Research Article

PB: A Product-Bitmatrix Construction to Reduce the Complexity of XOR Operations of PM-MSR and PM-MBR Codes over $GF(2^w)$

Chuqiao Xiao , Xueqing Gong , Yefeng Xia , and Qian Zhang

East China Normal University, Software Engineering Institute, Shanghai 200062, China

Correspondence should be addressed to Xueqing Gong; xqgong@sei.ecnu.edu.cn

Received 21 October 2020; Revised 18 November 2020; Accepted 18 December 2020; Published 30 January 2021

Academic Editor: Honghao Gao

Copyright © 2021 Chuqiao Xiao et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Edge computing, as an emerging computing paradigm, aims to reduce network bandwidth transmission overhead while storing and processing data on edge nodes. However, the storage strategies required for edge nodes are different from those for existing data centers. Erasure code (EC) strategies have been applied in some decentralized storage systems to ensure the privacy and security of data storage. Product-matrix (PM) regenerating codes (RGCs) as a state-of-the-art EC family are designed to minimize the repair bandwidth overhead or minimize the storage overhead. Nevertheless, the high complexity of the PM framework contains more finite-domain multiplication operations than classical ECs, which heavily consumes computational resources at the edge nodes. In this paper, a theoretical derivation of each step of the PM minimum storage regeneration (PM-MSR) and PM minimum bandwidth regeneration (PM-MBR) codes is performed and the XOR complexity over finite fields is analyzed. On this basis, a new construct called product bitmatrix (PB) is designed to reduce the complexity of XOR operations in the PM framework, and two heuristics are used to further reduce the XOR numbers of the PB-MSR and PB-MBR codes, respectively. The evaluation results show that the PB construction significantly reduces the XOR number compared to the PM-MSR, PM-MBR, Reed–Solomon (RS), and Cauchy RS codes while retaining optimal performance and reliability.

1. Introduction

Edge computing has emerged as a new paradigm for addressing local computing needs and moving data computation or storage to an edge node near the end user [1–3]. In contrast to cloud storage built on a network, edge computing-based storage is distributed among the edges of the network structure [4, 5]. Consequently, these edge nodes are particularly in need of fault-tolerant techniques to ensure system reliability and availability and even more importantly to ensure data privacy and security [6]. Nowadays, whether it is a distributed storage system [7] or a decentralized system [8, 9], it needs to guarantee the storage reliability of the source data, and these systems have used an erasure code (EC) strategy as opposed to a replication strategy which has a higher storage overhead. [10]. The well-known Reed–Solomon (RS) codes or more generally, maximum distance

separable codes (MDS), have been applied to many storage systems [11]. But in a word, an EC is essentially a particular linear combination of data symbols that involves a large number of finite field matrix product operations, and the calculation complexity over finite field is too high. There have been many studies on this issue [12–14]. In addition to the problem of high computational complexity over $GF(2^w)$, another problem of EC is that their repair bandwidth is equal to the size of the entire source data [15]. It has been reported that, based on RS coding, an average of 95,500 coded blocks per day needs to be recovered, and more than 180 TB of data per day must be transmitted through the TOR switch for data recovery [16]. Recently, a new class of ECs called regenerating codes (RGCs) has emerged that trades off both storage overhead and repair bandwidth [17]. RGC maintains the same reliability as ECs, but both of them are calculated over a finite field. There are two principal RGC classes,

namely, minimum storage regenerating (MSR) and minimum bandwidth regenerating (MBR) codes, which imply two extreme points in a trade-off known as the *storage-repair bandwidth trade-off*. There have been many frameworks designed for MSR and MBR codes, respectively [7, 18–20], but as far as we know, the product-matrix (PM) framework proposed by Rashmi et al. is the only framework that constructs two codes in a unified way [21], and the PM framework provides exact repair of PM-MSR ($2k - 2 \leq d$) and PM-MBR ($k \leq d \leq n - 1$) codes. While there has been some research on the optimization of extensions based on this framework, such as reducing the disk I/O overhead or dynamically changing the number of helper nodes according to system requests and network changes, these studies have not analyzed and optimized the framework itself [22–24]. In this paper, we focus on the PM framework, analyzing the computational complexity of the encoding, decoding, and repair processes over $\text{GF}(2^w)$. Since the PM framework has a large number of XORs at each step, especially in the decoding process of the PM-MSR code, the computational complexity of the inversion of the Vandermonde matrix is $O(n^3)$. Therefore, we propose a new construction called product bitmatrix (PB). Our contributions include the following:

- (i) The computational complexity of the PM-MSR and PM-MBR codes is strictly theoretically derived in combination with the finite field arithmetic operations.
- (ii) A new construction called product bitmatrix for MSR and MBR codes is designed, and we elaborate on the encoding, decoding, and repair processes in detail.
- (iii) Two new heuristic algorithms are presented that find a locally optimal PB which has low XORs, thereby reducing computational complexity. The experimental results show that the number of XORs dropped by 11.73% of the PB-MSR and 20.17% of the PB-MBR codes.

The remainder of this paper is organized as follows. The background of the two codes and arithmetic complexity analysis over the finite field are analyzed in Section 2. Section 3 is the analysis of the computational complexity of the encoding, decoding, and repair processes of the PM framework. In Section 4, a new PB construction is presented and implemented. In Section 5, two different heuristic algorithms are designed to find the locally optimal product bitmatrix that has the least number of XORs. Then, the results of many evaluation experiments conducted to verify the performance of optimization are presented in Section 6. Finally, Sections 7 and 8 are related works and conclusions.

2. Foundations

2.1. MSR and MBR Codes. RGCs are state-of-the-art ECs that aim to reduce the amount of data during the repair process, which means a new replacement node needs to

connect to any d helper nodes and download β symbols from each node. As shown in Figure 1, a comparison between the reconstruction and repair processes is presented. Any $k\alpha$ symbols in an α -level stripe are transferred to reconstruct origin symbols, while $d\beta$ symbols downloaded for repair are much smaller than the total size of B . It is shown in the following formula:

$$B \leq \sum_{i=0}^{k-1} \min(\alpha, (d-i)\beta). \quad (1)$$

Since both storage overhead α and bandwidth overhead β are costs, it is desirable to minimize both α as well as β . In [17], when (n, k, d, B) are fixed values, there is a trade-off between α and β ; the two extreme points in this trade-off are termed the MSR and MBR. Clearly, (α, β) of the minimum storage point is given by $((B/k), (\alpha/d - k + 1))$, and at the the minimum bandwidth point of the trade-off, (α, β) is given by $(d\beta, (2B/k(2d - k + 1)))$. RGC over $\text{GF}(2^w)$ is associated with a set of parameters $\{n, k, d, \alpha, \beta, B\}$. A comparison about metrics between Replication, EC, MSR, and MBR is shown in Table 1.

As shown in Table 1, the upper bound of storage efficient R_{MBR} code is reached (1/2) when $k = d = n - 1$, but there is no such limitation of R_{MSR} . For the sake of simplicity, if the size of a file is $B = 6$, $(n, k, d) = (6, 3, 5)$, the repair bandwidth γ of EC, MSR, and MBR is 6, 3.33, and 2.5, respectively, and the total storage overhead $n\alpha$ of EC, MSR, and MBR is 12, 12, and 15, respectively. Although the EC and MSR codes use the same storage, the repair bandwidth of EC is higher than MSR. MBR code is where storage space is allowed to expand, but repair bandwidth is at the lowest bound. The MSR codes minimize γ based on the optimal α , while the MBR codes minimize α under the optimal γ .

2.2. Finite Field Arithmetic Complexity. Since the finite field of prime size is not suitable for computers, the finite field of a size equal to a power of 2 is generally preferred. In $\text{GF}(2^w)$, each element is represented by the *formal power series*, which is a way of writing infinite sequences of bits. For example, a and b are bit sequences over the finite field $\text{GF}(2)$; $\mathbf{a} = (a_0, a_1, a_2, a_3, \dots)$; $\mathbf{b} = (b_0, b_1, b_2, b_3, \dots)$. Addition of a and b could be easily done by bitwise XOR, but multiplication is more complicated, $a \cdot b = (a_0b_0, a_0b_1 + a_1b_0, \dots, \sum_{i=0}^{l-1} a_i b_{l-1-i}, \dots)$, where l is length of sequence. A notation $a(z)$ is used as a formal power series (polynomial) to represent an infinite sequence:

$$a(z) = a_0 + a_1z + a_2z^2 + \dots + a_{l-1}z^{l-1}. \quad (2)$$

The multiplication over $\text{GF}(2^w)$ is performed by multiplying two polynomials with coefficients in binary field $\text{GF}(2)$ and then reducing the product by irreducible polynomials [25]. Hence, a polynomial represents a field element, and the set of polynomials is denoted by $\mathbb{F}_2[z]$. Due to the highest power series, *dega*(z) of $a(z)$ is equal to $w-1$, and the sum of nonzero terms of polynomials as the *weight* of $a(z)$ is denoted as $\|a(z)\|$ which equals w .

FIGURE 1: An example of parameters ($n = 10, k = 5, d = 9$): reconstruction by the data collector (DC) is shown on the left and repair by a new node when old node has failed is shown on the right.

