alpha_7,1} + \alpha_2 A_{\alpha_7,2})$ $\cdot(\alpha_2 B_{\alpha_7,1} + B_{\alpha_7,2})$	$(A_{\alpha_6,1} + \alpha_3 A_{\alpha_6,2})$ $\cdot(\alpha_3 B_{\alpha_6,1} + B_{\alpha_6,2})$, $(A_{\alpha_7,1} + \alpha_3 A_{\alpha_7,2})$ $\cdot(\alpha_3 B_{\alpha_7,1} + B_{\alpha_7,2})$	$(A_{\alpha_6,1} + \alpha_4 A_{\alpha_6,2})$ $\cdot(\alpha_4 B_{\alpha_6,1} + B_{\alpha_6,2})$, $(A_{\alpha_7,1} + \alpha_4 A_{\alpha_7,2})$ $\cdot(\alpha_4 B_{\alpha_7,1} + B_{\alpha_7,2})$	$(A_{\alpha_6,1} + \alpha_5 A_{\alpha_6,2})$ $\cdot(\alpha_5 B_{\alpha_6,1} + B_{\alpha_6,2})$, $(A_{\alpha_7,1} + \alpha_5 A_{\alpha_7,2})$ $\cdot(\alpha_5 B_{\alpha_7,1} + B_{\alpha_7,2})$

servers.

From Example 9, when there is no straggler (which is more likely in most practical systems), we can reduce the computation load of each server from $L/2$ to $L/3$. In the worst case, we can tolerate 2 stragglers and get the desired results. The resulting latency under an exponential model is plotted in Fig. 4.1.

In this example, the storage size required for each server is $\frac{2\lambda\kappa}{3} + \frac{2\kappa\mu}{3}$ for our flexible construction, and $\frac{\lambda\kappa}{2} + \frac{\kappa\mu}{2}$ for the EP code. We will discuss how to partition the matrices to obtain a good performance on storage size in Section 4.4.

Next, we present the general construction of our flexible schemes.

Construction 6. Assume we have $N \geq R_1 > R_2 = R$, $R_j = p_j m_j n_j + p_j - 1, j \in [2]$, and distinct elements $\{\alpha_i \mid i \in [N + R_1 - R_2]\}$ from the finite field $\mathbb{F}$. With p_1, m_1, n_1, matrices A, B are partitioned as

$$\begin{bmatrix} A_{(1,1)} & \cdots & A_{(1,p_1)} \\ A_{(2,1)} & \cdots & A_{(2,p_1)} \\ \vdots & \vdots & \vdots \\ A_{(m_1,1)} & \cdots & A_{(m_1,p_1)} \end{bmatrix}, \begin{bmatrix} B_{(1,1)} & \cdots & B_{(1,n_1)} \\ B_{(2,1)} & \cdots & B_{(2,n_1)} \\ \vdots & \vdots & \vdots \\ B_{(p_1,1)} & \cdots & B_{(p_1,n_1)} \end{bmatrix}. \tag{4.7}$$

In Layer 1, set $A^{(1)} = A, B^{(1)} = B$, we calculate $f_{1,A^{(1)}}(\alpha_i) \cdot f_{1,B^{(1)}}(\alpha_i)$ in server i, where

$$f_{1,A^{(1)}}(\alpha_i) = \sum_{u=1}^{m_1} \sum_{v=1}^{p_1} A_{(u,v)}^{(1)} \alpha_i^{v-1+p_1(u-1)}, \tag{4.8}$$

$$f_{1,B^{(1)}}(\alpha_i) = \sum_{u=1}^{p_1} \sum_{v=1}^{n_1} B_{(u,v)}^{(1)} \alpha_i^{p_1-u+p_1 m_1(v-1)}, \tag{4.9}$$

are shares based on EP codes [110]. Here, for Server i, $\widetilde{A}_{i,1} = f_{1,A^{(1)}}(\alpha_i)$, $\widetilde{B}_{i,1} = f_{1,B^{(1)}}(\alpha_i)$.

In Layer 2, we partition matrices $f_{1,A^{(1)}}(\alpha_{N+t})$, $f_{1,B^{(1)}}(\alpha_{N+t})$, $t \in [R_1 - R_2]$, with parameters p_2, m_2, n_2. Server i calculates $f_{2,A^{(2)}}(\alpha_i) \cdot f_{2,B^{(2)}}(\alpha_i)$, where $\left(A^{(2)}, B^{(2)}\right) \in \{(f_{1,A^{(1)}}(\alpha_{N+t}),$ $f_{1,B^{(1)}}(\alpha_{N+t}) \mid t \in [R_1 - R_2]\}$ and

$$f_{2,A^{(2)}}(\alpha_i) = \sum_{u=1}^{m_2} \sum_{v=1}^{p_2} A_{(u,v)}^{(2)} \alpha_i^{v-1+p_2(u-1)}, \tag{4.10}$$

$$f_{2,B^{(2)}}(\alpha_i) = \sum_{u=1}^{p_2} \sum_{v=1}^{n_2} B_{(u,v)}^{(2)} \alpha_i^{p_2-u+p_2 m_2(v-1)}. \tag{4.11}$$

