Storage Codes with Flexible Number of Nodes

Weiqi Li, Zhiying Wang, Taiting Lu, and Hamid Jafarkhani

Abstract

This paper presents flexible storage codes, a class of error-correcting codes that can recover information from a flexible number of storage nodes. As a result, one can make better use of the available storage nodes in the presence of unpredictable node failures and reduce the data access latency. Assume a storage system encodes $k\ell$ information symbols over a finite field $\mathbb{F}$ into n nodes, each of size ℓ symbols. The code is parameterized by a set of tuples $\{(R_j, \ell_j) : 1 \leq j \leq a\}$, satisfying $\ell_1 < \ell_2 < ... < \ell_a = \ell$ and $R_1 > R_2 > \cdots > R_a$, such that the information symbols can be reconstructed from any R_j nodes, each node accessing ℓ_j symbols, for any $1 \leq j \leq a$. In other words, the code allows a flexible number of nodes for decoding to accommodate the variance in the data access time of the nodes. Code constructions are presented for different storage scenarios, 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.

I. 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 paper, 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 $(C_1, ..., C_n), C_i = (C_{1,i}, \ldots, C_{\ell,i})^T \in \mathbb{F}^\ell$, 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. R is called the recovery threshold. We define the download time of the slowest node among the R nodes as the *data access latency*. In

The authors are with Center for Pervasive Communications and Computing (CPCC), University of California, Irvine, USA. Emails: {weiqil4, zhiying, taitingl, hamidj}@uci.edu. The paper is partially presented in [1].

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 [2]. 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 code* 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

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

and if we take ℓ_j particular coordinates of each node, denoted by $(C_{m_1,i}, \ldots, C_{m_{\ell_j},i})^T \in \mathbb{F}^{\ell_j}, 1 \leq i \leq n$, we can recover the entire information from any R_j nodes. Here, k_j can be viewed as the code dimension when only ℓ_j coordinates are considered in each node.

For example, *flexible maximum distance separable (MDS) codes* are codes satisfying the singleton bound for every $1 \leq j \leq a$, namely, $R_j = k_j$. Fig. 1 shows an example. It is easy to see that the flexible code in the example has a better expected latency than that of 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 from any 3 nodes, or 3 symbols from any 2 nodes are delivered. Therefore, the latency of the flexible code is the minimum of the two fixed codes.

Naively, the flexible (n, k, ℓ) MDS code can be achieved by an $(n\ell, k\ell, 1)$ MDS code, where ℓ codeword symbols are viewed as one node in the flexible (n, k, ℓ) MDS code. However, by doing so, a large field with a size of at least $n\ell$ is required. The complexity of such a code is more than that of the codes that require smaller field sizes. While several works have attempted to improve the computation over large fields [3]–[6], the large field size still significantly increases the memory and the time complexity of encoding and decoding [7], [8] due to the need for large look-up tables. For example, 64 KB of memory is required for a standard multiplication table in $GF(2^8)$, while 8 GB is required for $GF(2^{16})$. With limited look-up tables, the computing speed is much slower in large fields [3]. Efforts to alleviate the high cost of memory and computing complexity in larger fields can be seen in [3]–[6].

Several constructions of flexible MDS codes exist in the literature, though intended for different application scenarios, including error-correcting codes [9], universally decodable matrices [10], [11],

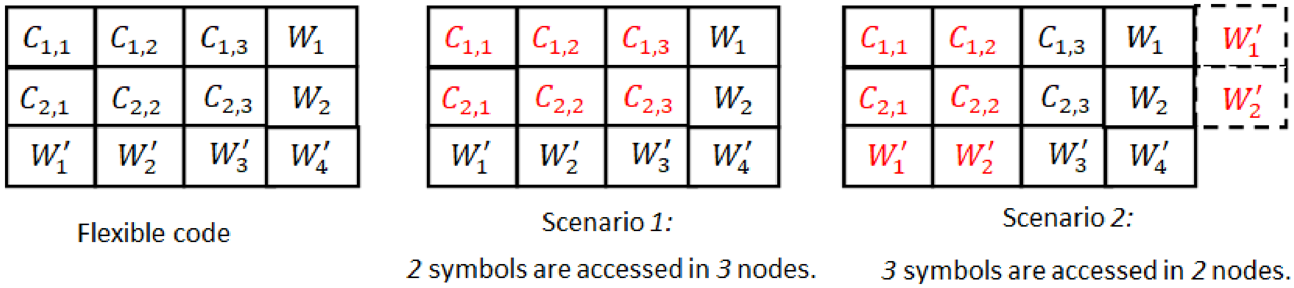

Fig. 1. Example of a $(4, 2, 3)$ flexible MDS code over $GF(5)$. $C_{1,1}, C_{1,2}, C_{1,3}, C_{2,1}, C_{2,2}, C_{2,3}$ are the 6 information symbols. We set $W_1 = C_{1,1} + C_{1,2} + C_{1,3}, W_1' = C_{1,1} + 2C_{1,2} + 3C_{1,3}$ as the parities for $C_{1,1}, C_{1,2}, C_{1,3}$, and $W_2 = C_{2,1} + C_{2,2} + C_{2,3}, W_2' = C_{2,1} + 2C_{2,2} + 3C_{2,3}$ as 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.

secrete sharing [12], and private information retrieval [13]. However, flexible constructions remain an open problem 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. Single failure recovery is essential for efficient storage management of distributed systems [14], [15], while mixed types of failures are common in solid-state drives [16]. In this paper, we provide a framework that can produce flexible storage codes for different code families. The main contributions of the paper 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. The framework keeps the same code rate k/n as the original fixed code. Therefore, if the original fixed code has an optimal code rate, our constructions are also optimal. Furthermore, the application of our framework to the three types of codes listed below provides optimal code rates and optimal recovery thresholds.

- **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 (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 in [17].

• **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) flexible PMDS code parameterized by $\{(R_j, k_j, \ell_j) : 1 \le j \le a\}$ satisfying (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 in [18].

• **Flexible MSR (minimum storage regenerating) codes** are a 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 [19] and [20].

• **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 with 6% improvement in latency 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 [21] to minimize a cost function such as a linear combination of bandwidth, delay or the number of hops. Flexible MDS codes were proposed in [9] to 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 [9] is to reduce the bandwidth instead of the number of accessed symbols. Universally decodable matrices (UDM) [10], [11] can also be used for the flexible MDS problem. UDM is a generalization of the flexible MDS code where the decoder can obtain different number of symbols from the nodes. In particular, from the first v_i symbols of node C_i, for any $v_i, 1 \le i \le n$ such that $\sum_{i=1}^{n} v_i \ge k\ell$, the entire information can be recovered. Flexibility problems are also considered for secret sharing [12], [22]–[24] and private information retrieval [13], [25]–[29], such that the number of available nodes is flexible. The constructions in [12] and [13] are equivalent to each other and they achieve 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. The schemes in [9]–[13] achieve the optimal field size of $|\mathbb{F}| = n$.

There are several works on latency and flexibility in the literature in distributed coded computing [30]–[33]. Specifically, fixed MDS codes are well studied [30], [31], 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 [30], [31], 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 [32], [34], [35]. 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 paper is organized as follows: In Section II, we present the definition and the construction of our flexible storage codes. We present the flexible LRC, PMDS, and MSR codes in Sections III-A, III-B, and III-C, respectively. In Section IV, we analyze the latency of data access using our flexible codes and compare it with that of fixed codes. Conclusion remarks are made in Section V.