TABLE 1: Comparison of replication, EC, MSR, and MBR.

	Replication	Erasure code	MSR	MBR
Parameters	—	(n, k)	$(n \geq 2k - 1, k, d \geq 2k - 2)$	(n, k, d)
Storage efficient $\mathbf{R}$	(k/n)	(k/n)	(k/n)	$(2d - k + 1/2d) \cdot (k/n)$
Storage overhead α	(B/k)	(B/k)	(B/k)	$(B/k) \cdot (2d/(2d - k + 1))$
Repair bandwidth γ	(B/k)	B	$(B/k) \cdot (d/(d - k + 1))$	$(B/k) \cdot (2d/(2d - k + 1))$
Disk access	1	k	d	d

The addition is a bitwise XOR operation whose complexity is w .

For multiplication, where $c(z) = a(z)b(z)$, there are two steps to compute complexity: compute the *product* and *reduce* while $a(z), b(z) \in \mathbb{F}_2[z], a(z) \neq b(z)$.

Step 1: in the *product* stage, the complexity of this stage is the *coefficients-XOR*, where

$$c(z) = a_0b_0 + (a_1b_0 \oplus a_0b_1)z + (a_2b_0 \oplus a_1b_1 \oplus a_0b_2)z^2 + \dots + (a_{w-1}b_0 \oplus a_{w-2}b_1 \oplus a_{w-3}b_2 \oplus \dots \oplus a_1b_{w-2} \oplus a_0b_{w-1})z^{w-1}. \quad (3)$$

For example, $a_0 = 1, a_1 = 1, a_2 = 1, b_0 = 0, b_1 = 1, b_2 = 1$. $c(z) = 0 + (0 \oplus 1)z + (0 \oplus 1 \oplus 1)z^2 + (1 \oplus 1)z^3 + z^4 = z^4 + z$. The sum of XORs is $S = 1 + \dots + w - 2 + w - 1 + w - 2 + \dots + 1 = (w - 1)^2$, so the *product* stage is $(w - 1)^2$.

Step 2: in the second stage, if $\deg c(z) \geq w$, $c(z)$ is reduced by an irreducible polynomial $g(z)$, until $\deg c(z) < w$. Let $l = \deg c(z)$, and convert the term $c_l z^l$ to $c_l(g(z) - z^w)z^{l-w}$. Because an

irreducible polynomial $g(z)$ degree is w , $h(z) = g(z) - z^w$, $\deg h(z) = w'$, and maximum *weight* of $h(z)$ is $w' + 1$.

Divide $c(z)$ into three polynomials: *no-reduce* part $d(z)$, *once-reduce* part $e(z)$, and *twice-reduce* part $f(z)$ as shown in Table 2. While $w - 2 + w' \leq w - 1$, there is a nonexistent second reduction.

Because $c(z) = d(z) + e(z) + f(z)$, the reduction part is as follows:

TABLE 2: Dividing $c(z)$ into three polynomials.

	$d(z)$	$e(z)$	$f(z)$
Power range	$[0, w-1]$	$[w, 2w-w'-1]$	$[2w-w', 2w-2]$
Maxweight	w	$w-w'$	$w'-1$
Product XORs	0	w'^2	$(w-2)^2 + w'^2$

$$\begin{aligned}
e(z) + f(z) &= c_w z^w + c_{w+1} z^{w+1} + c_{w+2} z^{w+2} + \dots + c_{2w-2} z^{2w-2} \\
&= z^w \left(c_w + c_{w+1} z + c_{w+2} z^2 + \dots + c_{2w-w'-1} z^{w-w'-1} \right) \\
&\quad + z^w \left(c_{2w-w'} z^{w-w'} + \dots + c_{2w-2} z^{w-2} \right) \\
&= h(z) \left(c_w + c_{w+1} z + c_{w+2} z^2 + \dots + c_{2w-w'-1} z^{w-w'-1} \right) \\
&\quad + h(z) \left(c_{2w-w'} z^{w-w'} + \dots + c_{2w-2} z^{w-2} \right).
\end{aligned} \tag{4}$$

Therefore, the *once-reduce* part $e(z)$ has w'^2 XORs. Then, we compute number of XORs of the *twice-reduce* part.

Let $h(z)(c_{2w-w'} z^{w-w'} + \dots + c_{2w-2} z^{w-2}) = h(z) * m(z)$, and the new polynomial $m(z) = \sum_{i=2}^{w'} z^{w-i}$, $i \in (2, w')$,

$i \in (2, w')$; $\deg m(z) = w-2$ and $\|m(z)\| = w'-1$. Then, $f(z) = f_0 + f_1 z + f_1 z^2 + \dots + f_{w+w'-2} z^{w+w'-2}$ which needs $(w-2)^2$ XORs. And if $w' \geq 1$, $f(z)$ also must be reduced.

$$\begin{aligned}
f(z) &= (f_0 + f_1 z + \dots + f_{w-1} z^{w-1}) + (f_w z^w + \dots + f_{w+w'-2} z^{w+w'-2}) \\
&= (f_0 + f_1 z + \dots + f_{w-1} z^{w-1}) + \underbrace{h(z) \left(f_w + \dots + f_{w+w'-2} z^{w'-2} \right)}_{\text{product XORs is } w'^2} \\
&= \underbrace{(f_0 + f_1 z + \dots + f_{w-1} z^{w-1})}_{\text{addition XORs is } 2w'-2} + (n_0 + n_1 z + \dots + n_{w-1} z^{w'-2}),
\end{aligned} \tag{5}$$

where $n(z)$ is a polynomial in which $\deg n(z) = 2w'-2$. Thus, *twice-reduce* part $f(z)$ needs $w'^2 + (w-2)^2 + 2w'-1$ XORs.

Consequently, the sum of XORs μ of the field *multiplication* over $\text{GF}(2^w)$ including two steps of the *addition* XOR among three parts is

$$\begin{aligned}
\mu &= \text{XOR}_{\text{step1}} + \text{XOR}_{\text{step2}} \\
&= (w-1)^2 + [(w')^2 + w] + [w'^2 + (w-2)^2 + 2w'-1 + w] \\
&= 2w(w-2) + 2w'(2w'+1) + 4.
\end{aligned} \tag{6}$$

Generally, a multiplication operation over $\text{GF}(2^w)$ takes $O(w^2)$ bit operations. For example, when $w=3$, multiplication results are as shown in Table 3. Because the finite field size equals 2^8 that corresponds to a byte in the computer, w is usually set to be an integral times of 8 to make encoding operations convenient.

Matrix inversion is often used in EC decoding and repair processes. Suppose A is $(n \times n)$ matrix ($3 \leq n$), and

$A^{-1} = (A^*/|A|)$; the determinant of A is solved by trigonometric matrix. Elements in the solution process include addition and multiplication, which require $\sum_{i=2}^n i(i-1)\mu + \sum_{i=2}^n i(i-1)w$ XORs. Adjoint matrix A^* is an $(n \times n)$ matrix composed of algebraic cofactors in which each element $A_{i,j}$ equals the determinant value of A excluding row i and column j . Therefore, computing A^* requires $n^2 \sum_{i=2}^{n-1} i(i-1)\mu + n^2 \sum_{i=2}^{n-1} i(i-1)w$ XORs.

Consequently, the XORs of matrix inversion are

$$\begin{aligned}
&(n^2 + 1)(\mu + w) \sum_{i=2}^{n-1} i(i-1) + n(n-1)(\mu + w) \\
&= \frac{n^3 - 3n^2 + 2n}{3} (n^2 + 1)(\mu + w) + n(n-1)(\mu + w) \\
&= \frac{n^5 - 3n^4 + 4n^3 - n}{3} (\mu + w).
\end{aligned} \tag{7}$$

TABLE 3: Multiplication over $\text{GF}(2^3)$ while $g(z) = z^3 + z + 1$ is an irreducible polynomial.

	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
	000	001	010	011	100	101	110	111
0	000	0	0	0	0	0	0	0
1	001	0	1	x	$x+1$	x^2	x^2+1	x^2+x
x	010	0	x	x^2	x^2+x	1	x^2+x+1	x^2+1
$x+1$	011	0	$x+1$	x^2+x	x^2+1	x^2+x+1	x^2	1
x^2	100	0	x^2	$x+1$	x^2+x+1	x^2+x	x	x^2+1
x^2+1	101	0	x^2+1	1	x^2	x	x^2+x+1	$x+1$
x^2+x	110	0	x^2+x	x^2+x+1	1	x^2+1	$x+1$	x^2+x
x^2+x+1	111	0	x^2+x+1	x^2+1	x	1	x^2+x	x^2

3. The Computational Complexity of PM-MSR and PM-MBR

The famous PM-RGC construction is introduced to provide MBR constructions for all feasible values of (n, k, d) and MSR constructions for all $(n, k, d \geq 2k-2)$ [21]. A detailed description of the complexity of the PM-MSR and PM-MBR codes is presented in this section, where matrices are used in the framework listed in Table 4.

