Hindawi
Mathematical Problems in Engineering
Volume 2019, Article ID 4163780, 14 pages
https://doi.org/10.1155/2019/4163780

Research Article

A New Minimize Matrix Computation Coding Method for Distributed Storage Systems

Chao Yin,[1] **Haitao Lv**,[1] **Tongfang Li**,[1] **Xiaoping Qu**,[1] **Jianzong Wang**,[2] **and Guangyong Gao**[3]

[1]*Department of Information Science and Technology, Jiujiang University, Jiujiang, China*
[2]*Department of Federal Learning Technology, Ping An Technology (Shenzhen) Co., Ltd., Shenzhen, China*
[3]*Department of School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, China*

Correspondence should be addressed to Guangyong Gao; gaoguangyong@163.com

Received 23 April 2019; Accepted 29 August 2019; Published 15 November 2019

Academic Editor: Leonardo Acho

With the number of nodes increasing in scale, the requirements of storage space enlarge sharply in distributed storage systems. Failure-tolerance schemes such as Reed–Solomon codes (RS codes in short) and Cauchy Reed–Solomon codes (CRS codes in short) are used to save storage space. However, these failure-tolerance schemes severely degrade the system performance. In this paper, we propose optimal RS codes (OptRS codes in short) based on RS codes and CRS codes that can offer better performance for encoding and decoding as well as maximizing the utilization of storage space. OptRS codes can speed up the matrix computation which is regarded as the most important factor to impact the efficiency of coding by transferring the matrix computation from the Galois field mapping to the XOR operation. OptRS codes employ an algorithm called row elimination scheme (RE scheme in short), which can eliminate the same XOR operation to minimize the number of XOR operations. We analyze optimal matrices (OM in short) in theory, which prove the optimal performance of OptRS codes over the Galois field. Our method is implemented on the top of the distributed storage system, and code parameters were carefully chosen. The test result shows that OptRS codes can improve the performance in different data block numbers, parity block numbers, block size, normal reading, and degraded reading, compared with RS codes and CRS codes.

1. Introduction

The data become increasingly important to people with expanding in distributed storage systems as data center scale [1–3]. Whether it is for individuals or for companies, data corruption or even loss will cause unpredictable losses. In this case, the reliability is the first factor for us to consider when choosing a storage system. Replication technology and erase codes have blossomed in recent years.

The traditional distributed storage systems have already employed three-way replications to ensure failure tolerance, such as Amazon S3 [4], HDFS [5], GFS [6], which can tolerate two nodes failed at the same time. Erasure codes can tolerate more failures through different parity parameter configurations. A class of erasure codes known as maximum distance separable (MDS) codes, such as RS codes [7] and CRS codes [8], which can provide the same reliability as four-way replications, five-way replications, or even more-way replications through setting the suitable parity value as four, five, or other numbers. If erasure codes provide the same level of failure tolerance as replication technology, they can bring minimal storage overhead. For example, it is reported that RS codes have been used in Facebook [9] to reduce storage overhead of 1.4x. As a comparison, the original three-way replication costs the storage overhead of 3x. This is the reason why large-scale distributed storage systems begin to focus on erasure codes instead of replication.

It is important for these distributed storage systems to be highly efficient so that they incur low cost in different utilizations, such as mobile terminals, internet of things, and cloud computing. Compared with replication technology,

the disadvantage of the traditional MDS codes is their high writing and repair cost. Replication has much lower writing cost. When writing data, the same original data are simply replicated and written to multiple nodes. In comparison, erasure codes have a much higher writing cost due to extra complicated encoding process. The original data should be cut to many blocks and encoded in MDS codes. After encoding, the data in original blocks and parity blocks will be sent to the different nodes for load balancing. Replication has a much lower repair cost. If a node fails, an exact copy of lost data can be simply read from one of its available replicas directly. In contrast, erasure codes have much higher repair cost. If a node fails, all the user data and redundancy data within a coding group should be read from surviving nodes, and the lost data are reconstructed through the complex decoding process. It requires a large amount of read data and high computation cost during the decoding, which result in a higher repair cost compared with the replication. The high writing and repair cost induced by complex encoding and decoding severely degrade the system performance, which are the main drawbacks of conventional erasure codes.