In Layer 2, for Server $i \in [N]$, index $t \in [R_1 - R_2]$, $A^{(2)} = f_{1,A^{(1)}}(\alpha_{N+t})$, and $B^{(2)} = f_{1,B^{(1)}}(\alpha_{N+t})$, the corresponding coded matrices are

$$\widetilde{A}_{i,t+1} = f_{2,A^{(2)}}(\alpha_i),$$
$$\widetilde{B}_{i,t+1} = f_{2,B^{(2)}}(\alpha_i).$$

The calculation tasks in both layers are shown in Table 4.2. Since we only use $N + R_1 - R_2$ distinct α_i values, the required field size is $|\mathbb{F}| \geq N + R_1 - R_2$. In Layer 1, $A^{(1)} = A, B^{(1)} = B$. Layer 2 calculates for all pairs of $\left(A^{(2)}, B^{(2)}\right) \in \{\left(f_{1,A^{(1)}}(\alpha_{N+t}), f_{1,B^{(1)}}(\alpha_{N+t}\right) \mid t \in [R_1 - R_2]\}$.

Table 4.2: Calculation tasks in each server for the general construction.

	Server 1	Server 2	...	Server N
Layer 1	$f_{1,A^{(1)}}(\alpha_1) \cdot f_{1,B^{(1)}}(\alpha_1)$	$f_{1,A^{(1)}}(\alpha_2) \cdot f_{1,B^{(1)}}(\alpha_2)$	...	$f_{1,A^{(1)}}(\alpha_N) \cdot f_{1,B^{(1)}}(\alpha_N)$
Layer 2	$f_{2,A^{(2)}}(\alpha_1) \cdot f_{2,B^{(2)}}(\alpha_1)$	$f_{2,A^{(2)}}(\alpha_2) \cdot f_{2,B^{(2)}}(\alpha_2)$	...	$f_{2,A^{(2)}}(\alpha_N) \cdot f_{2,B^{(2)}}(\alpha_N)$

Theorem 14. In Construction 6, assume we have R^* available servers and $R_2 \leq R^* \leq N$, we only need

$$L_{\text{flex}} = \begin{cases} \frac{\lambda\kappa\mu}{m_1 p_1 n_1}, & R^* \geq R_1, \\ \frac{\lambda\kappa\mu}{m_1 p_1 n_1} + \frac{\lambda\kappa\mu(R_1 - R^*)}{m_1 m_2 p_1 p_2 n_1 n_2}, & R_2 \leq R^* < R_1. \end{cases} \tag{4.12}$$

computation load in each server to obtain the final product, and the storage capacity required is

$$C_{\text{flex}} = \frac{1}{p_1}\left(\frac{\lambda\kappa}{m_1} + \frac{\kappa\mu}{n_1}\right) + \frac{R_1 - R_2}{p_1 p_2}\left(\frac{\lambda\kappa}{m_1 m_2} + \frac{\kappa\mu}{n_1 n_2}\right). \tag{4.13}$$

Proof: We first look at the computation load.

In the case that the number of available servers $R^* \geq R_1$, according to the correctness of EP codes [110], the required results $A \times B$ can be obtained by collecting R_1 evaluation points of $f_{1,A^{(1)}}(\alpha_i) \times f_{1,B^{(1)}}(\alpha_i)$. Thus, we only need the computation in Layer 1. In Layer 1, we calculate $f_{1,A^{(1)}}(\alpha_i) \cdot f_{1,B^{(1)}}(\alpha_i)$. From (4.7), (4.8) and (4.9) we know that $f_{1,A^{(1)}}(\alpha_i)$ has size $\frac{\lambda}{m_1} \cdot \frac{\kappa}{p_1}$ and $f_{1,B^{(1)}}(\alpha_i)$ has size $\frac{\kappa}{p_1} \cdot \frac{\mu}{n_1}$. Thus, normalized by the cost of a single multiplication operation, the computation in Layer 1 is

$$L_1 = \frac{\lambda\kappa\mu}{m_1 p_1 n_1}. \tag{4.14}$$

When $R_2 \leq R^* < R_1$, we only have R^* evaluation points of $f_{1,A^{(1)}}(\alpha_i) \cdot f_{1,B^{(1)}}(\alpha_i)$ calculated in Layer 1. Then, we need to obtain additional $R_1 - R^*$ evaluation points. In Layer 2, $f_{2,A^{(2)}}(\alpha_i) \cdot f_{2,B^{(2)}}(\alpha_i)$, $i \in [N]$, are calculated at the servers with $(A^{(2)}, B^{(2)})$ chosen from $\{\big(f_{1,A^{(1)}}(\alpha_{N+t}), f_{1,B^{(1)}}(\alpha_{N+t})\big)\}, t \in [R_1 - R_2]\}$. With each pair of $(A^{(2)}, B^{(2)})$, the master can