For an $(n, k, d = 2k-2)$ MSR code with $\beta=1$, $\alpha = d - k + 1 = k - 1$, $d = 2\alpha$, $B = k\alpha = \alpha(\alpha+1)$ over $\text{GF}(2^w)$. The original symbols $\{u_i\}_{i=1}^B$ are arranged in the form of $M = [S_1 \ S_2]^t$ where the $\binom{\alpha+1}{2}$ entries in the

upper-triangular part of S_1 and S_2 are filled by $\binom{\alpha+1}{2}$ symbols.

Encoding is constructed using $\Psi = [\Phi \ \Lambda\Phi]$ where each element is chosen to satisfy the properties where (1) any d rows of Ψ are linearly independent; (2) any α rows of Φ are linearly independent; and (3) the n diagonal elements of Λ are distinct. Let the i -th row of Ψ be $(1 \times d)$ ψ_i^t , the i -th row of Φ be $(1 \times \alpha)$ ϕ_i^t , and the i -th row of Λ be $(1 \times n)$ λ_i . The α coded symbols stored by the i -th node are given by $\underline{c}_i^t = \psi_i^t M = \phi_i^t S_1 + \lambda_i \phi_i^t S_2$. The complexity of encoding equals the sum of addition and multiplication XORs, where the MSR encoding complexity is $n\alpha[d\mu + (d-1)w]$ and each data encoding needs $(n/k)(d\mu + (d-1)w)$ XORs.

Decoding is $\mathcal{DC}$ to reconstruct B . This process has three steps. Let $\Psi_{\text{DC}} = [\Phi_{\text{DC}} \Lambda_{\text{DC}} \Phi_{\text{DC}}]$ be the submatrix of Ψ corresponding to the k nodes. Then, get encoded data

$$\Psi_{\text{DC}} M = [\Phi_{\text{DC}} \Lambda_{\text{DC}} \Phi_{\text{DC}}] [S_1 \ S_2]^t = \Phi_{\text{DC}} S_1 + \Lambda_{\text{DC}} \Phi_{\text{DC}} S_2. \quad (8)$$

- (1) Let $\Psi_{\text{DC}} M \Phi_{\text{DC}}^t = \Phi_{\text{DC}} S_1 \Phi_{\text{DC}}^t + \Lambda_{\text{DC}} \Phi_{\text{DC}} S_1 \Phi_{\text{DC}}^t$; the result is $(k \times k)$ matrix. This step has $k^2[\alpha\mu + (\alpha-1)w]$ XORs.
- (2) Set $P = \Phi_{\text{DC}} S_1 \Phi_{\text{DC}}^t$, $Q = \Phi_{\text{DC}} S_2 \Phi_{\text{DC}}^t$, $\Phi_{\text{DC}} = [\phi_1^t \phi_2^t \dots \phi_\alpha^t]^t$. Because S_1, S_2 is symmetric, P and Q

TABLE 4: The PM framework matrix list.

Notation	Size	Name
M	MSR: $d \times \alpha$, MBR: $d \times d$	Message matrix
S_1, S_2	$\alpha \times \alpha$	Symmetric matrix
S	$k \times k$	Submatrix of M
T, T^t	$k \times (d-k), (d-k) \times k$	Submatrix of M
Ψ	$n \times d$	Encoding matrix
Φ	MSR: $n \times \alpha$, MBR: $n \times k$	Submatrix of Ψ
Λ	$n \times n$	Diagonal matrix
Δ	$n \times (d-k)$	Submatrix of Ψ
C	MSR: $n \times \alpha$, MBR: $n \times d$	Code matrix
Φ_{DC}	MSR: $k \times \alpha$, MBR: $k \times k$	Submatrix of Ψ_{DC}
Ψ_{DC}	$k \times d$	Decoding matrix
Ψ_{repair}	$d \times d$	Repair matrix

are symmetric. $\Psi_{\text{DC}} M \Phi_{\text{DC}}^t = P + \Lambda_{\text{DC}} Q$. Since $i \neq j$ and $1 \leq (i, j) \leq k$, the $(i, j)^{\text{th}}$ element is $P_{ij} + \lambda_i Q_{ij} = P_{ji} + \lambda_i Q_{ji}$. Then, $(k(k-1)/2)$ sets of equations are derived and one set of equations is as follows:

$$\begin{cases} P_{ij} + \lambda_i Q_{ij} = \psi_i^t M \phi_i \\ P_{ij} + \lambda_j Q_{ij} = \psi_j^t M \phi_j \end{cases} \longrightarrow \begin{bmatrix} 1 & \lambda_i \\ 1 & \lambda_j \end{bmatrix} \begin{bmatrix} P_{ij} \\ Q_{ij} \end{bmatrix} = \begin{bmatrix} \psi_i^t M \phi_i \\ \psi_j^t M \phi_j \end{bmatrix}. \quad (9)$$

This step involves matrix inversion; the XORs to compute a pair (P_{ij}, Q_{ij}) is $16\mu + 8w$. Consequently, the DC solves the values of P_{ij}, Q_{ij} for all $i \neq j$. The sum of XORs in this stage is $(k(k-1)/2)(16\mu + 8w)$.

- (3) After elements of P are known, the i -th row (excluding the diagonal element) is given by $\phi_i^t S_1 [\phi_1 \dots \phi_{i-1} \phi_{i+1} \dots \phi_{\alpha+1}]$ ($1 \leq i \leq \alpha+1$), and $\{\phi_i^t S_1 | 1 \leq i \leq \alpha+1\}$ is recovered through right product $[\phi_1 \dots \phi_{i-1} \phi_{i+1} \dots \phi_{\alpha+1}]^{-1}$. Then, compute $[\phi_1 \dots \phi_\alpha]^t S_1$, from which S_1 is recovered. S_2 is similarly recovered from Q . In this stage, XOR is

$$2 \left[\frac{\alpha^5 - 3\alpha^4 + 4\alpha^3 - \alpha}{3} (\mu + w)(\alpha + 1) + 2\alpha^3\mu + 2\alpha^2(\alpha - 1)w \right]. \quad (10)$$

For example, if $k = 3$, $\alpha = 2$ where

$$\begin{aligned} \Psi_{DC} M \Phi_{DC}^t &= \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}_{3 \times 3}, \\ P &= \begin{bmatrix} \backslash & \phi_1^t S_1 \phi_2 & \phi_1^t S_1 \phi_3 \\ \phi_2^t S_1 \phi_1 & \backslash & \phi_2^t S_1 \phi_3 \\ \phi_3^t S_1 \phi_1 & \phi_3^t S_1 \phi_2 & \backslash \end{bmatrix}_{3 \times 3}, \\ Q &= \begin{bmatrix} \backslash & \phi_1^t S_2 \phi_2 & \phi_1^t S_2 \phi_3 \\ \phi_2^t S_2 \phi_1 & \backslash & \phi_2^t S_2 \phi_3 \\ \phi_3^t S_2 \phi_1 & \phi_3^t S_2 \phi_2 & N_3 \end{bmatrix}_{3 \times 3}. \end{aligned} \quad (11)$$

The sum of XORs during *decoding* is $126\mu + 77w$.

Repair is to regenerate coded symbols on failed node f . Thus, the aim is to reconstruct $\underline{c}_f$ where

$$\underline{c}_f = \psi_f^t M = [\phi_f^t \lambda_f \phi_f^t] M = \phi_f^t S_1 + \lambda_f \phi_f^t S_2. \quad (12)$$

There are two roles in this stage: helper(h) and replacement(r) nodes, respectively. At first, the i -th helper node ($h_i, i \in [d]$) computes the inner product $\psi_{h_i}^t M \phi_f$ and transmits this value to the r node. Then, then r node obtains the vector $\Psi_{\text{repair}} M \phi_f = [\psi_{h_1} \psi_{h_2} \dots \psi_{h_d}]^t M \phi_f$. Because Ψ_{repair} is invertible, the vector $M \phi_f$ is recovered. From $M \phi_f$: $S_1 \phi_f, S_2 \phi_f$, since S_1 and S_2 are symmetric matrices,

$$(M \phi_f)^t = \phi_f^t M^t = [\phi_f^t S_1 \quad \phi_f^t S_2]. \quad (13)$$

The content $\underline{c}_f$ is eventually recovered. The process of repair is not complicated. The number of XORs on each h node is $\alpha\mu + (\alpha - 1)w$ and the total is $d[\alpha\mu + (\alpha - 1)w]$. The computational complexity of r includes matrix inversion, matrix product, and coefficient product and is represented as follows:

$$\frac{d^5 - 3d^4 + 4d^3 - d}{3} (\mu + w) + d^2\mu + d(d - 1)w + \alpha(\mu + w). \quad (14)$$

All feasible values of (n, k, d) of MBR codes have $\beta = 1$, $\alpha = d$, $B = \binom{k+1}{2} + k(d - k)$ over $\text{GF}(2^w)$. The original symbols $\{u_i\}_{i=1}^B$ are arranged in the form of $M = \begin{bmatrix} S & T \\ T^t & 0 \end{bmatrix}$ where the upper-triangular part of the symmetric matrix S is filled by $\binom{k+1}{2}$ symbols, and the remaining $k(d - k)$ symbols are used to construct matrix T .