Actually, a lot of research has been proposed on improving the performance of RS codes and CRS codes over the Galois field. Plank et al. [10] have used fast Galois field arithmetic using Intel SIMD instructions. In [11], an algorithm is proposed to generate good matrices. A new C/C++ language library is developed for the convenience of researchers [12]. We continue to study on the basis of our predecessors. This paper presents OptRS codes that extend the theoretical construction in RS codes and CRS codes. Firstly, we observe that the numbers of XOR are the most important role in studying the regular pattern of RS codes and CRS codes over the Galois field. The higher the performance of encoding and decoding in RS codes and CRS codes is, the less number of XOR operations is. Secondly, we have developed the RE scheme constructed by eliminating redundant rows from the analysis of the related rows. It can improve the system performance by decreasing the computation cost in the encoding and decoding process.

We make the following key contributions in this work:

(i) We have developed OptRS codes which eliminate the unit matrix of conventional RS/CRS codes. OptRS codes retain the characteristics of CRS codes to use fast arithmetic over the Galois field.

(ii) We have proposed a RE scheme to noticeably reduce the number of XOR operations and improve the encoding/decoding speed. The RE scheme has been used in the computation to get the coded values in OptRS codes. It can decrease the numbers of XOR and improve the encoding and decoding speed.

(iii) We have presented theoretical analysis to prove that OptRS codes can produce optimal matrices over the Galois field. The algorithms are implemented to validate their performance. The results show that the coding efficiency is higher than using CRS codes and RS codes averagely.

The remainder of this paper is organized as follows: Section 2 describes the background and motivation. We describe the details of our coding method in Section 3. Section 4 presents the matrix computation performance analysis in Galois field theoretically. The architecture of the system is introduced in Section 5. The experimental evaluation and results are shown in Section 6. Section 7 introduces the related works. The conclusions are made in Section 8.

2. Background and Motivation

Erasure codes can save storage space as an alternative to replication in storage systems. An erasure code system is usually expressed by the symbol $[n, k]$, in which n means the total block numbers and k means data block numbers. We can calculate that the parity block numbers are $n - k$, which are always denoted as m in some place. There are two situations to store the data. One is set n coded data into n blocks without distinguishing between data block or parity block; the other one is that special k blocks are set original data and $n - k$ blocks are parity data. No matter how to store data, the storage overhead of an erasure code is defined by $1 - k/n$. We calculate erasure codes over the Galois field commonly. There is a third parameter w, which means the numbers of bits in a block. For example, the size of the field is assumed size $O(2^8)$ so that each block can be represented by a byte.

2.1. RS Codes. RS codes are the encoding methods that can satisfy the number of arbitrary data disks k and the number of parity disks m, which are theoretically the most excellent fault tolerance. After long-term development, RS codes have a relatively complete theoretical foundation. The encoding methods perform standard bit arithmetic (addition is XOR and multiplication is bitwise AND) of corresponding elements over the Galois field $\mathrm{GF}(2^w)$, which belong to horizontal codes. The generator matrix used in RS codes is a Vandermonde matrix, so that RS codes are also called the Vandermonde codes.