calculate one evaluation point of $f_{1,A^{(1)}}(\alpha_{N+t}) \cdot f_{1,B^{(1)}}(\alpha_{N+t})$ since (4.10) and (4.11) are exactly the EP code [110]. From (4.10) and (4.11), we know that $f_{2,A^{(2)}}(\alpha_i)$ has size $\frac{\lambda}{m_1 m_2} \cdot \frac{\kappa}{p_1 p_2}$ and $f_{2,B^{(2)}}(\alpha_i)$ has size $\frac{\kappa}{p_1 p_2} \cdot \frac{\mu}{n_1 n_2}$. Thus, the total computation in Layer 2 is

$$L_2 = \frac{(R_1 - R^*)L_1}{p_2 m_2 n_2}. \tag{4.15}$$

Combining (4.14) and (4.15), the computation load is given as (4.12).

For the storage, we first look at the storage size required for each layer. In Layer 1, we need to store $f_{1,A^{(1)}}(\alpha_i), f_{1,B^{(1)}}(\alpha_i)$, then

$$C_1 = \frac{1}{p_1}\left(\frac{\lambda\kappa}{m_1} + \frac{\kappa\mu}{n_1}\right). \tag{4.16}$$

In Layer 2, we need to store all pairs of $f_{2,A^{(2)}}(\alpha_i), f_{2,B^{(2)}}(\alpha_i)$, for $R_1 - R_2$ choices of $(A^{(2)}, B^{(2)})$. The required storage size is

$$C_2 = \frac{(R_1 - R_2)}{p_1 p_2}\left(\frac{\lambda\kappa}{m_1 m_2} + \frac{\kappa\mu}{n_1 n_2}\right). \tag{4.17}$$

Thus, we obtain the total required storage size as (4.13). ■

Remark. Cross Subspace Alignment codes and Generalized Cross Subspace Alignment codes [48] are designed to handle batch processing of matrix multiplication. Our construction can also be easily modified to handle batch processing based on these two codes.

4.4 Optimization

In this section, we discuss how to pick partitioning parameters $p, m, n,$ to improve the system performance, i.e., to minimize the computation load given the storage constraint C in each server.

We first discuss fixed EP code with a fixed recovery threshold R, which satisfies $R = m_0 p_0 n_0 + p_0 - 1$ according to [110], for some undetermined p_0, m_0, n_0. The computation load and the storage size required are shown in [110] as

$$L_{\text{EP}} = \frac{\lambda\kappa\mu}{m_0 p_0 n_0}, \;\; C_{\text{EP}} = \frac{1}{p_0}\left(\frac{\lambda\kappa}{m_0} + \frac{\kappa\mu}{n_0}\right). \tag{4.18}$$

Thus, the optimization problem can be formulated as

$$\begin{aligned} \min_{p_0, m_0, n_0} \quad & L_{\text{EP}} = \frac{\lambda\kappa\mu}{m_0 p_0 n_0}, \\ \text{s.t.} \quad & R = p_0 m_0 n_0 + p_0 - 1, \\ & \frac{\lambda\kappa}{p_0 m_0} + \frac{\kappa\mu}{p_0 n_0} \leq C, \\ & p_0, m_0, n_0 \text{ are integers.} \end{aligned} \tag{4.19}$$

Theorem 15. The optimization in (4.19) without the integer constraint has solution

$$p_0^* = \frac{1}{2}(R+1) - \frac{1}{2}\sqrt{(R+1)^2 - 16\frac{\lambda\kappa^2\mu}{C^2}}, \tag{4.20}$$

and m_0^*, n_0^* are given by $m_0 n_0 = \frac{R+1}{p_0} - 1$ and $\lambda\kappa n_0 = \kappa\mu m_0$.

Proof: Using the threshold constraint

$$p_0 m_0 n_0 = R + 1 - p_0, \tag{4.21}$$

we have $L_{\text{EP}} = \frac{\lambda\kappa\mu}{R+1-p_0}$, which is an increasing function of p_0. So, we minimize p_0 under the

constraint that

$$\frac{(\lambda\kappa n_0 + \kappa\mu m_0)}{R+1-p_0} \leq C. \tag{4.22}$$

Also, we have

$$\lambda\kappa n_0 + \kappa\mu m_0 \geq 2\sqrt{\lambda\kappa^2\mu m_0 n_0} = 2\sqrt{\lambda\kappa^2\mu\left(\frac{R+1}{p_0} - 1\right)} \tag{4.23}$$

and it holds with equality if and only if $\lambda\kappa n_0 = \kappa\mu m_0$. Thus, we have (4.22) as

$$2\sqrt{\frac{\lambda\kappa^2\mu}{(R+1-p_0)p_0}} \leq C, \tag{4.24}$$

which decreases with p_0 since the derivative satisfies

$$\frac{\mathrm{d}\,(R+1-p_0)p_0}{\mathrm{d}\,p_0} = R+1-2p_0 = p_0 m_0 n_0 - p_0 \geq 0. \tag{4.25}$$

Thus, L_{EP} reaches its optimal value when (4.24) holds with equality and $\lambda\kappa n_0 = \kappa\mu m_0$. Combining (4.21), the optimal p_0^* is given by (4.20), and then m_0^*, n_0^* can be obtained accordingly. ■

Notice that p_0, m_0, n_0 are integers, we pick these 3 parameters close to the optimal values that satisfy all the constraints in (4.19).