Notation. For any integer $a \geq 1$, $[a]$ denotes the set $\{1, 2, \dots, a\}$. For a matrix A over a field $\mathbb{F}$, $rank(A)$ denotes its rank. For a set of matrices $A_1, A_2, \dots, A_n$ of size $x \times y$, $diag(A_1, A_2, \dots, A_n)$ denotes the corresponding diagonal matrix of size $nx \times ny$.

II. The framework for flexible codes

In this section, we define flexible storage codes and provide the framework 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 III.

First, we define flexible storage codes. In our illustrations, the codeword is represented by an $\ell \times n$ array over $\mathbb{F}$, denoted by $C \in \left(\mathbb{F}^{\ell}\right)^n$, 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 $a > 0$, $0 < \ell_1 < \ell_2 < \cdots < \ell_a = \ell$, and *recovery thresholds* $R_j \in [n]$, for $j \in [a]$. Let the *decoding columns* $\mathcal{R}_j \subseteq [n]$ be a subset of R_j columns and the *decoding rows* $\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_{R_j} \subseteq [\ell]$ be subsets of rows, each with size ℓ_j. Denote by $C\mid_{\mathcal{R}_j:\mathcal{I}_1,\mathcal{I}_2,\dots,\mathcal{I}_{R_j}}$, 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 $\mathcal{R}_j, \dots,$ and the rows $\mathcal{I}_{R_j}$ in the last column of $\mathcal{R}_j$. The information will be reconstructed from this subarray.

The information consists of $k\ell$ symbols. We choose $k_1 > k_2 > \cdots > k_a = k$ such that $k_j \ell_j = k\ell$ for all $j \in [a]$. Informally, k_j represents the dimension of the code when C is limited to a sub-packetization of ℓ_j. For flexible MDS codes, flexible MSR codes, and flexible PMDS codes, we have

$$R_j = k_j.$$

and we simply omit the parameter R_j.

Since the minimum distance of LRC codes is lower bounded by $n - k_j - \left\lceil \frac{k_j}{r} \right\rceil + 2$ [14], where r is the locality, we require flexible LRC codes to be optimal and satisfy

$$R_j = k_j + \left\lceil \frac{k_j}{r} \right\rceil - 1.$$

Definition 1. 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. That is, the code is defined by

- an encoding function $\mathcal{E} : \left(\mathbb{F}^{\ell}\right)^k \rightarrow \left(\mathbb{F}^{\ell}\right)^n$,
- decoding functions $\mathcal{D}_{\mathcal{R}_j} : \left(\mathbb{F}^{\ell_j}\right)^{R_j} \rightarrow \left(\mathbb{F}^{\ell}\right)^k$, 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 1, such that $R_j = k_j$.

In the rest of this section, we first prove in Lemma 1 that the the example in Fig. 1 is a flexible MDS code. Then, the general flexible code framework is presented in Construction 1, based on which Fig. 1 is designed. Afterwards, we prove in Theorem 1 that this framework can provide a flexible MDS code for arbitrary parameters.

Lemma 1. Fig. 1 presents 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. 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 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. 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 the 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 are 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 I

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 I, we construct (n,k,ℓ) flexible storage codes with $\{(k_j,\ell_j) : 1 \le j \le a\}$, 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 called storage nodes are stored and the extra parities are auxiliary. Let us set $\ell_0 = 0$. We have a layers and Layer j is an $(n+k_j-k_a, k_j, \ell_j - \ell_{j-1})$ code

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

with $j \in [a]$, 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 $j, j \ge 2$, is encoded from extra parities $C'_{j',i}$, for

$j' \in [j-1], k_j - k_a + 1 \leq i \leq k_{j-1} - k_a$. As a sanity check,

$$\sum_{j'=1}^{j-1}(k_{j-1} - k_j)(\ell_{j'} - \ell_{j'-1}) = (k_{j-1} - k_j)(\ell_{j-1} - \ell_0) = k_j(\ell_j - \ell_{j-1})$$

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 $k_{j-1}\ell_{j-1} = k_j\ell_j$.

Remark 1. It can be seen that the code rate of our (n, k, ℓ) flexible code remains the same as the original fixed (n, k, ℓ) code since the number of information symbols is $k_1\ell_1 = k_a\ell_a = k\ell$.

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.

The (n, k, ℓ) flexible MDS codes parametrized by $\{(R_j, k_j, \ell_j) : 1 \leq j \leq a\}$ is constructed by applying a set of $(n + k_j - k_a, k_j, \ell_j - \ell_{j-1}), j \in [a], \ell_0 = 0$ MDS codes over $\mathbb{F}$ to Construction 1. Namely, 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. Next, we prove that the code satisfies Definition 1 and $R_j = k_j$. That is, we can recover the entire information from any k_j nodes, each node accessing its first ℓ_j symbols.

Theorem 1. 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 1 and $R_j = k_j$.

Proof: 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 information symbols can be recovered.

We prove by induction that we are able to decode Layer j' for all $j' = j, j-1, \ldots, 1$. As a result, after decoding Layer $j' = 1$, we can recover all information symbols.

Base case: For Layer 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 $k_{j'} - k_j$ extra parities $C'_{j',i}, k_j - k_a + 1 \leq i \leq k_{j'} - k_a$. Together with the k_j nodes that 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 [36] and vector codes [37]. In the case of vector codes, the codeword symbols of the MDS codes are from a vector space rather than a finite field.

III. 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 have optimal code rate k/n and optimal recovery threshold R. They 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.

A. Flexible LRC

An (n,k,ℓ,r) LRC code is defined as a code with length n, dimension k, sub-packetization size ℓ, and locality r. Locality here means that 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 [14], [38]–[41]. The minimum Hamming distance of an (n,k,ℓ,r) LRC code is lower bounded in [14] as

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

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 $\ell=1$. For k divisible by r, and n divisible by $r+1$, Tamo and Barg [17] constructed optimal (n,k,r) LRC codes that encode the k information symbols into

$$C=[C_{1,1},C_{1,2},\ldots,C_{1,r+1},...,C_{\frac{n}{r+1},1},C_{\frac{n}{r+1},2},...,C_{\frac{n}{r+1},r+1}].$$

Here, each group $\{C_{m,i}: i\in[r+1]\}, m\in[\frac{n}{r+1}]$, is an MDS code with dimension r and the code C has a minimum distance of $n-k-\frac{k}{r}+2$, i.e., we can decode all 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 1, such that all the symbols of any node can be recovered by reading at most r other nodes and

$$R_j=k_j+\left\lceil\frac{k_j}{r}\right\rceil-1.$$

The above R_j matches the minimum distance lower bound (2). As a result, our definition of flexible LRC code implies optimal minimum Hamming distance when we consider ℓ_j symbols at each node.

In the following, we present flexible LRC codes in Construction 2. Then, Table II illustrates the structure of our code. We prove in Theorem 2 that Construction 2 leads to flexible LRC codes. When the specific LRC in [17] is applied to each layer, Table II is further explained in Example 1 at the end of this subsection.

Code overview. The flexible LRC code is based on Construction 1, where each layer consists of LRC codes. 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 accesses.

Construction 2. Let n be divisible by $r+1$ and all $k_j, j \in [a]$ be divisible by r. We apply a set of optimal LRC codes by groups over $\mathbb{F}$ with parameters $(n+(k_j-k_a)\frac{r+1}{r}, k_j, r), j \in [a]$ to Construction 1.