The Vandermonde matrix is used as the generator matrix to obtain parity data. Now, let us assume that the input data are denoted D_1 to D_k and the generated parity data are denoted C_1 to C_m. The parity data of RS codes are obtained by multiplying user-defined parity matrix and the original data. The relationship between the original data and the parity data can be described as follows:

$$E = GD = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 2 & \cdots & k \\ 1 & 4 & \cdots & k^2 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 2^{m-1} & \cdots & k^{m-1} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_k \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_m \end{bmatrix}. \quad (1)$$

In order to better represent the data stored on the disk, the coding matrix equation is usually expressed as follows:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & k \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 2^{m-1} & 3^{m-1} & \cdots & k^{m-1} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_k \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}. \quad (2)$$

It can be found that this generator matrix denoted as G is a combination of a unit matrix and a Vandermonde matrix. If we want to get the encoded data directly, we can put the parity matrix and a unit matrix in a generator matrix and multiply the original data matrix denoted as D to get the matrix containing the original data and the parity data. For easy explanation, the obtained matrix is denoted as E. If we think about this in the traditional RAID idea, the encoded result is all the data that a stripe needs to store.

We have already made it clear that the encoding process is very regular. The next step is how to perform the decoding operation. When some data cannot be read in the stored data D_1 to D_n and C_1 to C_m, how can we recover the original data correctly? Assuming m data blocks are lost, the m rows corresponding to the data blocks can be deleted in the generator matrices G and the obtained matrices E. From the mathematical principle, the lost data can be recovered by multiplying the surplus valid data by the inverse matrix of the generator matrix. It means we can obtain a new generation matrix denoted as G_2 with k rows and k columns, which is the same as a result matrix denoted as E_2 with one row and n columns. Since the generator matrix has a combination of a Vandermonde matrix and a unit matrix, any n row subset of the matrix G can be guaranteed to be linearly independent. Therefore, the data can be recovered by the inverse matrix product of G_2 and E_2. The Cauchy decoding equation is described as follows:

$$\begin{aligned} \text{Decode}(D') &= (G_2')^{-1}(E_2') = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_k \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & k \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 2^{m-1} & 3^{m-1} & \cdots & k^{m-1} \end{bmatrix}^{-1} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}. \end{aligned} \quad (3)$$

From a mathematical point of view, the RAID5 and RAID6 algorithms we use today are just a subset of the Vandermonde matrix algorithm. When there is only one redundant data, it degenerate into a RAID5 algorithm. In the real field, only the input data need to be accumulated to obtain the parity codes. The coding operations are placed over the Galois field in order to simplify the computational complexity so that the addition operations become an XOR logic operation. When there are two redundant data, the Vandermonde matrix degenerates into a RAID6 algorithm, and it can also be operated over the Galois field by looking up the logarithm table and doing XOR operations.

In order to convert this multiplication into a simple XOR operation, a log/antilog operation is introduced. We know that multiplication can be converted to addition by logarithmic operations. This feature is used by RAID codes. For example, the operation of matrix multiplication $A * B$ can be converted as the following equation:

$$\begin{aligned} &A * B = C, \\ \Longrightarrow &\log_g^A + \log_g^B = \log_g^C, \\ \Longrightarrow &C = g^{\log_g^A + \log_g^B}. \end{aligned} \quad (4)$$

Therefore, the operation process of matrix multiplication $A * B$ can be decomposed into the following four steps:

Step 1. Get A_1 (the value of $\log A$) by logarithm table

Step 2. Get B_1 (the value of $\log B$) by logarithm table

Step 3. Get C_1 by XOR logical operation A_1 and B_1

Step 4. Get C from C_1 by antilog table

All coding operations of RAID can be converted into XOR operations through the aforementioned method, which can improve I/O efficiency greatly.

From this perspective, the erasure code method based on the Vandermonde matrix is an extension of the traditional RAID5/RAID6 algorithm. In the implementation process, multiplication, division, addition, and subtraction operations can also be performed over the Galois field. In order to implement a faster algorithm, it is necessary to construct two logarithmic tables and an antilog table and complete encoding and decoding operations through table lookup and XOR operation quickly.

It is relatively simple to encode and decode based on the Vandermonde matrix, which can be regarded as an extension of the RAID5/RAID6 algorithm. But the main drawback is the algorithm complexity still relatively being high. The coding complexity is $O(mk)$, where m is the number of parity data and k is the number of the original data. The decoding complexity is $O(k^3)$, which is very high.