Next, we consider the flexible constructions with predetermined $R_1, R_2 = R$. Assume that the probability that each server is a straggler is ϵ. The average computation load is

$$\begin{aligned} E[L_{\text{flex}}] = &\sum_{R^*=R_2}^{N} \binom{N}{R^*} (1-\epsilon)^{R^*} \epsilon^{N-R^*} \frac{\lambda\kappa\mu}{p_1 m_1 n_1} \\ &+ \sum_{R^*=R_2}^{R_1-1} \binom{N}{R^*} (1-\epsilon)^{R^*} \epsilon^{N-R^*} \frac{\lambda\kappa\mu(R_1-R^*)}{m_1 m_2 p_1 p_2 n_1 n_2}. \end{aligned} \tag{4.26}$$

In practical systems, ϵ is small (e.g., less than 110 failures over 3000-node production clusters

of Facebook per day [84]), so we ignore the second term in (4.26) and use the approximation $L_{\text{flex}} = \frac{\lambda\kappa\mu}{p_1 m_1 n_1}$ in our optimization problem. Combined with (4.13), the optimization problem can be formulated as

$$\begin{aligned}
&\min_{p_1,m_1,n_1,p_2,m_2,n_2} L_{\text{flex}} = \frac{\lambda\kappa\mu}{p_1 m_1 n_1}, \\
\text{s.t.}\quad & R_j = p_j m_j n_j + p_j - 1, j \in [2], \\
& \frac{1}{p_1}\left(\frac{\lambda\kappa}{m_1} + \frac{\kappa\mu}{n_1}\right) + \frac{(R_1 - R_2)}{p_1 p_2}\left(\frac{\lambda\kappa}{m_1 m_2} + \frac{\kappa\mu}{n_1 n_2}\right) \leq C, \\
& p_1, m_1, n_1, p_2, m_2, n_2 \text{ are integers.}
\end{aligned} \tag{4.27}$$

Theorem 16. The solution to (4.27) without the integer constraint for p_1, m_1, n_1, p_2 is

$$p_1^* = \frac{R_1 + 1}{2} - \sqrt{\frac{(R_1+1)^2}{4} - \frac{4\lambda\kappa^2\mu(2R_1 - R_2 + 1)^2}{C^2(R_2+1)^2}}, \tag{4.28}$$

m_1^*, n_1^* are given by $m_1 n_1 = \frac{R_1+1}{p_1} - 1$ and $\lambda\kappa n_1 = \kappa\mu m_1$, and $p_2^* = \frac{R_2+1}{2}, m_2^* = 1, n_2^* = 1$.

Proof: Using $p_1 m_1 n_1 = R + 1 - p_1$, we have $L_{\text{flex}} = \frac{\lambda\kappa\mu}{R_1+1-p_1}$, which is an increasing function of p_1, so we need to minimize p_1.

Using $m_j n_j = \frac{R_j+1}{p_j} - 1$, similar to (4.23), we have:

$$\frac{1}{p_1}\left(\frac{\lambda\kappa}{m_1} + \frac{\kappa\mu}{n_1}\right) \geq 2\sqrt{\frac{\lambda\kappa^2\mu}{(R_1 + 1 - p_1)p_1}}, \tag{4.29}$$

$$\begin{aligned}
&\frac{(R_1 - R_2)}{p_1 p_2}\left(\frac{\lambda\kappa}{m_1 m_2} + \frac{\kappa\mu}{n_1 n_2}\right) \\
\geq & 2(R_1 - R_2)\sqrt{\frac{\lambda\kappa^2\mu}{(R_1 + 1 - p_1)(R_2 + 1 - p_2)p_1 p_2}}.
\end{aligned} \tag{4.30}$$

Similar to (4.25), we know that (4.30) is a decreasing function of p_1 and p_2. Thus, when p_2 reaches its maximum, p_1 is minimized. Noticing that $p_2 = \frac{R_2+1}{m_2 n_2 + 1}$ and m_2, n_2 are integers, we set

$p_2^* = \frac{R_2+1}{2}, m_2^* = 1, n_2^* = 1$. The optimal p_1^* is obtained from (4.29) and (4.30). ■

Again, we pick $p_1, m_1, n_1, p_2, m_2, n_2$ as integers around the optimal value satisfying (4.27) as our final choice.