Encoding is to obtain coding matrix C by matrix product with Ψ and M . Define the encoding matrix Ψ as the form $\Psi = [\Phi \ \Delta]$ where each element is chosen in such a way that (1) any d rows of Ψ are linearly independent and (2) any k rows of Φ are linearly independent. Then, C is given by $C = \Psi M$. During this process, the sum of XORs is $nd[d\mu + (d - 1)w]$.

Decoding is *DC* to recover original symbols B by connecting to any k nodes. Because the i -th node stored the vector $\psi_i^t M$, the DC gets the $(k \times d)$ matrix $\Psi_{DC} M = [\Phi_{DC} S + \Delta_{DC} T^t \ \Phi_{DC} T]$. Thus, multiply the matrix Φ_{DC}^{-1} on the left of $\Phi_{DC} T$ to recover first T , and the number of XOR in this step is $(k^5 - 3k^4 + 4k^3 - k/3)(\mu + w) + k(d - k)[k\mu + (k - 1)w]$. Subsequently, compute $\Phi_{DC} S + \Delta_{DC} T^t - \Delta_{DC} T^t$ and then multiply Φ_{DC}^{-1} on the left to recover S . The XOR of this step is $k^2 + k^2[k\mu + (k - 1)w]$.

Repair is recover $\psi_f^t M$ symbols stored in the failed node to a replacement node by connecting to an arbitrary set $\{h_j | j = 1, \dots, d\}$ of d helper nodes. First, each h_j computes the inner product $\psi_{h_j}^t M \psi_f$ and sends the result to the replacement node. The replacement node thus obtains the d symbols $\Psi_{\text{repair}} M \psi_i$ and then multiplies on the left by $\Psi_{\text{repair}}^{-1}$ to recover matrix $M \psi_f$. Since M is symmetric, $(M \psi_f)^t = \psi_f^t M$. The XOR of repair on the i -th node is $d\mu + (d - 1)w$ and on the replacement node is $(d^5 - 3d^4 + 4d^3 - d/3)(\mu + w) + d^2[d\mu + (d - 1)w]$.

The computational complexity of the PM-MSR and PM-MBR codes is shown in Table 5. As seen from Table 5, the PM framework requires multiple matrix inversions, and the number of XORs in the decoding process is very high.

4. The New Product Bitmatrix Ψ Constructed by Cauchy Matrix

In this section, we introduce the process of converting finite field elements into bitmatrices first. Then, we construct a new Ψ with the Cauchy matrix and transform this matrix into a bitmatrix called PB. The encoding, decoding, and repair processes of the MSR and MBR codes based on PB are introduced in detail.

4.1. Transforming Finite Field Elements Using Bitmatrix. Through the analysis of the finite field arithmetic in the previous section, each element e in $\text{GF}(2^w)$ is represented as a *formal power series* and the length of these polynomials is w . In [26], they have described a $1 \times w$ row vector V or a $w \times w$ matrix M which is represented as an element over $\text{GF}(2^w)$ in a new representation over $\text{GF}(2)[z]$. For any $e \in \text{GF}(2^w)$, use $M(e)$ as the matrix whose i^{th} ($0 \leq i$) column is $V(e * z^{i-1})$; $M(1)$ is the identity matrix and $M(0)$ is the all-zero matrix. For example, Figure 2 shows bitmatrices over $\text{GF}(2^3)$. The bitmatrix of $e = 3$ whose 1^{st} column is 011, 2^{nd}

TABLE 5: The number of XORs of the PM framework in encoding, decoding, and repair processes.

	PM-MSR	PM-MBR
Encoding: XORs	$n\alpha[d\mu + (d-1)w]$	$nd[d\mu + (d-1)w]$
Decoding: XORs	$\frac{(k^2[\alpha\mu + (\alpha-1)w] + / (k(k-1)/2)[16\mu + 8w] +)}{2[(\alpha^5 - 3\alpha^4 + 4\alpha^3 - \alpha/3)(\mu + w)(\alpha + 1) + 2\alpha^2\mu + 2\alpha^2(\alpha-1)w]}$	$(k^5 - 3k^4 + 4k^3 - k/3)(\mu + w) + k(d-k)[k\mu + (k-1)w] + k^2[k\mu + (k-1)w]$
Repair: XORs of helper nodes	$d[\alpha\mu + (\alpha-1)w]$	$d[d\mu + (d-1)w]$
Repair: XORs of replacement nodes	$(d^5 - 3d^4 + 4d^3 - d/3)(\mu + w) + d^2\mu + d(d-1)w + \alpha(\mu + w)$	$(d^5 - 3d^4 + 4d^3 - d/3)(\mu + w) + d^2[d\mu + (d-1)w]$

FIGURE 2: Bitmatrix of each element over $GF(2^w)$ ($w=3$); $M(0)$ is the all-zero matrix and $M(1)$ is the identity matrix. Each columns except 1st column is calculated by the left-hand column*2.

FIGURE 3: Two forms of multiplication based on bitmatrix over $GF(8)$.

column is $3 * z = 3 * 2 = 110$, and 3rd column is $3 * z^2 = 3 * 4 = 111$. There are two forms of multiplication of bitmatrix as shown in Figure 3. Using the bitmatrix representation, the encoding and decoding of PB-MSR and PB-MBR are accomplished by XOR operations, together with some copying operations.

In paper [21], they have adopted classical “Vandermonde” construct Ψ , but the inverse of an $n \times n$ Vandermonde-based matrix needs time of complexity $O(n^3)$. One well-known choice is to use the *Cauchy matrix* whose inverse has a time complexity $O(n^2)$. Let $X = \{x_1 \dots x_\alpha\}$ and $Y = \{y_1 \dots y_n\}$, where x_i and y_j are distinct elements over $GF(2^w)$ and $X \cap Y = \emptyset$. Then, the Cauchy matrix defined by X and Y has $(1/x_i + y_j)$ in element (i, j) . It is clear that any submatrix of a Cauchy matrix is still a Cauchy matrix. If using a Cauchy matrix as Ψ and converting it to bitmatrix, the number of ones in the bitmatrix means the number of XOR operations in encoding and o is the average number of ones per row in the matrix. Choosing different X and Y will get different o . Shown in Figure 4 is the instance over the finite field $GF(8)$ of the PB-MSR code with $[n=5, k=3, d=4, w=3, B=6, \alpha=2, \beta=1]$ and PB-MBR code with $[n=4, k=2, d=4, w=3, B=5, \alpha=3, \beta=1]$. When Ψ is constructed by Cauchy matrix, it uses bitmatrix transfer $\Psi \times M$.

4.2. New Product Bitmatrix Ψ of MSR Codes. For the PB-MSR code in the upper part of Figure 4, $X = \{1, 2\}$ and $Y = \{0, 3, 4, 5, 6\}$ are used to construct $(n \times \alpha)$ matrix Φ , and Λ is an $(n \times n)$ diagonal matrix. Because $\Psi = [\Phi \ \Lambda \Phi]$, any d rows of Ψ are linearly independent, any α rows of Φ are linearly independent, and the n diagonal elements of Λ are distinct. According to these three conditions, encoded matrix $C = \Psi \times M$ is generated, and each row of coded symbols in C is stored on each corresponding node, which means the i -th symbols of a row are stored on the i -th node.

$$c_1 = [u_1 + 5u_2 + u_4 + 5u_5, u_2 + 5u_3 + u_5 + 5u_6]. \quad (15)$$

In the reconstruction scene, DC links any three nodes and downloads $k\alpha = 6$ symbols.

$$\Psi_{DC} \times M = \begin{bmatrix} u_1 + 5u_2 + u_4 + 5u_5 & u_2 + 5u_3 + u_5 + 5u_6 \\ 5u_1 + u_2 + u_4 + 2u_5 & 5u_2 + u_3 + u_5 + 2u_6 \\ 2u_1 + 3u_2 + 6u_4 + 5u_5 & 2u_2 + 3u_3 + 6u_5 + 5u_6 \end{bmatrix}. \quad (16)$$

According to the decoding process introduced in Section 3, the content of the original symbols is decoded. When node 5 fails, helper nodes $d=4$. Through connecting any 4 helpers h_i ($i \in [d]$) (nodes 1, 2, 3, 4 in the example), a replacement

node r can exact regenerate symbols. Each h_i inner product bitmatrix ϕ_f as shown in Figure 5 sends results to replacement node r . Then r calculate equation (17) to obtain intermediate results:

$$\Psi_{\text{repair}} \times M = \psi_{h_i}^t \times M \times \phi_f = \begin{bmatrix} 4u_1 + 5u_2 + 6u_3 + 4u_4 + 5u_5 + 6u_6 \\ 2u_1 + 2u_2 + 7u_3 + 4u_4 + 4u_5 + 5u_6 \\ 3u_1 + 2u_2 + 2u_3 + 5u_4 + 6u_5 + 6u_6 \\ u_1 + 5u_2 + u_3 + 4u_4 + 2u_5 + 4u_6 \end{bmatrix}. \quad (17)$$

FIGURE 4: Using Bitmatrix transform $\Psi \times M$. Shown in the upper part of this figure is the transformation of PB-MSR codes for the parameters $n = 5, k = 3, d = 4$, and $B = 6$. The transformation of PB-MBR for the parameters $n = 4, k = 2, d = 3$, and $B = 5$ is shown in the bottom half of the illustration. All encoded blocks are stored on the corresponding nodes.