2.2. CRS Codes. Using the Vandermonde matrix as the coding matrix, the biggest problem is that the complexity of the algorithm is too high so that it will still affect I/O performance and increase I/O latency. We should try to find another matrix for the erasure code used widely to find a more reasonable coding matrix and reduce the algorithm complexity. CRS codes are based on the Cauchy matrix developed, which have presented two important performance improvements.

The first is using the Cauchy matrix instead of the Vandermonde matrix. Since the complexity of the Vandermonde matrix inversion operation is $O(n^3)$, the complexity of the Cauchy matrix inversion operation is only $O(n^2)$. The use of the Cauchy matrix can reduce the computational complexity of decoding. The second is the use of binary matrix over the Galois field to improve the operation efficiency, which can greatly reduce the computational complexity by converting the multiplication into XOR operation directly.

The elements in the Cauchy matrix can be described as $M(1/(X_i + Y_j))$. Both X_i and Y_j are elements of the Galois field $\mathrm{GF}(2^W)$. There are two characteristics in the Cauchy matrix. First, any submatrix is a singular matrix which has an inverse matrix. Second, the inverse operation of the Cauchy matrix over the Galois field can be completed in the computational complexity $O(n^2)$.

The encoding process is the same as that of RS codes, except that the Vandermonde matrix is replaced by a Cauchy matrix. It is a series of multiplication and addition operations over the Galois field, which is described as follows:

$$\mathrm{Encode}(D) = \mathrm{GD} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ \frac{1}{x_1 + y_1} & \frac{1}{x_1 + y_2} & \frac{1}{x_1 + y_3} & \cdots & \frac{1}{x_1 + y_k} \\ \frac{1}{x_2 + y_1} & \frac{1}{x_2 + y_2} & \frac{1}{x_2 + y_3} & \cdots & \frac{1}{x_2 + y_k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{x_m + y_1} & \frac{1}{x_m + y_2} & \frac{1}{x_m + y_3} & \cdots & \frac{1}{x_m + y_k} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_k \end{bmatrix}. \tag{5}$$

When any data element $d(i)$ is damaged, data recovery is required through the decoding process. The data decoding process can be divided into the following steps:

Step 1. Select the remaining valid data blocks to form a decoded column vector. For example, if d_1 and d_3 data blocks are damaged, the remaining data d_0, d_2, c_0, and c_2 can be selected as the decoded column vector.

Step 2. Extract the row corresponding to the decoded column vector in the generator matrix G to form a square of the matrix G'. The inverse matrix of the matrix is the decoding generator matrix $\mathrm{inv}(G')$.

Step 3. The lost data d_1 and d_3 can be obtained by multiplying the matrix $\mathrm{inv}(G')$ and the decoded column vector.

The Cauchy decoding process is the same as that of the encoding so that we will not describe in detail.

From the perspective of the whole process, the matrix inversion process has the largest computational overhead. The decoding process is the same as the Vandermonde matrix coding, but the inverse of the Cauchy matrix is less complex than the Vandermonde matrix so that it has better performance. The largest computational amount of the Cauchy code is multiplication and addition in the encoding and decoding process. In Vandermonde coding, we can convert the multiplication into an addition using the log/antilog table and the addition into an XOR operation over the Galois field; however, we should think about other methods in Cauchy coding.

3. The Proposed Scheme of OptRS Codes

3.1. Encoding Process Optimization. In order to reduce the complexity of multiplication, the Cauchy codes can use the principle of binary matrix representation in elements of the finite field. It converts addition, bitwise AND, and multiplication into XOR logic operation over the Galois field. It also improves the decoding efficiency.

From a mathematical point of view, in the finite field of the Galois field, any element over $\mathrm{GF}(2^W)$ can be mapped to the $\mathrm{GF}(2)$ binary field, and a binary matrix is used to represent $\mathrm{GF}(2^W)$. For example, an element over $\mathrm{GF}(2^3)$ can be represented as a binary matrix over $\mathrm{GF}(2)$. We show $M(e)$ for each element e over $\mathrm{GF}(2^3)$ in Figure 1. From the figure, we can see that black squares indicate logic one and white squares indicate logic zero.