Example 10. Assume we have $N = 8$ servers and we need to tolerate $N - R = 1$ straggler. $\lambda = \kappa = \mu$ and the storage size of each server is limited by $C = \frac{8}{7}\lambda\kappa$. Using the EP code, the optimal choice of $\{p_0, m_0, n_0\}$ is $\{1, 1, 7\}$, which results in a storage size of $\frac{8}{7}\lambda\kappa$ and a computation load per server of $\frac{1}{7}\lambda\kappa\mu = 0.143\lambda\kappa\mu$. Using the 2-layer flexible codes with $R_1 = 8$ and $R_2 = 7$, the optimal parameters are chosen as $p_1 = 1, m_1 = 2, n_1 = 4, p_2 = 4, m_2 = 1, n_2 = 1$, which cost a storage size of $\frac{15}{16}\lambda\kappa$ and a computation load of $\frac{1}{8}\lambda\kappa\mu$ when there is no straggler, with an additional computation load of $\frac{1}{32}\lambda\kappa\mu$ when there is one straggler. Assuming the probability of one straggler to be 10%, the average computation load is $0.128\lambda\kappa\mu$. In this example, we save both storage size and average computation load while maintaining one straggler tolerance.

4.5 Conclusion

In this chapter, a flexible construction for distributed matrix multiplication is proposed and the optimal parameters are discussed. The construction can also be generalized to batch processing of matrix multiplication.

Chapter 5

Conclusion and Future Work

In this dissertation, flexible coding constructions and schemes are provided. The repair problem of RS codes is studied, our code constructions and repair schemes provide a flexible tradeoff between the repair bandwidth and the sub-packetization size. Aiming at reduce the accessing latency with unknown failures, flexible storage code constructions are proposed and accessing latency model is analyzed. The constructions can be applied to different application scenarios. The flexible distributed matrix computing schemes are also investigated. The code constructions are proposed, and the optimization problem of assigning tasks to servers is analyzed when given the capacity of the servers.

For the future work, we would like to extend our flexible constructions and schemes for more application scenarios, like distributed machine learning algorithms, DNA storage and consistent data storage. In these scenarios, the stragglers are hard to predict, a flexible construction that allows efficient use different number of available servers can reduce the latency of the system. Also, with some feedback and communications from the servers, the flexible constructions and schemes can be improved. The task assignments can be adjusted according to the feedback and the servers can be used more efficiently with communications.

Bibliography

[1] M. Aliasgari, O. Simeone, and J. Kliewer. Distributed and private coded matrix computation with flexible communication load. *arXiv preprint arXiv:1901.07705*, 2019.

[2] M. M. Amiri and D. Gündüz. Computation scheduling for distributed machine learning with straggling workers. *IEEE Transactions on Signal Processing*, 67(24):6270–6284, 2019.

[3] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran. Straggler-proofing massive-scale distributed matrix multiplication with d-dimensional product codes. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1993–1997. IEEE, 2018.

[4] A. Beimel and Y. Stahl. Robust information-theoretic private information retrieval. *Journal of Cryptology*, 20(3):295–321, 2007.

[5] R. Bitar and S. El Rouayheb. Staircase-pir: Universally robust private information retrieval. In *2018 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2018.

[6] R. Bitar, P. Parag, and S. E. Rouayheb. Minimizing latency for secure coded computing using secret sharing via staircase codes. *IEEE Transactions on Communications*, 68(8):4609–4619, 2020.

[7] R. Bitar, M. Xhemrishi, and A. Wachter-Zeh. Adaptive private distributed matrix multiplication. *arXiv preprint arXiv:2101.05681*, 2021.

[8] R. Bitar, Y. Xing, Y. Keshtkarjahromi, V. Dasari, S. E. Rouayheb, and H. Seferoglu. Private and rateless adaptive coded matrix-vector multiplication. *arXiv preprint arXiv:1909.12611*, 2019.

[9] M. Blaum. Multiple-layer integrated interleaved codes: A class of hierarchical locally recoverable codes. *arXiv preprint arXiv:2009.12456*, 2020.

[10] M. Blaum, J. Bruck, and A. Vardy. MDS array codes with independent parity symbols. *IEEE Transactions on Information Theory*, 42(2):529–542, 1996.

[11] M. Blaum, J. L. Hafner, and S. Hetzler. Partial-MDS codes and their application to raid type of architectures. *IEEE Transactions on Information Theory*, 59(7):4510–4519, 2013.

[12] 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*, 59(5):2974–2987, 2013.

[13] G. Calis and O. O. Koyluoglu. A general construction for PMDS codes. *IEEE Communications Letters*, 21(3):452–455, 2016.

[14] W. Chang and R. Tandon. On the capacity of secure distributed matrix multiplication. *IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2018.

[15] Z. Chen, Z. Jia, Z. Wang, and S. A. Jafar. GCSA codes with noise alignment for secure coded multi-party batch matrix multiplication. *IEEE Journal on Selected Areas in Information Theory, Early Access*, 2021.

[16] A. Chowdhury and A. Vardy. Improved schemes for asymptotically optimal repair of MDS codes. *arXiv preprint arXiv:1710.01867*, 2017.

[17] H. Dau, I. Duursma, H. M. Kiah, and O. Milenkovic. Repairing Reed-Solomon codes with multiple erasures. *IEEE Transactions on Information Theory*, 2018.