FIGURE 5: Product bitmatrix ϕ_f on node 1.

Then, r calculates $\Psi_{\text{repair}}^{-1}$ according to the number of helper node like this.

$$\Psi_{\text{repair}}^{-1} = \begin{bmatrix} 5 & 1 & 0 & 4 \\ 0 & 4 & 5 & 7 \\ 2 & 5 & 3 & 7 \\ 7 & 7 & 3 & 1 \end{bmatrix}. \quad (18)$$

Use $\Psi_{\text{repair}}^{-1} \times \Psi_{\text{repair}} \times M \times \phi_f$ to get vector $M\phi_f$.

$$M\phi_f = \begin{bmatrix} 4u_1 + 7u_2 \\ 4u_2 + 7u_3 \\ 4u_4 + 7u_5 \\ 4u_5 + 7u_6 \end{bmatrix}. \quad (19)$$

After the DC recalculates $[S_1\phi_f \ S_2\phi_f]$, the content of node 5 is recovered.

However, the method of random enumeration to find suitable Λ based on the third condition is not reasonable. Thus, we present a simple heuristic to create a better Λ .

Lemma 1. In the finite field, any element a (except 0) has a unique multiplicative inverse element a^{-1} , so the formula $aa^{-1} = (a^{-1})a = 1$ holds.

If $X = \{x_1, x_2\}$ and $Y = \{y_1, y_2, y_3, y_4\}$, where x_i and y_j are distinct elements over $\text{GF}(2^w)$, then a Cauchy matrix is as Φ :

$$\Phi = \begin{bmatrix} \frac{1}{x_1 + y_1} & \frac{1}{x_2 + y_1} \\ \frac{1}{x_1 + y_2} & \frac{1}{x_2 + y_2} \\ \frac{1}{x_1 + y_3} & \frac{1}{x_2 + y_3} \\ \frac{1}{x_1 + y_4} & \frac{1}{x_2 + y_4} \end{bmatrix}. \quad (20)$$

Because every $(1/(x_i + y_j))$ is distinct and their multiplicative inverse is also distinct according to Lemma 1, $(1/(x_i + y_j))^{-1} = x_i + y_j$. Then, the square of the multiplicative inverse of each element in i -th row and j -th column of Φ is used to construct diagonal matrix Λ and Ψ as follows:

$$\Psi = \begin{bmatrix} \frac{1}{x_1 + y_1} & \frac{1}{x_2 + y_1} & x_1 + y_1 & \frac{(x_1 + y_1)^2}{x_2 + y_1} \\ \frac{1}{x_1 + y_2} & \frac{1}{x_2 + y_2} & x_1 + y_2 & \frac{(x_1 + y_2)^2}{x_2 + y_2} \\ \frac{1}{x_1 + y_3} & \frac{1}{x_2 + y_3} & x_1 + y_3 & \frac{(x_1 + y_3)^2}{x_2 + y_3} \\ \frac{1}{x_1 + y_4} & \frac{1}{x_2 + y_4} & x_1 + y_4 & \frac{(x_1 + y_4)^2}{x_2 + y_4} \end{bmatrix}. \quad (21)$$

Each column of Ψ is linearly independent. Nevertheless, for every operation based on the finite fields $\text{GF}(2^w)$, when w and primitive polynomial change, the results of all operations change. Hence, matrix inversion is needed to determine whether the current structure is invertible. If the current Ψ has an irreversible submatrix, it changes each element of Λ by multiplying by the power to obtain a new completely different element. Since matrix inverses are encoded, decoded, and repaired only a few times and matrix products occur countless times, it is important to optimize the number of exclusive XORs for the matrix. The other details are covered in the next section.

4.3. New Product Bitmatrix Ψ of MBR Codes. For the lower part of Figure 4, combine $X = \{0, 1, 2\}$ and $Y = \{3, 4, 5, 6\}$ to construct $(un \times wd)$ matrix Ψ of PB-MBR. Because $\Psi = [\Phi \ \Delta]$ where Φ and Δ are $(n \times k)$ and $(n \times (d - k))$, any d

rows of Ψ are linearly independent and any k rows of Φ are linearly independent. As shown in the example, each node stores $(1 \times \alpha)$ vector, such as

$$\mathbf{c}_1 = [6u_1 + 5u_2 + u_4 \ 6u_2 + 5u_3 + u_5 \ 6u_4 + 5u_5]. \quad (22)$$

When decoding the original symbols, DC connects any two nodes and downloads $k\alpha = 6$ data, where

$$\Psi_{\text{DC}} \times M = \begin{bmatrix} 6u_1 + 5u_2 + u_4 & 6u_2 + 5u_3 + u_5 & 6u_4 + 5u_5 \\ 7u_1 + 2u_2 + 3u_4 & 7u_2 + 2u_3 + 3u_5 & 7u_4 + 2u_5 \end{bmatrix}. \quad (23)$$

When $\Psi_{\text{DC}} M = [\Phi_{\text{DC}} S + \Delta_{\text{DC}} T^t \ \Phi_{\text{DC}} T]$, T is recovered by multiplying Φ_{DC}^{-1} on the left.

$$\Phi_{\text{DC}}^{-1} \times \Phi_{\text{DC}} \times T = \begin{bmatrix} 2 & 5 \\ 7 & 6 \end{bmatrix} \times \begin{bmatrix} 6u_4 + 5u_5 \\ 7u_4 + 2u_5 \end{bmatrix} = \begin{bmatrix} u_4 \\ u_5 \end{bmatrix}. \quad (24)$$

Subsequently, S is recovered.

For example, if node 4 fails, to accurately regenerate coded symbols on the r node, it requires connecting to any three helpers $h_i(1,2,3$ in the example). Each h_i inner product bitmatrix ψ_f , respectively, sends the result to the replacement node. r node obtains d vectors $\Psi_{\text{repair}} M \psi_f$ and multiplies $\Psi_{\text{repair}}^{-1}$ on the left where

$$\Psi_{\text{repair}}^{-1} = \begin{bmatrix} 4 & 2 & 5 \\ 1 & 1 & 5 \\ 1 & 2 & 4 \end{bmatrix}. \quad (25)$$

Then, the DC recovers $(M \psi_f)^t = \psi_f^t M$ which represents the lost symbols on node 4.

5. Optimization of PB Framework Based on Minimizing the Number of Ones

An arbitrary Cauchy matrix with different X and Y brow the various number of ones, and the impact on performance is significantly different [27]. It is necessary to find a better bitmatrix Ψ which has fewer ones. Therefore, in this section, we optimize the two new Ψ by two new heuristic algorithms.

5.1. Optimizing the Number of Ones in Bitmatrix Ψ of PB-MSR. It is easy to know the number of ones of each element over $\text{GF}(2^w)$. Figure 2 shows that 1–7 elements have several ones that are represented by an array as [3, 4, 7, 5, 4, 7, 6]. A bitmatrix with fewer ones can reduce the encoding computation, and element 1 has the least ones.

For j^{th} ($j \in [1, (n/2)]$) column of bitmatrix Ψ of PB-MSR,

- (1) Count the number of ones.
- (2) For each element divide by element $\Phi[i, j]$ (i is row) and get a new column.
- (3) Repeat the first two steps n times and obtain the best j^{th} column which has the minimum number of ones.

Then, repeat the first three steps for all columns and get a new matrix which has a minimal number of ones called Φ' as well as the invertibility property.

Next, it is necessary to find $\Lambda_{\min} \times \Phi'$ which has a minimal number of ones. Let $\lambda = \{\Lambda_1, \Lambda_2, \dots, \Lambda_{d-k+1}\}$ denote the set of $(n \times n)$ diagonal matrices and let each Λ_j be n multiplicative inverses of j^{th} column of Φ' . For each Λ_j in λ , do the following:

- (1) Construct $\Psi_j = [\Phi' \ \Lambda_j \times \Phi']$.
- (2) Determine whether the new Ψ is invertible. If it is, continue to the next step; if it is irreversible, $\Lambda_j = \Lambda_j \times \Lambda_j$.
- (3) Count the number of ones of $\Lambda_j \times \Phi'$.
- (4) Repeat the first three steps and whichever $\Lambda_{\min}$ gives the minimal number of ones, form Ψ by this $\Lambda_{\min} \times \Phi'$.

Algorithm 1 outlines the procedure to generate the set of λ and the process to find the best Λ_k matrix.

For example, as shown in Figure 6, Ψ_{MSR} has $22 + 22 + 22 + 24 = 90$ ones and XORs per row $o_{\text{before}} = 90 / (4 * 3) - 1 = 6.5$, but optimized Ψ'_{MSR} has $18 + 18 + 18 + 18 = 72$ ones and XORs per row $o_{\text{after}} = 72 / (4 * 3) - 1 = 5$, which is a 23% performance improvement.