Through this conversion, the array in $\mathrm{GF}(2^W)$ can be converted into a binary array over $\mathrm{GF}(2)$. Firstly, we consider the situation over $\mathrm{GF}(2^3)$, and the array conversion of the generated matrix is expressed as Figure 2. The generator matrix over the Galois field $\mathrm{GF}(2^W)$ is $k * (k + m)$, which is transformed into the Galois field $\mathrm{GF}(2)$ and becomes the $(w * k) * (w * (k + m))$ binary matrix. An important property of these projections is that using standard bit arithmetic (addition is XOR, multiplication is bitwise AND), $M(e_1) * M(e_2) = M(e_1 e_2)$, which can greatly reduce the computational complexity. Compared with the log/antilog method mentioned in Vandermonde codes, this method does not need to construct a log/antilog table.

In order to make the XOR operation more efficient, the packet size should be an integer multiple of the machine word length. The size of the strip is therefore equal to w times the size

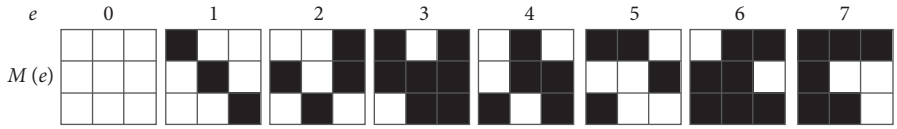


Figure 1: The transformation of the generated matrix to a binary matrix when w is 3.

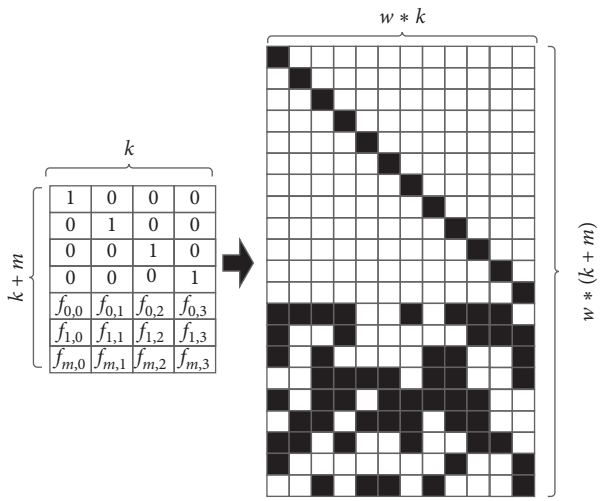


Figure 2: The transformation of the generated matrix to a binary matrix when w is 3.

of the packet. In this way, w is not related to the machine word length and may not be limited to a fixed value of 8, 16, 32, 64. It may be selected in any value, which only needs to satisfy $n \leq 2w$.

The encoding operation can be expressed after adopting this conversion method. We partition each storage block's entire data space into w packets, where each packet's size (B/w bytes) must be a multiple of the machine's word size.

3.2. Performance Influence in OptRS Codes. The Cauchy matrix used in the CRS code has better performance than the Vandermonde matrix used in the RS code. The reason is the inverse exists in the Cauchy matrix in the $O(n^2)$ time which can bring better property of encoding and decoding. We have found that the performance is related to the number of binary code *one* in the matrix. The more numbers of *one* are, the higher the performance of the code is. Furthermore, it can be further optimized by matrix transformation in the process of implementing CRS codes.

The matrix can be obtained with higher efficiency by transforming the coding operation. The basic assumption in GF(2^W) is that let w be equal to 3, while k is the number of data blocks and m is the number of the parity blocks.