[18] H. Dau and O. Milenkovic. Optimal repair schemes for some families of full-length Reed-Solomon codes. In *Information Theory (ISIT), 2017 IEEE International Symposium on*, pages 346–350. IEEE, 2017.

[19] C. Devet, I. Goldberg, and N. Heninger. Optimally robust private information retrieval. In *21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 269–283, 2012.

[20] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *IEEE Transactions on Information Theory*, 56(9):4539–4551, 2010.

[21] S. Dutta, Z. Bai, H. Jeong, T. Low, and P. Grover. A unified coded deep neural network training strategy based on generalized polydot codes for matrix multiplication. *ArXiv:1811.10751*, Nov. 2018.

[22] S. Dutta, V. Cadambe, and P. Grover. Short-dot: Computing large linear transforms distributedly using coded short dot products. In *Advances In Neural Information Processing Systems*, pages 2100–2108, 2016.

[23] S. Dutta, V. Cadambe, and P. Grover. Coded convolution for parallel and distributed computing within a deadline. *arXiv preprint arXiv:1705.03875*, 2017.

[24] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover. On the optimal recovery threshold of coded matrix multiplication. *IEEE Transactions on Information Theory*, 66(1):278–301, 2020.

[25] I. Duursma and H. Dau. Low bandwidth repair of the RS (10, 4) Reed-Solomon code. In *2017 Information Theory and Applications Workshop (ITA)*, pages 1–10. IEEE, 2017.

[26] R. G. D'Oliveira, S. E. Rouayheb, and D. Karpuk. GASP codes for secure distributed matrix multiplication. *IEEE Transactions on Information Theory*, 2020. early access, DOI: 10.1109/TIT.2020.2975021.

[27] N. Ferdinand and S. C. Draper. Hierarchical coded computations. *IEEE International Symposium on Information Theory*, 2018.

[28] A. Fikes. Colossus, successor to Google File System, 2015.

[29] M. Forbes and S. Yekhanin. On the locality of codeword symbols in non-linear codes. *Discrete mathematics*, 324:78–84, 2014.

[30] A. Ganesan and P. O. Vontobel. On the existence of universally decodable matrices. *IEEE transactions on information theory*, 53(7):2572–2575, 2007.

[31] S. B. Gashkov and I. S. Sergeev. Complexity of computation in finite fields. *Journal of Mathematical Sciences*, 191(5):661–685, 2013.

[32] P. Gopalan, C. Huang, B. Jenkins, and S. Yekhanin. Explicit maximally recoverable codes with locality. *IEEE Transactions on Information Theory*, 60(9):5245–5256, 2014.

[33] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin. On the locality of codeword symbols. *IEEE Transactions on Information theory*, 58(11):6925–6934, 2012.

[34] S. Goparaju, A. Fazeli, and A. Vardy. Minimum storage regenerating codes for all parameters. *IEEE Transactions on Information Theory*, 63(10):6318–6328, 2017.

[35] D. Goss. *Basic structures of function field arithmetic*. Springer Science & Business Media, 2012.

[36] K. M. Greenan, E. L. Miller, and S. T. J. Schwarz. Optimizing galois field arithmetic for diverse processor architectures and applications. In *Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. MASCOTS 2008. IEEE International Symposium on*, pages 1–10. IEEE, 2008.

[37] V. Guruswami, S. V. Lokam, and S. V. M. Jayaraman. Epsilon-MSR codes: Contacting fewer code blocks for exact repair. *arXiv preprint arXiv:1807.01166*, 2018.

[38] V. Guruswami and A. S. Rawat. MDS code constructions with small sub-packetization and near-optimal repair bandwidth. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2109–2122. Society for Industrial and Applied Mathematics, 2017.

[39] V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 28–37. IEEE, 1998.

[40] V. Guruswami and M. Wootters. Repairing Reed-Solomon codes. *IEEE Transactions on Information Theory*, 2017.

[41] F. Haddadpour and V. R. Cadambe. Codes for distributed finite alphabet matrix-vector multiplication. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1625–1629. IEEE, 2018.

[42] B. Hasırcıoglu, J. Gómez-Vilardebó, and D. Gündüz. Bivariate polynomial coding for exploiting stragglers in heterogeneous coded computing systems. *ArXiv:2001.07227*, 2020.

[43] B. Hasırcıoğlu, J. Gómez-Vilardebó, and D. Gündüz. Bivariate hermitian polynomial coding for efficient distributed matrix multiplicationn. *2020 IEEE Global Communications Conference*, pages 1–6, 2020.

[44] W. Huang, M. Langberg, J. Kliewer, and J. Bruck. Communication efficient secret sharing. *IEEE Transactions on Information Theory*, 62(12):7195–7206, 2016.

[45] H. Jafarkhani and M. Hajiaghayi. Cost-efficient repair for storage systems using progressive engagement, Jan. 22 2019. US Patent 10,187,088.

[46] T. Jahani-Nezhad and M. A. Maddah-Ali. Codedsketch: A coding scheme for distributed computation of approximated matrix multiplications. *arXiv preprint arXiv:1812.10460*, 2018.