In Layer j, we apply an $(n+(k_j-k_a)\frac{r+1}{r}, k_j, r), j \in [a]$ 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 $(n, k_j, r), j \in [a]$ optimal LRC code. Then, in the remaining $\frac{k_j-k_a}{r}$ groups, we pick r nodes in each group, containing $k_j - k_a$ nodes, as extra parities.

Table II shows an example of $(n = 12, k = 4, \ell = 3, r = 2)$ flexible LRC code parameterized by $\{(R_1 = 8, k_1 = 6, \ell_1 = 2), (R_2 = 5, k_2 = 4, \ell_2 = 3)\}$. In this code, Rows 1 and 2 are $(n+(k_1-k_2)\frac{r+1}{r} = 15, k_1 = 6, r = 2)$ LRC codes encoded from the information and one extra group is generated in each row. We take 4 extra parities from the extra groups, which are encoded into the $(n = 12, k_2 = 4, r = 2)$ LRC code in Row 3. In this code, 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. It will be seen in Theorem 2 that to recover the entire information, we require either any $R_1 = 8$ nodes, each accessing the first $\ell_1 = 2$ symbols, or any $R_2 = 5$ nodes, each accessing all $\ell_2 = 3$ symbols.

Theorem 2. Construction 2 results in an (n, k, ℓ, r) flexible LRC code parameterized by $\{(R_j, k_j, \ell_j) : 1 \le j \le a\}$. Moreover, when only the first ℓ_j symbols are considered at each node, any single node failure can also be recovered from r helpers.

TABLE II

CONSTRUCTION OF $(n = 12, k = 4, \ell = 3, r = 2)$ FLEXIBLE LRC CODE

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

Proof: We first prove the reconstruction of all information symbols from ℓ_j symbols of $R_j = k_j + \frac{k_j}{r} - 1$ nodes, for any $j \in [a]$. Then we prove the locality.

Reconstruction: We prove by induction that for $j' = j, j-1, \ldots, 1$, we can decode Layer j'. Therefore, all information symbols can be recovered after decoding Layer $j' = 1$.

Base case: From Layer j, since each row 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 minimum Hamming distance property of the optimal LRC codes.

Induction step: Suppose that Layers $j'+1, j+2, \ldots, j$ are decoded. We prove that Layer j' can be decoded. By the construction, $k_{j'} - k_j$ extra parities (from $\frac{k_{j'}-k_j}{r}$ extra groups) in each row of Layer j' can be obtained from the decoded layers. By Construction 2, the extra parities in Layer j' consist of r parity symbols in each extra group. Thus, according to locality, the remaining symbol in each of the $\frac{k_{j'}-k_j}{r}$ extra groups in each row of Layer j' can be recovered. In total, we get additional $(k_{j'} - k_j)\frac{r+1}{r}$ symbols in each row of Layer j' from the extra parities. Together with the R_j accessed symbols in each row of Layer j', we get $R_{j'}$ symbols and we are able to decode Layer j'.

Locality: Since each row is encoded as an LRC code with locality r, the code restricted to the first ℓ_j rows also has locality r. ■

When applying the LRC codes in [17], our flexible LRC code requires a finite field of size at least $n + (k_1 - k_a)\frac{r+1}{r}$. Below, we show the encoding, the reconstruction, and the locality for the code in Table II using [17].

Example 1. We set $(n, k, l, r) = (12, 4, 3, 2)$, $(R_1, k_1, \ell_1) = (8, 6, 2), (R_2, k_2, \ell_2) = (5, 4, 3)$. 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 [17].

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

$$\begin{bmatrix} C_{1,1,1} & C_{1,1,2} & \cdots & C_{1,1,12} \\ C_{1,2,1} & C_{1,2,2} & \cdots & C_{1,2,12} \\ C_{2,1,1} & C_{2,1,2} & \cdots & C_{2,1,12} \end{bmatrix}, \tag{3}$$

where every entry in Row m will be constructed as $f_m(x)$ for some polynomial $f_m(\cdot)$ and some evaluation point $x \in \mathbb{F}$ as below, $m = 1, 2, 3$.

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

According to [17], we define $g(x) = x^3$ and one can check that $g(x)$ is a constant for each group A_i, $i \in [5]$. Then, the first 2 rows are encoded with

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

The last row is encoded with

$$f_3(x) = \big(f_1(\alpha^4) + f_1(\alpha^9)g(x)\big) + x\big(f_2(\alpha^4) + f_2(\alpha^9)g(x)\big). \tag{5}$$

For each group, since $g(x)$ is a constant, $f_m(x), m \in [3]$ can be viewed as a polynomial of degree 1. 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.

Reconstruction with $\ell_1 = 2, R_1 = 8$: 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.

Reconstruction with $\ell_2 = 3, R_2 = 5$: Since $f_3(x)$ 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 $R_2 = 5$ other evaluations of $f_1(x)$ and $f_2(x)$ obtained in Rows 1 and 2, we are able to decode all information symbols.

B. Flexible PMDS codes

PMDS codes are first introduced in [16] to overcome mixed types of failures in Redundant Arrays of Independent Disks (RAID) systems using solid-state drives. A code consisting of an $\ell \times n$ array is called an (n, k, ℓ, s) PMDS code if every row is an (n, k) MDS code and it can tolerate $n - k$ node or column failures and s additional arbitrary symbol failures in the code.

Let $\ell_0 = 0$ and $\{(k_j, \ell_j) : 1 \leq j \leq a\}$ satisfy (1). We define an (n, k, ℓ, s) *flexible PMDS code* parameterized by $\{(k_j, \ell_j) : 1 \leq j \leq a\}$ to be an $\ell \times n$ array such that any row in the range $\ell_{j-1} + 1$ to ℓ_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 1, the number of information symbols for a flexible PMDS code is at most $k\ell - s \triangleq K$.

Example 2. We demonstrate the information reconstruction requirement of the $(5, 3, 4, 2)$ flexible PMDS code with $\{(k_1, \ell_1), (k_2, \ell_2)\} = \{(4, 3), (3, 4)\}$ in Table III. 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 should be independent and sufficient to reconstruct the entire information.

TABLE III

AN EXAMPLE OF $(5, 3, 4, 2)$ FLEXIBLE PMDS CODE WITH $\{(k_1, \ell_1), (k_2, \ell_2)\} = \{(4, 3), (3, 4)\}$.

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

Code overview. To tolerate additional symbol failures, the fixed PMDS code in [18] 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 also encodes the information using Gabidulin code into auxiliary symbols, which are allocated to each layer according to $k_j, j \in [a]$. MDS codes with different dimensions are then applied to each row, thus ensuring flexible information reconstruction.

We first introduce the construction in [18] and then show how to apply it to flexible PMDS codes.

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 $u_i \in \mathbb{F}, i = 0, 1, ..., K-1$ are the information symbols. 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 [18, Construction 1], the (n,k,ℓ,s) codeword is an $\ell\times n$ matrix over $\mathbb{F}=GF(q^{k\ell})$ shown below:

$$\begin{bmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,n} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ C_{\ell,1} & C_{\ell,2} & \cdots & C_{\ell,n} \end{bmatrix}, \tag{6}$$

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. For each row m, $m\in[\ell]$,

$$[C_{m,k+1},...,C_{m,n}]=[C_{m,1},...,C_{m,k}]G_{\text{MDS}}, \tag{7}$$

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 [18, Lemma 2] that t_m symbols in Row $m, m\in[\ell]$, are 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 $t_m\le k$ and

$$\sum_{m=1}^{\ell}\min(t_m,k)=\sum_{m=1}^{\ell}t_m=\ell k-s=K. \tag{8}$$

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 (6) with a different dimension.