Let us define the Cauchy matrix with k rows and n columns. In this matrix, data with k rows are represented as $X_1, X_2, \ldots, X_k$, while data in m columns are represented as $Y_1, Y_2, \ldots, Y_m$. The Cauchy matrix can be expressed as follows:

$$M(X,Y) = \sum_{i=1}^{k} \sum_{j=1}^{n} V\left(X_i, Y_j\right). \tag{6}$$

Let $n = 5$, $k = 2$, $X = 1, 2$, and $Y = 0, 3, 4, 5, 6$. The first n rows of the matrix constitute a unit matrix and the last k rows are a Cauchy matrix. We have made a computation which maps matrix the XOR operation instead of being conducted over the Galois field to improve the speed of the matrix calculation as the last section introduces. To complete this computation, it needs the following operations. First, each data element in GF(2^W) should be expressed as the form of w rows and w columns. After this transformation, the original CRS matrix is changed into a binary matrix with $(k+n)*w$ rows and $n*w$ columns. Accordingly, the matrix encoding and decoding operations in GF(2) have the lower computational complexity than in GF(2^W). The whole matrix transformation is shown in Figure 3.

Moreover, instead of partitioning the data into words of size $2w$, we partition each storage device's entire data space into w packets, where each packet's size (B/w bytes) must be a multiple of the machine's word size. Continuing our example from Figure 3, suppose each device holds 3 GB, then each device D_i and C_i will be partitioned into three 1 GB packets, $D_{i,1}$, $D_{i,2}$, and $D_{i,3}$ or $C_{i,1}$, $C_{i,2}$, and $C_{i,3}$. Then, the encoding and decoding processes may now be implemented over GF(2) (bit arithmetic) rather than GF(2^3). Figure 4 shows the transformation of the matrix. The unit matrix is omitted to make the graph concise, retaining only the Cauchy matrix with k rows and n columns.

The shaded grid is *one* and the blank one is *zero*. Because the transformed matrix is in GF(2), we can use the XOR operation to compute the encoded values. The value of $C_{i,j}$ can be computed by $D_{i,j}$ and all XOR of its related columns. For example,

$$C_{1,1} = D_{1,1} \oplus D_{2,1} \oplus D_{2,2} \oplus D_{3,3} \oplus D_{4,1} \oplus D_{4,2} \oplus D_{4,3} \oplus D_{5,2}. \tag{7}$$

The number of *one* is 8 in this line, which means there are eight data packets to do the XOR operations. The number of XOR operations is the subtracted one from the corresponding number of *one*, which is 7.

According to the computer method, we can get the following equation:

$$\begin{cases} C_{1,1} = D_{1,1} \oplus D_{2,1} \oplus D_{2,2} \oplus D_{3,3} \oplus D_{4,1} \oplus D_{4,2} \oplus D_{4,3} \oplus D_{5,2}, \\ C_{1,2} = D_{1,2} \oplus D_{2,3} \oplus D_{3,1} \oplus D_{3,3} \oplus D_{4,1} \oplus D_{5,2} \oplus D_{5,3}, \\ C_{1,3} = D_{1,3} \oplus D_{2,1} \oplus D_{3,2} \oplus D_{4,1} \oplus D_{4,2} \oplus D_{5,1} \oplus D_{5,3}, \\ C_{2,1} = D_{1,1} \oplus D_{1,2} \oplus D_{2,1} \oplus D_{3,1} \oplus D_{3,3} \oplus D_{4,2} \oplus D_{5,1} \oplus D_{5,2} \oplus D_{5,3}, \\ C_{2,2} = D_{1,3} \oplus D_{2,2} \oplus D_{3,1} \oplus D_{3,2} \oplus D_{3,3} \oplus D_{4,2} \oplus D_{4,3} \oplus D_{5,1}, \\ C_{2,3} = D_{1,1} \oplus D_{2,3} \oplus D_{3,2} \oplus D_{3,3} \oplus D_{4,1} \oplus D_{4,3} \oplus D_{5,1} \oplus D_{5,2}. \end{cases} \tag{8}$$