[47] H. Jeong, F. Ye, and P. Grover. Locally recoverable coded matrix multiplication. In *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 715–722. IEEE, 2018.

[48] Z. Jia and S. Jafar. Cross-subspace alignment codes for coded distributed batch computation. *ArXiv:1909.13873*, 2019.

[49] M. Jones. Kumaraswamy's distribution: A beta-type distribution with some tractability advantages. *Statistical methodology*, 6(1):70–81, 2009.

[50] J. Kakar, S. Ebadifar, and A. Sezgin. On the capacity and straggler-robustness of distributed secure matrix multiplication. *IEEE Access*, 7:45783–45799, 2019.

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

[52] S. Kiani, N. Ferdinand, and S. C. Draper. Exploitation of stragglers in coded computation. *IEEE International Symposium on Information Theory*, 2018.

[53] M. Kim and J. Lee. Private secure coded computation. *IEEE Communications Letters*, 23(11):1918–1921, 2019.

[54] M. Kim, J.-y. Sohn, and J. Moon. Coded matrix multiplication on a group-based model. *arXiv preprint arXiv:1901.05162*, 2019.

[55] K. Kralevska, D. Gligoroski, R. E. Jensen, and H. Øverby. HashTag erasure codes: from theory to practice. *IEEE Transactions on Big Data*, 4(4):516–529, 2018.

[56] C. Lai, S. Jiang, L. Yang, S. Lin, G. Sun, Z. Hou, C. Cui, and J. Cong. Atlas: Baidu's key-value storage system for cloud data. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–14. IEEE, 2015.

[57] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529, 2017.

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

[59] J. Li and X. Tang. A systematic construction of mds codes with small sub-packetization level and near optimal repair bandwidth. *arXiv preprint arXiv:1901.08254*, 2019.

[60] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr. Coding for distributed fog computing. *IEEE Communications Magazine*, 55(4):34–40, 2017.

[61] 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*, 65(9):5484–5502, 2019.

[62] G. Liang and U. C. Kozat. Tofec: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 826–834. IEEE, 2014.

[63] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications*. Cambridge university press, 1994.

[64] J. Luo, K. D. Bowers, A. Oprea, and L. Xu. Efficient software implementations of large finite fields GF(2^n) for secure storage applications. *ACM Transactions on Storage (TOS)*, 8(1):2, 2012.

[65] F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes*. Elsevier, 1977.

[66] A. Mallick, M. Chaudhari, U. Sheth, G. Palanikumar, and G. Joshi. Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(3), 2019.

[67] J. Mardia, B. Bartan, and M. Wootters. Repairing multiple failures for scalar MDS codes. *IEEE Transactions on Information Theory*, 2018.

[68] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang, et al. f4: Facebook's warm BLOB storage system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 383–398, 2014.

[69] P. Narayanan, S. Samal, and S. Nanniyur. Yahoo cloud object store-object storage at exabyte scale, 2017.

[70] E. Ozfatura, S. Ulukus, and D. Gündüz. Straggler-aware distributed learning: Communication–computation latency trade-off. *Entropy*, 22(5):544, 2020.

[71] D. S. Papailiopoulos, A. G. Dimakis, and V. R. Cadambe. Repair optimal erasure codes through hadamard designs. *IEEE Trans. Inf. Theory*, 59(5):3021–3037, 2013.

[72] H. Park, K. Lee, J.-y. Sohn, C. Suh, and J. Moon. Hierarchical coding for distributed computing. *arXiv preprint arXiv:1801.04686*, 2018.

[73] P. Peng, E. Soljanin, and P. Whiting. Diversity vs. parallelism in distributed computing with redundancy. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 257–262. IEEE, 2020.

[74] J. S. Plank, K. M. Greenan, and E. L. Miller. Screaming fast galois field arithmetic using intel simd instructions. In *FAST*, pages 299–306, 2013.

[75] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, Z. Wilcox-O'Hearn, et al. A performance evaluation and examination of open-source erasure coding libraries for storage. In *Fast*, volume 9, pages 253–265, 2009.

[76] A. Ramamoorthy, L. Tang, and P. O. Vontobel. Universally decodable matrices for distributed matrix-vector multiplication. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 1777–1781. IEEE, 2019.

[77] K. V. Rashmi, N. B. Shah, and P. V. Kumar. Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction. *IEEE Transactions on Information Theory*, 57(8):5227–5239, 2011.

[78] 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*, pages 1–6. IEEE, 2016.

[79] 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*, 64(12):7529–7550, 2018.

[80] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[81] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr. Coded computation over heterogeneous clusters. *IEEE Transactions on Information Theory*, 65(7):4227–4242, 2019.

[82] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *Computer*, 27(3):17–28, 1994.

[83] W. Ryan and S. Lin. *Channel codes: classical and modern*. Cambridge University Press, 2009.

[84] 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*, volume 6, pages 325–336. VLDB Endowment, 2013.

[85] A. Severinson, A. G. i Amat, and E. Rosnes. Block-diagonal and lt codes for distributed computing with straggling servers. *IEEE Transactions on Communications*, 67(3):1739–1753, 2018.