Construction 3. Fix (n,k,ℓ,s) and $\{(k_j,\ell_j):1\le j\le a\}$ satisfying (1). Assume there exists an (N,K) Gabidulin code over $GF(q^N)$ and a set of (n,k_j) systematic MDS codes over $GF(q)$, where $N=\sum_{j=1}^{a}k_j(\ell_j-\ell_{j-1})$, $\ell_0=0$, $K=\ell k-s$. We construct a storage code over $GF(q^N)$ that encodes K information symbols into an $\ell\times n$ codeword array.

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

$$[C_{j,m_j,k_j+1},...,C_{j,m_j,n}]=[C_{j,m_j,1},...,C_{j,m_j,k_j}]G_{n,k_j}, \tag{9}$$

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

Theorem 3. Construction 3 results in an (n, k, ℓ, s) flexible PMDS code over $GF(q^N)$ parameterized by $\{(k_j, \ell_j) : 1 \leq j \leq a\}$ satisfying (1).

Proof: It is obvious that each row in Layer j is an (n, k_j) MDS code due to (9). We will prove that we can decode the information from any $n-k_j, j \in [a]$, failures by accessing 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 [18, Construction 1]. From [18, 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 $f(x)$. ■

C. *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 [15] as

$$b \geq \frac{n-1}{n-k}. \tag{10}$$

An MDS code achieving the MSR bound is called an MSR code. MSR vector codes are well studied in [19], [42]–[48], where each symbol is a vector. As one of the most popular codes in practical systems, Reed-Solomon (RS) codes and their repair are studied in [8], [20], [49]–[51], where each symbol is a scalar.

We have shown in Theorem 1 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

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

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 1, such that $R_j = k_j$ and a single node failure is recovered using a repair bandwidth satisfying the MSR bound (10) with equality.

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 the minimum 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.

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

Example 3. We construct an $(n, k, \ell) = (4, 2, 3)$ flexible MSR code parameterized by $(k_1, \ell_1) = (3, 2)$ and $(k_2, \ell_2) = (2, 3)$.

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

$$H = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & h_{1,4} \\ h_{2,1} & h_{2,2} & h_{2,3} & h_{2,4} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{11}$$

where each $h_{i,j}$ is a 2×2 matrix over $\mathbb{F}$. A codeword symbol c_i is in $\mathbb{F}^2$, $i = 1, 2, 3, 4$, meaning c_i is a column vector of length 2 over $\mathbb{F}$. 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

$$S_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, S_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, S_3 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}, S_4 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{12}$$

It is easy to check that

$$rank \left(S_* \begin{bmatrix} h_{1,i} \\ h_{2,i} \end{bmatrix} \right) = \begin{cases} 2, i = * \\ 1, i \neq * \end{cases}. \tag{13}$$

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

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

which is one 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 below, each column being a node:

$$\begin{bmatrix} C_{1,1,1} & C_{1,1,2} & C_{1,1,3} & C_{1,1,4} \\ C_{1,2,1} & C_{1,2,2} & C_{1,2,3} & C_{1,2,4} \\ C_{2,1,1} & C_{2,1,2} & C_{2,1,3} & C_{2,1,4} \end{bmatrix}. \tag{14}$$

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 $i \in [4]$, $m_1 = 1,2, m_2 = 1$. 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

$$H_1 = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & h_{1,4} & h_{1,1} \\ h_{2,1} & h_{2,2} & h_{2,3} & h_{2,4} & \beta h_{2,1} \end{bmatrix}, \tag{15}$$

$$H_2 = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & h_{1,4} & h_{1,2} \\ h_{2,1} & h_{2,2} & h_{2,3} & h_{2,4} & \beta h_{2,2} \end{bmatrix}, \tag{16}$$

$$H_3 = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & h_{1,4} \\ \beta h_{2,1} & \beta h_{2,2} & h_{2,3} & h_{2,4} \end{bmatrix}. \tag{17}$$

The code is defined by

$$H_1 \times [C_{1,1,1}^T, C_{1,1,2}^T, C_{1,1,3}^T, C_{1,1,4}^T, C_{2,1,1}^T]^T = 0, \tag{18}$$

$$H_2 \times [C_{1,2,1}^T, C_{1,2,2}^T, C_{1,2,3}^T, C_{1,2,4}^T, C_{2,1,2}^T]^T = 0, \tag{19}$$

$$H_3 \times [C_{2,1,1}^T, C_{2,1,2}^T, C_{2,1,3}^T, C_{2,1,4}^T]^T = 0. \tag{20}$$

Next, we prove that it is an $(n,k,\ell) = (4,2,3)$ flexible MSR code parameterized by (k_j, ℓ_j) chosen from $\{(3,2),(2,3)\}$.

It is easy to check that the code defined by H_1 or H_2 is a $(5,3)$ MDS code and H_3 defines a $(4,2)$ MDS code. Thus, this code is the same as Construction 1 based on MDS codes and the flexible reconstruction of the entire information is shown in Theorem 1.

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

$$rank\left(S_*\begin{bmatrix}h_{1,i}\\h_{2,i}\end{bmatrix}\right)=rank\left(S_*\begin{bmatrix}h_{1,i}\\\beta h_{2,i}\end{bmatrix}\right)=\begin{cases}2, i=*\\1, i\neq *\end{cases}, \tag{21}$$

for $i=1,2$.

Then, we use the repair matrix S_* in (12) to repair the failed node $*$:

$$S_*\times H_1\times[C_{1,1,1}^T,C_{1,1,2}^T,C_{1,1,3}^T,C_{1,1,4}^T,C_{2,1,1}^T]^T=0, \tag{22}$$

$$S_*\times H_2\times[C_{1,2,1}^T,C_{1,2,2}^T,C_{1,2,3}^T,C_{1,2,4}^T,C_{2,1,2}^T]^T=0, \tag{23}$$

$$S_*\times H_3\times[C_{2,1,1}^T,C_{2,1,2}^T,C_{2,1,3}^T,C_{2,1,4}^T]^T=0. \tag{24}$$

Helper $i\in[4]$, $i\neq *$, transmits

$$S_*\begin{bmatrix}h_{1,i}\\h_{2,i}\end{bmatrix}C_{1,1,i}, \tag{25}$$

$$S_*\begin{bmatrix}h_{1,i}\\h_{2,i}\end{bmatrix}C_{1,2,i}, \tag{26}$$

$$S_*\begin{bmatrix}h_{1,i}\\\overline{\beta}h_{2,i}\end{bmatrix}C_{2,1,i}, \tag{27}$$

where $\overline{\beta}=\beta$ if $i=1,2$ and $\overline{\beta}=1$ if $i=3,4$. Note that to repair the failed node, in Eq. (22) and (23), 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 (27) or solved from Equation (24).

Then, from (13) and (21), it is clear that for any failed node, we only need one 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 (15) and (16) to be MSR codes, thus resulting in a smaller field. However, the rank condition (21) 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 [19] and the RS MSR code [20], we can construct flexible MSR codes.

1) Flexible MSR codes with parity check matrices: Below we present codes defined by parity check matrices similar to Example 3. We show in Theorem 4 that with certain choices of the parity check matrices, one obtains a flexible MSR code.