5.2. Optimizing the Number of Ones in Bitmatrix Ψ of PB-MBR. For bitmatrix Ψ of PB-MBR, let $\lambda = \{\Lambda_1, \Lambda_2, \dots, \Lambda_d\}$ (d is the number of helper nodes) denote the set of $(n \times n)$ diagonal matrices and let each Λ_j be n multiplicative inverses of j^{th} column of Ψ . For each Λ_j , do the following:

- (1) Use $\Lambda_j \times \Psi = \Psi'$.
- (2) Count the number of ones of new Ψ' .
- (3) Repeat the first two steps d times and obtain the best Ψ' .

After one column of Ψ' becomes element 1, Algorithm 2 does the same for the remaining $d-1$ columns, and each column needs to be optimized by dividing the value on this column so that the number of ones in the current column is minimized. The detailed process is written in Algorithm 2.

For example, as shown in Figure 7, original Ψ has $24 + 19 + 21 = 64$ ones and XORs per row $o_{\text{before}} = 64 / (4 * 3) - 1 = 4.33$, but optimized Ψ'' only has $12 + 18 + 18 = 48$ ones and XORs per row $o_{\text{after}} = 48 / 12 - 1 = 3$, which is a 30.7% performance improvement.

6. Experiment

In this section, we have implemented the new PB-MSR and PB-MBR codes in C/C++ and employed the Jerasure 2.0 [28] with GF-complete [29] libraries for finite field arithmetic operations. All evaluations over $\text{GF}(2^w)$ are about the reduction of XORs, encoding, decoding, and repair performances. All tests have been conducted on MacOS Catalina with 8 Intel Core i9 with 2.3 GHz clock speed and 16 GB 2667 MHz. In Section 6.1, we analyze the improved performances of the number of XORs based on new PB

construction. In Sections 6.2–6.4, we report on experiments about encoding, decoding, and repair performances. The encoding performance = total encoded file size/encoding time and the decoding performance = total decoded file size/decoding time. Finally, the influence of finite field size about w is analyzed. All experimental results were average values after maximum and minimum values have been removed.

6.1. Analyzing New Product Bitmatrix. To better understand the effects of the new PB construct in speeding up the encoding computation, some evaluation experiments were designed on reducing the number of XORs, and the results are reported in Table 6. The baseline is based on the original Cauchy matrix without any additional optimization. Original PB-MSR and original PB-MBR have two subcolumns of data, which are the sum of XORs of Cauchy matrices formed by different combinations of X and Y . Since the original PB-MBR code is constructed by original Cauchy matrix, the two columns of before PB-MBR are the actual values of original Cauchy matrices. The latter two columns are reduction of PB-MSR and reduction of PB-MBR, respectively. Improvement is measured in terms of the decrease in XORs. In the last row of the table, the average improvement over all the tested parameters is included.

Since combinations of X and Y are different and the construction process of the PB-MSR code product matrix is also more complicated, each change will bring completely different results, so this experiment only observes the overall change trend. The size of the finite field is dependent on the size of computer bytes, which are 2^8 , 2^{16} , and 2^{32} . And the choice of w needs to satisfy the range of $n + k \leq (w - 1)$, when $d = 2k - 2$. It can be seen from Table 1 that PB-MSR provides a 11.73% reduction in the number of XORs and PB-MBR provides a 20.17% reduction in the number of XORs. Because the optimization of PB-MBR is the overall optimization of matrix Ψ , the degree of optimization is greater. Figure 8 shows the XOR change curve. The two line graphs are the original PB-MSR code and PB-MBR code, which are baselines. And the number of XORs optimized for the two codes is lower than the values of the line graphs.

6.2. Encoding Performance Evaluation. The RS and Cauchy codes use the Vandermonde and Cauchy matrices, which are the most classic experiment benchmarks for all erasure codes. Figure 9 shows the comparison results of the encoding performance of PB-MSR, PM-MSR, PB-MBR, PM-MBR, RS, and Cauchy codes when the total number of coded blocks n is fixed and r equals 2, 2.5, or 3. Cauchy-good is the optimized Cauchy-matrix encoding scheme provided by the Jerasure 2.0 library. Since encoding performance is directly related to the number of XORs, when the redundancy rate r is the maximum, the coding matrix of any coding scheme is the minimum. Therefore, when $r = 3$, all encoding performances are optimal. With a decrease of r , the multiplicative operations in finite fields increase gradually, and all coding rates decrease. It can be seen from the figure that as r becomes smaller, the PB Ψ becomes larger and more

```

(1)  $O_m$  is the number of ones of a matrix;
(2)  $O_{c_j}$  is the number of ones of the  $j^{\text{th}}$  column;
(3) Use Cauchy matrix construct  $\Phi$  and count  $O_\Phi$ ;
(4) for each column  $j = 1$  to  $d - k + 1$  do
(5)   for each row  $i = 1$  to  $n$  do
(6)     for each row  $k = 1$  to  $n$  do
(7)       Calculate  $\Phi'[k, j] = \Phi[k, j]/\Phi[i, j]$ ;
(8)       Count  $O_{c_j}$  of this new column;
(9)     end for
(10)   end for
(11)   Choose the column with min  $O_{c_j}$  as new  $j^{\text{th}}$  column of  $\Phi'$ ;
(12) end for
(13) Construct  $\Phi'$  and each column has min  $O_{c_j}$ ;
(14) Count  $O_{\Phi'}$ ;
(15) for each column  $j = 1$  to  $d - k + 1$  do
(16)   for each row  $i = 1$  to  $n$  do
(17)      $\Lambda[i, j] = 1/\Phi'[i, j]$ ;
(18)     for each column  $x = 1$  to  $d - k + 1$  do
(19)        $\Lambda[i, j] \times \Phi'[i, x]$ ;
(20)     end for
(21)   end for
(22)   if  $\Psi_j = [\Phi' \ \Lambda_j \times \Phi']$  is inverse then
(23)     Count  $O_{\Lambda_j \times \Phi'}$ ;
(24)   else
(25)     repeat 16–26 steps by multiplying the power of  $\Lambda[i, j]$ ;
(26)   end if
(27) end for
(28) Choose the best  $\Psi$  which has the min  $O_{\Lambda_{\min} \times \Phi'}$ ;

```

ALGORITHM 1: Finding the optimized bitmatrix $\Psi = [\Phi' \ \Lambda\Phi']$ of PB-MSR (the algorithm first gets a new Φ' which has a minimal number of ones and then finds $\Lambda_{\min} \times \Phi'$ which has minimal number of ones).

FIGURE 6: Comparison of original Ψ_{MSR} and optimized Ψ'_{MSR} that has a minimal number of ones.

optimization time is required. Therefore, the coding rate of the PB-MSR and PB-MBR codes decreases faster than RS or Cauchy. According to the analysis in the previous section, the different Cauchy matrices will generate different optimization results, so the rise of PB-MBR code in $r = 3$ or $r = 2.5$ does not have the same proportions. However, the PB-MSR encoding rate is not as fast as the PM-MSR using the Vandermonde matrix directly when the coded object is small. This is because it requires more time to generate the optimization bitmatrix. The encoding rate of the PB-MBR code is lower than that of PB-MSR because more

redundancy is needed to be written to disk during the encoding process of the PB-MBR code. From Figure 9(b), it can be seen that with an increase of coding objects, the proportion of time to generate or optimize the matrix is relatively small, so the rate of all coding strategies increases.

6.3. Decoding Performance Evaluation. In the decoding/reconstruction experiments, the decoding performances for various values of r and fixed n for all coding schemes were tested. The decoding performance during reconstruction was measured in terms of the amount of data decoded per unit time. Figure 10 shows the comparison results. It can be observed that the decoding performances of all codes improve with increasing redundancy r . From Figure 10(a), we see that decoding performance of the PB-MSR code increased from 186.72 MB/s when $r = 2$ to 300.27 MB/s when $r = 3$. The PB-MBR code increased by 11.95% when $r = 3$ than when $r = 2$. From Figure 10(b), it can be seen that from $r = 2$ to $r = 3$, the decoding performance of the PB-MSR code increased by 48.26% and that of the PB-MBR code increased by 14.78%. Since the calculation complexity of PB-MSR and PB-MBR decoding is closely related to k , the smaller k , the lower the calculation complexity and the faster the decoding performances. Although the PB-MBR codes are similar to the PB-MSR codes, they are simpler and therefore have a higher decoding rate. As can be seen from Figure 10, PB-MSR

```

(1)  $O_m$  is the number of ones of a matrix
(2)  $O_{c_j}$  is the number of ones of the  $j^{\text{th}}$  column
(3) Use Cauchy matrix to construct  $\Psi$  and count  $O_\Psi$ 
(4)  $\lambda = \{\Lambda_1, \Lambda_2, \dots, \Lambda_d\}$  as the set of  $(n \times n)$  diagonal matrices and each matrix form from multiplicative inverses of  $\Psi$ 
(5) for each  $\Lambda_j \in \lambda$  do
(6)  $\Psi' = \Lambda_j \times \Psi$ 
(7) Count  $O_{\Psi'}$ 
(8) end for
(9) Choose the best  $\Psi'$  which has the min  $O_{\Psi'}$ 
(10) The column of the matrix with all number 1 is  $x^{\text{th}}$ 
(11) The set  $\eta$  includes all columns except  $x^{\text{th}}$  column
(12) for each column  $j \in \eta$  do
(13)   for each row  $i = 1$  to  $n$  do
(14)     for each row  $k = 1$  to  $n$  do
(15)       Calculate  $\Psi''[k, j] = \Psi'[k, j]/\Psi'[i, j]$ 
(16)     end for
(17)   Count  $O_{c_j}$  of this new column
(18)   end for
(19)   Choose the column with min  $O_{c_j}$  as new  $j^{\text{th}}$  column of  $\Psi''$ 
(20) end for
(21) Choose columns with the min  $O_{c_j}$  to construct  $\Psi''$ 
(22) Count  $O_{\Psi''}$ 

```

ALGORITHM 2: Finding the optimized bitmatrix Ψ of **PB-MBR** (the algorithm first denotes a set of λ and each $\Lambda_j \in \lambda$ and finds the best $\Psi' = \Lambda_j \Psi$ which has minimal number of ones).