[86] K. Shanmugam, D. S. Papailiopoulos, A. G. Dimakis, and G. Caire. A repair framework for scalar MDS codes. *IEEE Journal on Selected Areas in Communications*, 32(5):998–1007, 2014.

[87] U. Sheth, S. Dutta, M. Chaudhari, H. Jeong, Y. Yang, J. Kohonen, T. Roos, and P. Grover. An application of storage-optimal matdot codes for coded matrix multiplication: Fast k-nearest neighbors estimation. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1113–1120. IEEE, 2018.

[88] G. Suh, K. Lee, and C. Suh. Matrix sparsification for coded matrix multiplication. In *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1271–1278. IEEE, 2017.

[89] H. Sun and S. A. Jafar. The capacity of robust private information retrieval with colluding databases. *IEEE Transactions on Information Theory*, 64(4):2361–2370, 2017.

[90] R. Tajeddine and S. El Rouayheb. Robust private information retrieval on coded data. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 1903–1907. IEEE, 2017.

[91] R. Tajeddine, O. W. Gnilke, D. Karpuk, R. Freij-Hollanti, and C. Hollanti. Robust private information retrieval from coded systems with byzantine and colluding servers. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 2451–2455. IEEE, 2018.

[92] I. Tamo and A. Barg. A family of optimal locally recoverable codes. *IEEE Transactions on Information Theory*, 60(8):4661–4676, 2014.

[93] I. Tamo, Z. Wang, and J. Bruck. Zigzag codes: MDS array codes with optimal rebuilding. *IEEE Trans. Inf. Theory*, 59(3):1597–1616, March 2013.

[94] I. Tamo, M. Ye, and A. Barg. Optimal repair of Reed-Solomon codes: achieving the cut-set bound. *arXiv preprint arXiv:1706.00112*, 2017.

[95] I. Tamo, M. Ye, and A. Barg. Error correction based on partial information. *IEEE Transactions on Information Theory*, 66(3):1396–1404, 2019.

[96] H. Wang and D. S. Wong. On secret reconstruction in secret sharing schemes. *IEEE Transactions on Information Theory*, 54(1):473–480, 2008.

[97] S. Wang, J. Liu, and N. Shroff. Coded sparse matrix multiplication. *arXiv preprint arXiv:1802.03430*, 2018.

[98] S. Wang, J. Liu, N. Shroff, and P. Yang. Fundamental limits of coded linear transform. *arXiv preprint arXiv:1804.09791*, 2018.

[99] Z. Wang, I. Tamo, and J. Bruck. Explicit minimum storage regenerating codes. *IEEE Transactions on Information Theory*, 62(8):4466–4480, Aug 2016.

[100] N. Woolsey, R.-R. Chen, and M. Ji. Heterogeneous computation assignments in coded elastic computing. *arXiv preprint arXiv:2001.04005*, 2020.

[101] N. Woolsey, R.-R. Chen, and M. Ji. Coded elastic computing on machines with heterogeneous storage and computation speed. *IEEE Transactions on Communications*, pages 1–1, 2021.

[102] Y. Yang, M. Interlandi, P. Grover, S. Kar, S. Amizadeh, and M. Weimer. Coded elastic computing. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2654–2658. IEEE, 2019.

[103] M. Ye and A. Barg. Explicit constructions of MDS array codes and RS codes with optimal repair bandwidth. In *Information Theory (ISIT), 2016 IEEE International Symposium on*, pages 1202–1206. IEEE, 2016.

[104] M. Ye and A. Barg. Explicit constructions of high-rate MDS array codes with optimal repair bandwidth. *IEEE Transactions on Information Theory*, 63(4):2001–2014, 2017.

[105] M. Ye and A. Barg. Explicit constructions of optimal-access MDS codes with nearly optimal sub-packetization. *IEEE Transactions on Information Theory*, 63(10):6307–6317, Oct 2017.

[106] M. Ye and A. Barg. Repairing Reed-Solomon codes: universally achieving the cut-set bound for any number of erasures. *arXiv preprint arXiv:1710.07216*, 2017.

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

[108] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr. Coded fourier transform. *arXiv preprint arXiv:1710.06471*, 2017.

[109] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication. *arXiv preprint arXiv:1705.10464*, 2017.

[110] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr. Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding. *IEEE Transactions on Information Theory*, 66(3):1920–1933, 2020.

[111] X. Zhang. Modified generalized integrated interleaved codes for local erasure recovery. *IEEE Communications Letters*, 21(6):1241–1244, 2017.

[112] Z. Zhang, Y. M. Chee, S. Ling, M. Liu, and H. Wang. Threshold changeable secret sharing schemes revisited. *Theoretical Computer Science*, 418:106–115, 2012.

[113] M. Zorgui and Z. Wang. On the achievability region of regenerating codes for multiple erasures. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 2067–2071. IEEE, 2018.

[114] M. Zorgui and Z. Wang. Centralized multi-node repair regenerating codes. *IEEE Transactions on Information Theory*, 2019.