Construction 4. 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:

$$H_{j,m_j} = \begin{bmatrix} h_{j,m_j,1} & \cdots & h_{j,m_j,n} & g_{j,m_j,1} & \cdots & g_{j,m_j,k_j-k_a} \end{bmatrix}, \tag{28}$$

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

$$H_{j,m_j} \times [{C_{j,m_j,1}}^T, {C_{j,m_j,2}}^T, \cdots, {C_{j,m_j,n}}^T, {C'_{j,m_j,1}}^T, \cdots, {C'_{j,m_j,k_j-k_a}}^T]^T = 0, \tag{29}$$

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 $j < j' \le a$, we set

$$g_{j,x,y} = h_{j',x',y'}, \tag{30}$$

$$C'_{j,x,y} = C_{j',x',y'}. \tag{31}$$

Here, given $j, x \in [l_j - l_{j-1}], y$, we set j' such that $k_{j'} - k_a + 1 \le y \le k_{j'-1} - k_a$, and set

$$x' = \lfloor \frac{x(k_{j'-1} - k_{j'}) + y}{k_{j'}} \rfloor, \tag{32}$$

$$y' = (x(k_{j'-1} - k_{j'}) + y) \text{ mod } k_{j'}, \tag{33}$$

where "mod" denotes the modulo operation.

For instance, in Example 3, 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 4. Assume the parity check matrices of Construction 4 in (28) satisfy

1). [MDS condition.] The codes defined by (28) 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

$$rank(S_* h_{j,m_j,i}) = \begin{cases} L, i = * \\ \frac{L}{r}, i \neq * \end{cases}, \ i \in [n]. \tag{34}$$

Then, the code defined by Construction 4 is a flexible MSR code.

Proof: 1). If the MDS property is satisfied, Construction 4 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 1.

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

$$S_* \times H_{j,m_j} \times [{C_{j,m_j,1}}^T, {C_{j,m_j,2}}^T, \cdots, {C_{j,m_j,n}}^T, {C'_{j,m_j,1}}^T, \cdots, {C'_{j,m_j,k_j-k_a}}^T]^T = 0. \tag{35}$$

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 parity check sub-matrices, and the corresponding required information can be retrieved from the lower layers. Thus, the failed node can be repaired from $n-1$ helpers.

Clearly from (34), 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 [19] to show how to construct the flexible MSR codes satisfying the conditions in Theorem 4. The code structure in one row is similar to [52].

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 [19] can be represented as:

$$H = \begin{bmatrix} I & I & \cdots & I \\ A_1 & A_2 & \cdots & A_n \\ \vdots & \vdots & \ddots & \vdots \\ A_1^{r-1} & A_2^{r-1} & \cdots & A_n^{r-1} \end{bmatrix}, \tag{36}$$

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 zeros except the z-th element which is equal to 1. We write the r-ary expansion of z as $z = (z_n, z_{n-1}, \ldots, z_1)$, where $0 \le z_i \le r-1$ 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 [19] and [52, Sec. IV-A] as a diagonal block matrix:

$$S_* = \text{Diag}(D_*, D_*, ..., D_*), \tag{37}$$

with $\frac{L}{r} \times L$ matrix D_*. It is shown that

$$rank\left(S_*\begin{bmatrix} I \\ A_i \\ \vdots \\ A_i^{r-1}\end{bmatrix}\right) = rank\begin{pmatrix} D_* \\ D_*A_i \\ \vdots \\ D_*A_i^{r-1}\end{pmatrix} = \begin{cases} L, i = * \\ \frac{L}{r}, i \neq * \end{cases}. \tag{38}$$

Here, for $0 \leq x \leq r^{n-1} - 1, 0 \leq y \leq r^n - 1$, the (x, y)-th entry of D_* is equal to 1 if the r-ary expansion of x and y satisfies $(x_{n-1}, x_{n-1}, \ldots, x_1) = (y_n, y_{n-1}, \ldots, y_{i+1}, y_{i-1}, \ldots, y_1)$ and otherwise it is equal to 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 $\mathbb{F}$ [8, Lemma 1]. Now, we define for the storage nodes (the first n nodes)

$$h_{j,m_j,i} = \begin{bmatrix} I \\ \beta_{j,m_j}A_i \\ \beta_{j,m_j}^2A_i^2 \\ \vdots \\ \beta_{j,m_j}^{r-1}A_i^{r-1}\end{bmatrix}, \tag{39}$$

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 (32) and (33). Also, notice that A_i might appear 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 1. With parity check matrices defined by (39) and Condition 1, Construction 4 is a flexible MSR code.

Proof: We will prove the construction is flexible MSR using Theorem 4. We consider the m_j-th row in Layer j, $j \in [a], m_j \in [\ell_j - \ell_{j-1}]$.

1) [MDS condition.] For the codeword $(c_1^T, c_2^T, ..., c_{n+k_j-k_a}^T)^T$ 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

$$\begin{bmatrix} 1 & \cdots & 1 & 1 & \cdots & 1 \\ \beta_{j,m_j}\lambda_{1,z_1} & \cdots & \beta_{j,m_j}\lambda_{n,z_n} & \alpha_1\gamma_1 & \cdots & \alpha_{k_j-k_a}\gamma_{k_j-k_a} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ (\beta_{j,m_j}\lambda_{1,z_1})^{r-1} & \cdots & (\beta_{j,m_j}\lambda_{n,z_n})^{r-1} & (\alpha_1\gamma_1)^{r-1} & \cdots & (\alpha_{k_j-k_a}\gamma_{k_j-k_a})^{r-1} \end{bmatrix} \begin{bmatrix} c_{1,z} \\ c_{2,z} \\ \vdots \\ c_{n+k_j-k_a,z} \end{bmatrix} = 0. \tag{40}$$

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 (32) and (33) with $x = m_j$. Next, we show (40) corresponds to a Vandermonde matrix of full rank, 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, which is MDS. We just need to show that any two entries in the second row of the $r \times (n+k_j-k_a)$ matrix in (40) are distinct. Notice that each entry is the product of an additional coefficient and a λ variable (or a γ variable). There are three cases: (i) If the λ or the γ values are identical, by Condition 1, their additional coefficients differ. So, these two entries are distinct. (ii) If the λ or the γ values are distinct and the additional coefficients are identical, then the two entries are distinct. (iii) 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 $z = 0, 1, \ldots, L-1$ 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 (38) and (39),

$$rank(S_* h_{j,m_j,i}) = rank \begin{pmatrix} D_* \\ D_*\beta A_i \\ \vdots \\ D_*\beta^{r-1}A_i^{r-1} \end{pmatrix} = rank \begin{pmatrix} D_* \\ D_* A_i \\ \vdots \\ D_* A_i^{r-1} \end{pmatrix} = \begin{cases} L, i = * \\ \frac{L}{r}, i \neq * \end{cases}. \tag{41}$$

Since the code satisfies the above two conditions, using Theorem 4, 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. The required field size can be chosen as $|\mathbb{F}| \geq t|\mathbb{E}|$, where t is equal to the number of additional coefficients. In the following, we propose two possible coefficient assignments. It should be noticed that one might find better assignments with smaller field sizes.