FIGURE 7: Comparison of original Ψ_{MBR} and optimized Ψ''_{MBR} that has a minimal number of ones.

TABLE 6: Performance improvement by PB construction.

(n,k,d,w)	Original PB-MSR		Original PB-MBR(Cauchy)		Reduction of PB-MSR		Reduction of PB-MBR	
(5, 3, 4, 8)	597	536	643	640	16.42%	5.78%	26.28%	33.44%
(5, 3, 4, 16)	1892	1948	2123	2296	19.24%	15.09%	23.60%	35.58%
(5, 3, 4, 32)	6551	7278	8043	8807	19.94%	10.51%	7.68%	29.24%
(10, 5, 8, 8)	2559	2543	2642	2620	11.92%	11.88%	16.28%	15.31%
(10, 5, 8, 16)	8487	9383	8916	9512	5.10%	14.78%	12.07%	14.27%
(10, 5, 8, 32)	35196	35816	40672	40166	14.36%	16.07%	22.84%	18.47%
(16, 8, 14, 8)	7292	7190	7342	7402	6.62%	6.43%	10.71%	9.70%
(16, 8, 14, 16)	26599	24591	26722	27324	5.34%	4.15%	14.68%	19.34%
(16, 8, 14, 32)	111559	103443	113806	116468	13.09%	14.44%	24.56%	28.96%
AVERAGE					11.73%		20.17%

FIGURE 8: The number of XORs of PB-MSR and PB-MBR. The original PB-MSR and PB-MBR are the baselines represented by the line chart. The optimized PB-MSR and PB-MBR are represented by the bar graph.

FIGURE 9: (a) The results of encoding performance under 5 MB. (b) The results of encoding performance under 200 MB.

decoding performance is about 17.35% higher than PM-MSR, and PB-MBR is about 15.29% higher than PM-MBR.

6.4. Repair Performance Evaluation. Figure 11 shows the data transfers across the network to repair one failed node. It can be seen that both the PB-MSR and PB-MBR codes have significantly lower data transfer compared to RS codes. As can be seen from the bar chart in Figure 11(a), when the encoding object is 5 MB, the classical RS code needs 1.5 times overhead of data transfer compared to the PB-MSR code and needs 2.25 times overhead of data transfer compared to the PB-MBR code. It also can be observed from

Figure 11(a) that when the redundancy ratio r is fixed, changing the number of d will result in a smaller amount of data transferred across the network. When $d = n - 1$ in Figure 11(a), the data transferred overhead of PB-MSR code and PB-MBR code decreased by 35.74% and 16.22% compared to that of $d = 2 * k - 2$.

Additionally, the repair performances were tested as shown in Figure 12. From the overall trend, while the value of n is fixed, as the redundancy rate increases, k gradually decreases, and the unit storage capacity of each node increases; disk I/O overhead increases, and the overall repair rate decreases. In contrast to Figure 11, although PB-MSR has a larger network transmission overhead than PB-MBR, it

FIGURE 10: (a) The results of decoding performance under 5 MB. (b) The results of decoding performance under 200 MB.

FIGURE 11: An example of total amount of data transferred across the network while repairing one failed node. (a) Data transferred overhead under 5 MB. (b) Data transferred overhead under 200 MB.

is always the smallest in-unit storage capacity α . Thus, the data that need to be written to the disk is also the smallest. Besides, besides, based on the calculation and analysis results about repair complexity in Table 5, it can be seen that PB-MSR has lower calculation complexity than PB-MBR, so its repair rate is slightly higher.

6.5. The Effect of Finite Field $GF(2^w)$ on Coding Performance. Based on the analysis thus far, we know that the complexity of the encoding, decoding, or repair process is related to the finite field size w , which means every symbol $x \in [0, 2^w)$. Therefore, the experiment in this section analyzes the influence of the finite field size on the performance of ECs. The values of (n, k, d) were fixed, and the performances of $w = 8$, $w = 16$, and $w = 32$ were tested as shown in Figure 13.

Figures 13(a), 13(b), and 13(c) are the comparison diagrams of encoding, decoding, and repair performances, respectively. From these figures, we see that as the finite field size w gradually increases, the calculation complexity increases and the rate declines in varying degrees.

7. Related Works

Regenerating codes are state-of-the-art ECs with optimal repair bandwidth that have been used by some storage systems. The NCcloud storage system was one of the earliest implementable designs for 2-parity functional MSR codes (F-MSR), which maintain the same data redundancy level and same storage requirement as traditional ECs but use less repair traffic [30]. In [31], Pamies-Juarez et al. have presented the evaluation of a novel MSR code known as the

FIGURE 12: The result of repair performance.

FIGURE 13: Comparison of performance under different w .

Butterfly codes in both Ceph and HDFS. Their analysis shows that Butterfly codes are capable of reducing network traffic and reading I/O access during repairs. In [20], the Coupled-Layer (Clay) code is an MSR code that offers a simplified construction for decoding/repair by using pairwise coupling across multiple stacked layers of any single MDS code and has been evaluated over an Amazon AWS cluster. The Clay code is simultaneously optimal in terms of storage overhead, repair bandwidth, optimal access, and sub-packetization level. In [22], Rashmi et al. wanted to minimize disk I/O consumed, while simultaneously retaining optimality in terms of both storage, reliability, and network bandwidth. They presented an

algorithm to transform PM-MSR codes into I/O optimal codes (which they refer to as the PM-RBT codes). In addition to MSR codes, there was some new literature about MBR codes. In [32], Hu et al. presented NCFS which was a distributed file system under real network setting based on E-MBR codes [33]. In [34], Shah et al. presented the $(n = d + 1, k, d)$ exact-repair- (ER-) MBR codes without arithmetic operation in the repair process. In [35], the authors introduced a family of repair-by-transfer (RBT) codes which was a class of $(n, k, d = n - 1)$ ER-MBR codes that were constructed based on congruent transformations applied to a skew-symmetric matrix of message symbols. In this construction, the encoding complexity decreases from

n^4 to n^3 . Based on the new coding matrix for PM-MBR, the minimum of a finite field is reduced from $n-k+d$ to n . In [18], the authors first introduced a more natural extension to the classical PM-MBR framework, modified to provide flexibility in the choice of the number of helpers during node repair, permitting a certain number of error-prone nodes during repair. This was achieved by proving the nonsingularity of the family of matrices in large enough finite fields. To reduce the high coding and repair complexity that involves expensive multiplication operations in a finite field, another new class of RGC was proposed. In [36–38], the authors proposed a new framework of linear codes with binary parity-check code, named Binary Addition and Shift Implementable Cyclic-convolutional (BASIC) codes which enable coding and repair by XOR and bitwise cyclic shift.

Although some systems are currently using RGC, they are designed or optimized for coding structure, bandwidth overhead, storage overhead, and I/O overhead. However, there is still a lack of some classical coding framework complexity analysis from the perspective of the finite field.

ECs rely on finite field operations which can be performed using XOR operations. Many acceleration techniques have been proposed: optimizing bitmatrix design, optimizing computation schedule, common XOR operation reduction, cache management, and vectorization techniques [12]. In [13], two new heuristics have been derived called Uber-CHRS and X-Sets to schedule encoding and decoding bitmatrix by reducing the number of XORs. And several technologies were introduced in the same work by using different heuristic algorithms. In addition to smart scheduling and matching algorithms for reducing the number of XORs, other solutions were based on improving hardware performance. In [14], Plank et al. vectorized finite field operations directly based on single-instruction-multiple-data (SIMD).

8. Conclusion

In this paper, we have combined the finite field arithmetic operations to elaborate on the computational complexity of the famous PM framework in encoding, decoding, and repair processes and proposed a new construction called product bitmatrix (PB). Based on two heuristic algorithms, PB-MSR and PB-MBR codes find local minimum XORs that can improve encoding, decoding, and repair performances. Although PB has been optimized by the original framework in computation and has advantages in the decoding and repair processes, when the encoding object is small, the optimization time occupies a portion of the total encoding time. Compared with the PM adopted by Vandermonde, its advantage is insufficient.

In future research, in addition to theoretical research, we will conduct in-depth research on the fault-tolerant technology of edge computing nodes in combination with the real environment, for instance, improving network transmission performance [39], adapting the recommended data reliability storage algorithm of edge computing [40, 41], or to

design new nonlinear coding methods with deep neural networks [42, 43].