The simplest coefficient assignment sets 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 4, 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 for different y. In this case, $k_{j'} - k_a + 1 \leq y \leq k_{j'-1} - k_a$ and there will be at most $\lceil \frac{k_{j'-1}-k_{j'}}{k_{j'}} \rceil$ choices of y that make y' a constant in (33). 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 $t = 1 + \sum_{j=2}^{a} \lceil \frac{k_{j-1}-k_j}{k_j} \rceil$. Notice that $(k_{j-1} - k_j)\ell_{j-1} = k_j(\ell_j - \ell_{j-1})$ and we have

$$t = 1 + \sum_{j=2}^{a} \lceil \frac{k_{j-1} - k_j}{k_j} \rceil = 1 + \sum_{j=2}^{a} \lceil \frac{\ell_j - \ell_{j-1}}{\ell_{j-1}} \rceil \leq 1 + \sum_{j=2}^{a} (\ell_j - \ell_{j-1}) \leq \ell. \tag{42}$$

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 while $|\mathbb{F}| \geq a|\mathbb{E}|$.

Here, we briefly compare our construction with another flexible MSR construction in [9]. 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 [9] 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 $\mathbb{F}^{s(n-k)^n}$, with $|\mathbb{F}| \geq s(n-k)n$, where s is defined such that $s_j/s = \ell_j/\ell$ fraction of the information are downloaded in each node and 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. (42), the required field size of our construction is better than that of the construction in [9].

2) Flexible RS MSR codes: In this section, we introduce the construction of RS MSR codes.

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

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

where the evaluation points are defined as $\{\alpha_1, \alpha_2, \dots, \alpha_n\} \subseteq \mathbb{F}$, and $\deg()$ denotes the degree of a polynomial. The encoding polynomial is $f(x) = u_0 + u_1 x + \cdots + u_{k-1}x^{k-1}$, where $u_i \in \mathbb{F}, i = 0, 1, \dots, k-1$ 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, [49] and [8] show 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 $\{p_{*,v}(x), v \in [L]\}$ such that for the failed node $f(\alpha_*)$, $* \in [n]$,

$$rank_{\mathbb{B}}(\{p_{*,v}(\alpha_*) : v \in [L]\}) = L, \tag{43}$$

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

$$Tr_{\mathbb{F}/\mathbb{B}}(p_{*,v}(\alpha_i) f(\alpha_i)), v \in [L], \tag{44}$$

where the trace function $Tr_{\mathbb{F}/\mathbb{B}}(x)$ is a linear function such that for all $x \in \mathbb{F}$, $Tr_{\mathbb{F}/\mathbb{B}}(x) \in \mathbb{B}$ [53]. The repair bandwidth for the i-th helper is

$$b_i = rank_{\mathbb{B}}(\{p_{*,v}(\alpha_i) : v \in L\}) \tag{45}$$

symbols in $\mathbb{B}$.

The flexible RS MSR code construction is similar to Construction 4 based on parity check matrices, as presented below.

Construction 5. 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:

$$C_{j,m_j,i} = f_{j,m_j}(\alpha_{j,m_j,i}), \tag{46}$$

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

$$C'_{j,m_j,i} = f_{j,m_j}(\alpha_{j,m_j,i+n}), \tag{47}$$

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. We also arrange the extra parities the same way as in Construction 4. Specifically, for $C'_{j,x,y}$ in Layer j, $x \in [l_j - l_{j-1}]$, when $k_j - k_{j'-1} + 1 \le y \le k_j - k_{j'}$ for $j + 1 \le j' \le a$, 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 (32) (33). 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 5. Construction 5 is a flexible RS MSR 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 condition.] The same set of repair polynomials $p_{*,v}(x), * \in [n], v \in [L]$, can be used in each row such that:

$$rank_{\mathbb{B}}(\{p_{*,v}(\alpha_{j,m_j,*}) : v \in [L]\}) = L, \tag{48}$$

$$b_i = rank_{\mathbb{B}}(\{p_{*,v}(\alpha_{j,m_j,i}) : v \in [L]\}) = \frac{L}{r}, i \in [n] \backslash \{*\}. \tag{49}$$

Proof: 1). In the case when $\alpha_{j,m_j,i}, i \in [n + k_j - k_a]$ are distinct elements in $\mathbb{F}$, we have $\{f_{j,m_j}(\alpha_{j,m_j,i}), i \in [n + k_j - k_a]\}$ 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 5 is the same as Construction 1 by using the RS codes as the MDS codes. The flexible reconstruction property is shown in Theorem 1.

2). For the repair, since the extra parities share the same codeword symbols and evaluation points with the storage nodes in lower layers, from (44) we know that the transmission for repair is also the same. Thus, we only transmit information corresponding to the $(n - 1)$ storage nodes.

From (49), we know that in each row, each helper transmits L/r symbols, which is optimal. ■

We take the construction in [8] as the $RS(n + k_j - k_a, k_j), j \in [a]$ codes in Construction 5 to show how to construct flexible RS MSR codes.

In [8, Theorem 5], the RS code is defined in $\mathbb{F}$ with evaluation points chosen from the subset $\{\beta_1\alpha_i, \beta_2\alpha_i, ..., \beta_t\alpha_i, i \in [n]\}$ 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 $\mathbb{F}^*$ [8, Lemma 1]. For the repair polynomials $p_{*,v}(x)$ in [8],

$$rank_{\mathbb{B}}(\{p_{*,v}(\beta\alpha_i) : v \in [L]\}) = \begin{cases} L, i = *, \\ \frac{L}{r}, i \neq *, \end{cases} \tag{50}$$

for all β chosen from $\{\beta_1, ..., \beta_t\}$. The required subfield size in [8] is $|\mathbb{E}| \approx n^n$.

For Construction 5, 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 (32) and (33). We assign the additional coefficient β to satisfy Condition 1. Similar to Construction 4, we guarantee that in each row, the $n + k_j - k_a$ evaluation points are distinct and the total number of required β 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 2. With the RS code in [8], Construction 5 is a flexible RS MSR code.

Proof: We use Theorem 5 to prove that the code is a flexible RS MSR code.

1) [MDS condition.] We have assigned the evaluation points in each row as distinct elements in $\mathbb{F}$.
2) [Rank condition.] We know from (50) that the rank condition in Theorem 5 is satisfied. ■

IV. 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 paper 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 $T_{1,2,...,a} = \min(T_j, j \in [a])$ to represent the *latency*.

It is obvious that for the fixed code with the same failure tolerance level, i.e., $R = R_a, \ell = \ell_a$, the latency of the fixed code (T_a) is at least that of the flexible code:

$$T_{1,2,\dots,a} = \min(T_j, j \in [a]) \leq T_a, \tag{51}$$

and we reach the following remark.

Remark 2. 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

$$E(T_j) = \int_0^\infty \tau_j p_{R_j,\ell_j}(\tau_j) d\tau_j. \tag{52}$$

If a fixed code is adopted, one can optimize the expected latency and get an optimal pair (R^*, ℓ^*) for a given distribution [30], [31]. However, a flexible storage code still outperforms such an optimal fixed code in latency due to Remark 2. 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 [54], 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 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 the *beta distribution* [55]:

$$U_R \sim \text{Beta}(R, n+1-R, 0, t_{\text{pos}}), \tag{53}$$

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 is defined as

$$f(Y = y; \alpha, \beta, a, c) = \frac{(y-a)^{\alpha-1}(c-y)^{\beta-1}}{(c-a)^{\alpha+\beta-1} B(\alpha, \beta)}, \tag{54}$$

where

$$B(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1} dt \tag{55}$$

is the *Beta function*.

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

$$E(T_1) = \frac{R_1}{n+1} t_{\text{pos}} + \ell_1 t_{\text{trans}}. \tag{56}$$

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

$$E(T_2) = \frac{R_2}{n+1} t_{\text{pos}} + \ell_2 t_{\text{trans}}. \tag{57}$$

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

$$\Delta U = U_{R_1} - U_{R_2} \sim \text{Beta}(R_1 - R_2, n+1-(R_1-R_2), 0, t_{\text{pos}}). \tag{58}$$

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

$$\begin{aligned} E(T_{1,2}) &= E(min(T_1, T_2)) \\ &= E(T_1 | T_1 - T_2 \le 0) P(T_1 - T_2 \le 0) + E(T_2 | T_1 - T_2 > 0) P(T_1 - T_2 > 0) \\ &= E(T_1) - E(T_1 - T_2 | T_1 - T_2 > 0) P(T_1 - T_2 > 0) \\ &= \frac{R_1}{n+1} t_{\text{pos}} + \ell_1 t_{\text{trans}} - \int_{(\ell_2-\ell_1)t_{\text{trans}}}^{t_{\text{pos}}} [\Delta U - (\ell_2 - \ell_1) t_{\text{trans}}] f(\Delta U) d\Delta U, \end{aligned} \tag{59}$$

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

$$\begin{aligned} E(T_1 - T_{1,2}) &= \int_{(\ell_2-\ell_1)t_{\text{trans}}}^{t_{\text{pos}}} [\Delta U - (\ell_2 - \ell_1) t_{\text{trans}}] f(\Delta U) d\Delta U \\ &= \frac{a t_{\text{pos}}}{a+b} I_{1-x}(b, a+1) - (\ell_2 - \ell_1) t_{\text{trans}} I_{1-x}(b, a), \end{aligned} \tag{60}$$

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*:

$$I_x(a,b) = \frac{B(x;a,b)}{B(a,b)}, \tag{61}$$

with *incomplete beta function*

$$B(x;a,b) = \int_{t=0}^{x} t^{a-1}(1-t)^{b-1}dt. \tag{62}$$

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

$$E(T_1 - T_{1,2}) = (E(T_1) - E(T_2))I_{1-x}(b,a) + t_{\text{pos}}\frac{R_1 - R_2}{n+1}\frac{x^a(1-x)^b}{aB(a,b)}. \tag{63}$$

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

$$E(T_2 - T_{1,2}) = (E(T_2) - E(T_1))I_x(a,b) + t_{\text{pos}}\frac{R_1 - R_2}{n+1}\frac{x^a(1-x)^b}{aB(a,b)}. \tag{64}$$

From (56) and (57), 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 [30] and [31] based on the 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, a larger t_{trans} results in a larger R^*, while a larger t_{pos} results in a smaller R^*. In our flexible codes, we can always pick the best R_j over all $j \in [a]$ and thus provide a lower latency.

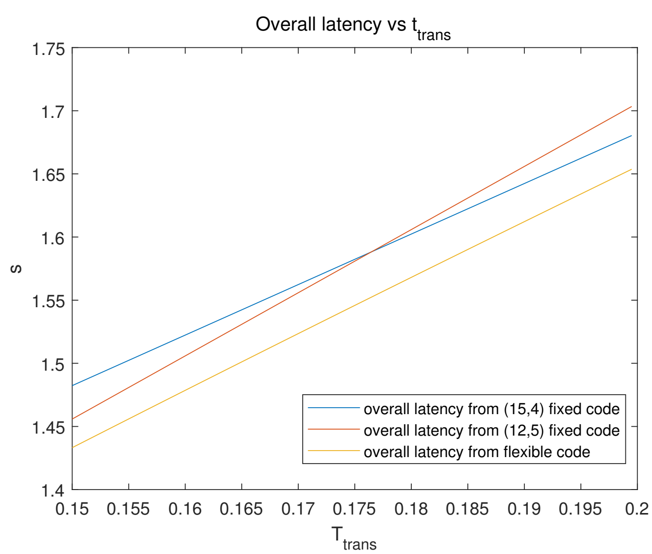

Fig. 2. Overall latency of fixed codes and flexible codes. $n = 16, R_1 = 15, R_2 = 12, \ell_1 = 4, \ell_2 = 5. t_{\text{pos}} = 1$.

Fig. 2 shows the overall latency of fixed codes and flexible recoverable codes. We fix the 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 [30]. 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. We can see a similar trend as that of Fig. 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.

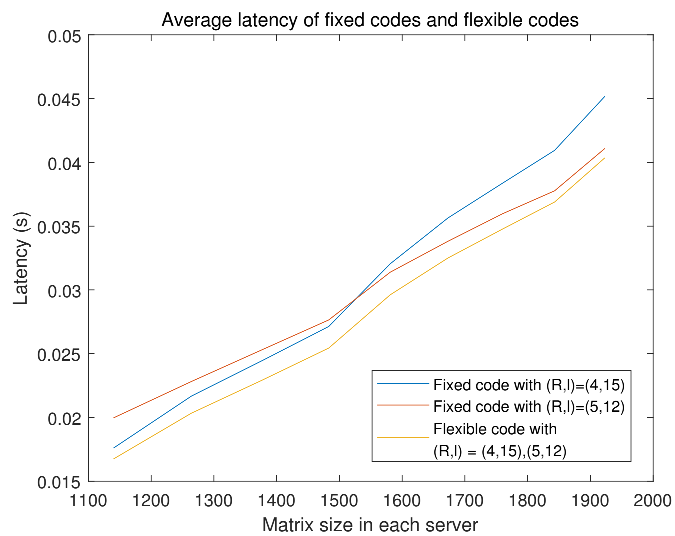

Fig. 3. Overall latency of fixed codes and flexible codes for matrix-vector multiplication in Amazon cluster. $n = 8, R_1 = 5, R_2 = 4, \ell_1 = 12, \ell_2 = 15$.

V. Conclusion

In this paper, 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 a smaller sub-packetization, and storage codes utilizing partial data transmission from each node similar to universally decodable matrices [10].

REFERENCES

[1] W. Li, T. Lu, Z. Wang, and H. Jafarkhani, "Flexible partial MDS codes," *Non-Volatile Memories Workshop (NVMW)*, 2021.

[2] 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*. IEEE, 2014, pp. 826–834.

[3] K. M. Greenan, E. L. Miller, and T. J. S. SJ, "Optimizing galois field arithmetic for diverse processor architectures and applications," in *2008 IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems*. IEEE, 2008, pp. 1–10.

[4] 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)*, vol. 8, no. 1, pp. 1–27, 2012.

[5] J. S. Plank, K. M. Greenan, and E. L. Miller, "Screaming fast galois field arithmetic using intel simd instructions." in *FAST*, 2013, pp. 299–306.

[6] S. B. Gashkov and I. S. Sergeev, "Complexity of computation in finite fields," *Journal of Mathematical Sciences*, vol. 191, no. 5, pp. 661–685, 2013.

[7] 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*, vol. 9, 2009, pp. 253–265.

[8] 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*, vol. 65, no. 9, pp. 5484–5502, 2019.

[9] I. Tamo, M. Ye, and A. Barg, "Error correction based on partial information," *IEEE Transactions on Information Theory*, vol. 66, no. 3, pp. 1396–1404, 2019.

[10] A. Ganesan and P. O. Vontobel, "On the existence of universally decodable matrices," *IEEE transactions on information theory*, vol. 53, no. 7, pp. 2572–2575, 2007.

[11] 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)*. IEEE, 2019, pp. 1777–1781.