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was supported by the National Key Research and Development Project (2019YFB2102600) and the National Natural Science Foundation of China (NSFC) (61572194 and 61672233).

References

- [1] M. Satyanarayanan, “The emergence of edge computing,” *Computer*, vol. 50, no. 1, pp. 30–39, 2017.
- [2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.
- [3] S. Deng, Z. Xiang, P. Zhao et al., “Dynamical resource allocation in edge for trustable internet-of-things systems: a reinforcement learning method,” *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6103–6113, 2020.
- [4] K. Dolui and S. Datta, “Comparison of edge computing implementations: fog computing, cloudlet and mobile edge computing,” in *Proceedings of the 2017 Global Internet of Things Summit (GIoTS)*, IEEE, Geneva, Switzerland, 2017.
- [5] H. Gao, W. Huang, and Y. Duan, “The cloud-edge based dynamic reconfiguration to service workflow for mobile ecommerce environments: a qos prediction perspective,” in *Proceedings of the ACM Transactions on Internet Technology*, vol. 21, 23 pages, December 2020.
- [6] W. Yu, L. Fan, X. He et al., “A survey on the edge computing for the internet of things,” *IEEE Access*, vol. 6, pp. 6900–6919, 2017.
- [7] S. B. Balaji, M. N. Krishnan, M. Vajha, V. Ramkumar, B. Sasidharan, and P. V. Kumar, “Erasure coding for distributed storage: an overview,” *Science China Information Sciences*, vol. 61, no. 10, Article ID 100301, 2018.
- [8] D. Vorick and L. Champine, *Sia: Simple Decentralized Storage*, Nebulous Inc, Boston, MA, USA, 2014.
- [9] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin, *Storj a Peer-To-Peer Cloud Storage Network*, 2014.
- [10] L. Liang, H. He, J. Zhao, C. Liu, Q. Luo, and X. Chu, *An Erasure-Coded Storage System for Edge Computing*, IEEE Access, New York, NY, USA, 2020.
- [11] 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.
- [12] T. Zhou and C. Tian, “Fast erasure coding for data storage,” *ACM Transactions on Storage*, vol. 16, no. 1, pp. 1–24, 2020.
- [13] S. P. James, D. Catherine, and B. D. Robison, “Heuristics for optimizing matrix-based erasure codes for fault-tolerant storage systems,” in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, IEEE, Edinburgh, UK, 2012.

- [14] S. P. James, K. M. Greenan, and E. L. Miller, "Screaming fast galois field arithmetic using intel simd instructions," in *Proceedings of the 11th Conference on File and Storage Technologies*, pp. 299–306, San Jose, CA, USA, 2013.
- [15] V. A. Vins, S. Umamageswari, and P. Saranya, "A survey on regenerating codes," *International Journal of Scientific and Research Publication*, vol. 24, 2014.
- [16] K. V. Rashmi, B. Nihar, D. Gu, H. Kuang, D. Borthakur, and R. Kannan, "A solution to the network challenges of data recovery in erasure-coded distributed storage systems: a study on the facebook warehouse cluster," in *Proceedings of the 5th Workshop on Hot Topics in Storage and File Systems*, vol. 13, San Jose, CA, USA, 2013.
- [17] 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.
- [18] K. Mahdavian, A. Khisti, and S. Mohajer, "Bandwidth adaptive & error resilient mbr exact repair regenerating codes," *IEEE Transactions on Information Theory*, vol. 65, no. 5, pp. 2736–2759, 2018.
- [19] M. Ye and B. Alexander, "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] M. Vajha, V. Ramkumar, B. Puranik et al., "Clay codes: moulding $\{MDS\}$ codes to yield an $c\{MSR\}$ code," in *Proceedings of the 16th Conference on File and Storage Technologies*, pp. 139–154, Santa Clara, CA, USA, 2018.
- [21] K. V. Rashmi, B. Nihar, 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.
- [22] K. V. Rashmi, P. Nakkiran, J. Wang, B. Nihar, and R. Kannan, "Having your cake and eating it too: jointly optimal erasure codes for i/o, storage, and network-bandwidth," in *Proceedings of the 13th Conference on File and Storage Technologies*, vol. 15, pp. 81–94, Santa Clara, CA, USA, 2015.
- [23] K. Mahdavian, S. Mohajer, and A. Khisti, "Product matrix msr codes with bandwidth adaptive exact repair," *IEEE Transactions on Information Theory*, vol. 64, no. 4, pp. 3121–3135, 2018.
- [24] K. Mahdavian, S. Mohajer, and A. Khisti, "Product matrix minimum storage regenerating codes with flexible number of helpers," in *Proceedings of the 2017 IEEE Information Theory Workshop (ITW)*, IEEE, Kaohsiung, Taiwan, 2017.
- [25] H. Cohen, G. Frey, R. Avanzi et al., *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, CRC Press, Boca Raton, FL, USA, 2005.
- [26] J. Bloemer, K. Malik, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, *An Xor-based erasure-resilient coding scheme*, ICSI Technical report, 1995.
- [27] S. P. James and L. Xu, "Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications," in *Proceedings of the Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)*, pp. 173–180, IEEE, Cambridge, MA, USA, 2006.
- [28] J. S. Plank and K. Jerasure, "A library in C facilitating erasure coding for storage applications—version 2.0," Technical Report UT-EECS-14-721, University of Tennessee, Knoxville, TN, USA, 2014.
- [29] J. S. Plank, K. M. Greenan, E. L. Miller et al., "GF-Complete: A comprehensive open source library for Galois Field arithmetic," *Technical report UT-CS-13*, University of Tennessee, Knoxville, TN, USA, 2013.
- [30] Y. Hu, H. C. H. Chen, P. P. C. Lee, and Y. Tang, "Nccloud: applying network coding for the storage repair in a cloud-of-clouds," *FAST*, vol. 21, 2012.
- [31] L. Pamies-Juarez, R. Mateescu, C. Gyuot, E. Gad, and Z. Bandic, "Opening the chrysalis: on the real repair performance of codes," in *Proceedings of the 14th Conference on File and Storage Technologies*, vol. 16, pp. 81–94, Santa Clara, CA, USA, 2016.
- [32] Y. Hu, Y. Chiu-Man, K. Yan, P. P. C. Lee, and J. C. S. Lui, "Ncfs: on the practicality and extensibility of a network-coding-based distributed file system," in *Proceedings of the 2011 International Symposium on Networking Coding*, pp. 1–6, IEEE, Beijing, China, 2011.
- [33] K. V. Rashmi and B. Nihar Shah, P. V. Kumar and R. Kannan, "Explicit construction of optimal exact regenerating codes for distributed storage," in *Proceedings of the 2009 47th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1243–1249, IEEE, Monticello, IL, USA, 2009.
- [34] B. Nihar, K. V. Shah, P. V. Kumar, and R. Kannan, "Distributed storage codes with repair-by-transfer and non-achievability of interior points on the storage-bandwidth tradeoff," *IEEE Transactions on Information Theory*, vol. 58, no. 3, pp. 1837–1852, 2011.
- [35] S.-J. Lin and W.-H. Chung, "Novel repair-by-transfer codes and systematic exact-mbr codes with lower complexities and smaller field sizes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3232–3241, 2014.
- [36] H. Hou, K. W. Shum, M. Chen, and H. Li, "Basic regenerating code: binary addition and shift for exact repair," in *Proceedings of the 2013 IEEE International Symposium on Information Theory*, pp. 1621–1625, IEEE, Istanbul, Turkey, 2013.
- [37] H. Hou, K. W. Shum, and H. Li, "Construction of exact-basic codes for distributed storage systems at the msr point," in *Proceedings of the 2013 IEEE International Conference on Big Data*, pp. 33–38, IEEE, 2013.
- [38] H. Hou, K. W. Shum, M. Chen, and H. Li, "Basic codes: low-complexity regenerating codes for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 62, no. 6, pp. 3053–3069, 2016.
- [39] H. Gao, C. Liu, Y. Li, and X. Yang, "V2VR:reliable hybrid-network-oriented v2v data transmission and routing considering rsus and connectivity probability," *IEEE Transactions on Intelligent Transportation Systems*, 2020.
- [40] H. Gao, Li Kuang, Y. Yin, B. Guo, and K. Dou, "Mining consuming behaviors with temporal evolution for personalized recommendation in mobile marketing apps," in *Proceedings of the ACM/Springer Mobile Netw. Appl.(MONET)*, vol. 25, pp. 1233–1248, 2020.
- [41] X. Yang, S. Zhou, and M. Cao, "An approach to alleviate the sparsity problem of hybrid collaborative filtering based recommendations: the product-attribute perspective from user reviews," *Mobile Networks and Applications*, vol. 2, pp. 1–15, 2019.
- [42] Y. Yin, Z. Cao, Y. Xu, H. Gao, R. Li, and Z. Mai, "Qos prediction for service recommendation with features learning in mobile edge computing environment," *IEEE Transactions on Cognitive Communications and Networking*, 2020.
- [43] K. Jack, K. V. Rashmi, and S. Venkataraman, "Learning a code: machine learning for approximate non-linear coded computation," 2018, <http://arxiv.org/abs/1806.01259>.