[12] W. Huang, M. Langberg, J. Kliewer, and J. Bruck, "Communication efficient secret sharing," *IEEE Transactions on Information Theory*, vol. 62, no. 12, pp. 7195–7206, 2016.

[13] R. Bitar and S. El Rouayheb, "Staircase-PIR: Universally robust private information retrieval," in *2018 IEEE Information Theory Workshop (ITW)*. IEEE, 2018, pp. 1–5.

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

[15] 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*, vol. 56, no. 9, pp. 4539–4551, 2010.

[16] M. Blaum, J. L. Hafner, and S. Hetzler, "Partial-MDS codes and their application to raid type of architectures," *IEEE Transactions on Information Theory*, vol. 59, no. 7, pp. 4510–4519, 2013.

[17] I. Tamo and A. Barg, "A family of optimal locally recoverable codes," *IEEE Transactions on Information Theory*, vol. 60, no. 8, pp. 4661–4676, 2014.

[18] G. Calis and O. O. Koyluoglu, "A general construction for PMDS codes," *IEEE Communications Letters*, vol. 21, no. 3, pp. 452–455, 2016.

[19] M. Ye and A. Barg, “Explicit constructions of high-rate MDS array codes with optimal repair bandwidth,” *IEEE Transactions on Information Theory*, vol. 63, no. 4, pp. 2001–2014, 2017.

[20] I. Tamo, M. Ye, and A. Barg, “Optimal repair of Reed-Solomon codes: Achieving the cut-set bound,” in *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2017, pp. 216–227.

[21] H. Jafarkhani and M. Hajiaghayi, “Cost-efficient repair for storage systems using progressive engagement,” Jan. 22 2019, US Patent 10,187,088.

[22] H. Wang and D. S. Wong, “On secret reconstruction in secret sharing schemes,” *IEEE Transactions on Information Theory*, vol. 54, no. 1, pp. 473–480, 2008.

[23] Z. Zhang, Y. M. Chee, S. Ling, M. Liu, and H. Wang, “Threshold changeable secret sharing schemes revisited,” *Theoretical Computer Science*, vol. 418, pp. 106–115, 2012.

[24] A. S. Rawat, O. O. Koyluoglu, and S. Vishwanath, “Centralized repair of multiple node failures with applications to communication efficient secret sharing,” *IEEE Transactions on Information Theory*, vol. 64, no. 12, pp. 7529–7550, 2018.

[25] H. Sun and S. A. Jafar, “The capacity of robust private information retrieval with colluding databases,” *IEEE Transactions on Information Theory*, vol. 64, no. 4, pp. 2361–2370, 2017.

[26] A. Beimel and Y. Stahl, “Robust information-theoretic private information retrieval,” *Journal of Cryptology*, vol. 20, no. 3, pp. 295–321, 2007.

[27] C. Devet, I. Goldberg, and N. Heninger, “Optimally robust private information retrieval,” in *21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 269–283.

[28] R. Tajeddine and S. El Rouayheb, “Robust private information retrieval on coded data,” in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 1903–1907.

[29] 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)*. IEEE, 2018, pp. 2451–2455.

[30] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, “Speeding up distributed machine learning using codes,” *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2017.

[31] P. Peng, E. Soljanin, and P. Whiting, “Diversity vs. parallelism in distributed computing with redundancy,” in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 257–262.

[32] 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)*. IEEE, 2019, pp. 2654–2658.

[33] N. Woolsey, R.-R. Chen, and M. Ji, “Heterogeneous computation assignments in coded elastic computing,” in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 168–173.

[34] E. Ozfatura, S. Ulukus, and D. Gündüz, “Straggler-aware distributed learning: Communication–computation latency trade-off,” *Entropy*, vol. 22, no. 5, p. 544, 2020.

[35] N. Woolsey, R.-R. Chen, and M. Ji, “Coded elastic computing on machines with heterogeneous storage and computation speed,” *IEEE Transactions on Communications*, pp. 1–1, 2021.

[36] I. S. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *Journal of the society for industrial and applied mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[37] M. Blaum, J. Bruck, and A. Vardy, “MDS array codes with independent parity symbols,” *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 529–542, 1996.

[38] M. Forbes and S. Yekhanin, “On the locality of codeword symbols in non-linear codes,” *Discrete mathematics*, vol. 324, pp. 78–84, 2014.

[39] P. Gopalan, C. Huang, B. Jenkins, and S. Yekhanin, “Explicit maximally recoverable codes with locality,” *IEEE Transactions on Information Theory*, vol. 60, no. 9, pp. 5245–5256, 2014.

[40] X. Zhang, “Modified generalized integrated interleaved codes for local erasure recovery,” *IEEE Communications Letters*, vol. 21, no. 6, pp. 1241–1244, 2017.

[41] M. Blaum, “Multiple-layer integrated interleaved codes: A class of hierarchical locally recoverable codes,” *arXiv preprint arXiv:2009.12456*, 2020.

[42] 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*, vol. 57, no. 8, pp. 5227–5239, 2011.

[43] D. S. Papailiopoulos, A. G. Dimakis, and V. R. Cadambe, “Repair optimal erasure codes through hadamard designs,” *IEEE Trans. Inf. Theory*, vol. 59, no. 5, pp. 3021–3037, 2013.

[44] I. Tamo, Z. Wang, and J. Bruck, “Zigzag codes: MDS array codes with optimal rebuilding,” *IEEE Trans. Inf. Theory*, vol. 59, no. 3, pp. 1597–1616, March 2013.

[45] Z. Wang, I. Tamo, and J. Bruck, “Explicit minimum storage regenerating codes,” *IEEE Transactions on Information Theory*, vol. 62, no. 8, pp. 4466–4480, Aug 2016.

[46] 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*. IEEE, 2016, pp. 1–6.

[47] S. Goparaju, A. Fazeli, and A. Vardy, “Minimum storage regenerating codes for all parameters,” *IEEE Transactions on Information Theory*, vol. 63, no. 10, pp. 6318–6328, 2017.

[48] M. Ye and A. Barg, “Explicit constructions of optimal-access MDS codes with nearly optimal sub-packetization,” *IEEE Transactions on Information Theory*, vol. 63, no. 10, pp. 6307–6317, Oct 2017.

[49] V. Guruswami and M. Wootters, “Repairing Reed-Solomon codes,” *IEEE transactions on Information Theory*, vol. 63, no. 9, pp. 5684–5698, 2017.

[50] H. Dau, I. Duursma, H. M. Kiah, and O. Milenkovic, “Repairing Reed-Solomon codes with multiple erasures,” *IEEE Transactions on Information Theory*, 2018.

[51] I. Tamo, M. Ye, and A. Barg, “The repair problem for Reed–Solomon codes: Optimal repair of single and multiple erasures with almost optimal node size,” *IEEE Transactions on Information Theory*, vol. 65, no. 5, pp. 2673–2695, 2019.

[52] V. Guruswami, S. V. Lokam, and S. V. M. Jayaraman, “epsilon-MSR codes: Contacting fewer code blocks for exact repair,” *IEEE Transactions on Information Theory*, vol. 66, no. 11, pp. 6749–6761, 2020.

[53] R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*. Cambridge university press, 1994.

[54] C. Ruemmler and J. Wilkes, “An introduction to disk drive modeling,” *Computer*, vol. 27, no. 3, pp. 17–28, 1994.

[55] M. Jones, “Kumaraswamy’s distribution: A beta-type distribution with some tractability advantages,” *Statistical methodology*, vol. 6, no. 1, pp. 70–81, 2009.