The number of *one* is, respectively, 8, 7, 7, 9, 8, 8 in each line, and the number of XOR operations corresponds to 7, 6, 6, 8, 7, 7. We can conclude that the total numbers of the XOR matrix are 41, which is the number of calculating XOR

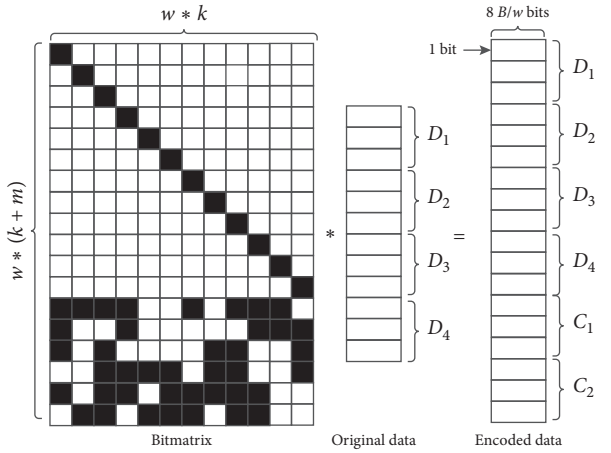


Figure 3: The whole matrix transformation of the CRS encoding process.

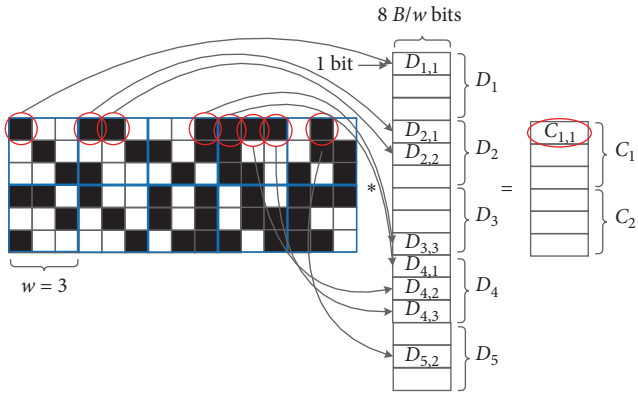


Figure 4: Matrix encoding operation in CRS codes when n is 5, k is 2, w is 3, X is 1, 2, and Y is 0, 3, 4, 5, 6. The number *one* is 8 in red circle, so that the number of XOR operations is 7.

matrix when n is 5, k is 2, and w is 3. In order to improve the encoding performance, we consider how to minimize the numbers of XOR.

3.3. Two-Row Elimination Scheme. The overhead of XOR operations is a big partial of the system computing, so that the numbers of calculating the XOR matrix can determine the computing speed, as well as the system performance. We develop an elimination scheme to solve this question.

At first, we set every two adjacent rows into one pair. There is the same XOR subset $D_{3,3} \oplus D_{4,1} \oplus D_{5,2}$ in the first pair $C_{1,1}$ and $C_{1,2}$, as we can see the pair of red circle from Figure 5. We can get

$$C_{1,1} = D_{1,1} \oplus D_{2,1} \oplus D_{2,2} \oplus D_{3,3} \oplus D_{4,1} \oplus D_{4,2} \oplus D_{4,3} \oplus D_{5,2},$$
$$C_{1,2} = D_{1,2} \oplus D_{2,3} \oplus D_{3,1} \oplus D_{3,3} \oplus D_{4,1} \oplus D_{5,2} \oplus D_{5,3}. \quad (9)$$

For convenience of description, we have summarized the relative parameters in Table 1.

When calculating the first pair $C_{1,1}$ and $C_{1,2}$, we can reduce two XOR operations. Similarly, the second pair $C_{1,3}$ and $C_{2,1}$ has the same XOR subset $D_{4,2} \oplus D_{5,1} \oplus D_{5,3}$, and we can reduce two XOR operations when calculating them. There are three XOR operations that can be reduced in calculating the third pair $C_{2,2}$ and $C_{2,3}$. We can see the pair of purple circle and green circle from Figure 6(a).

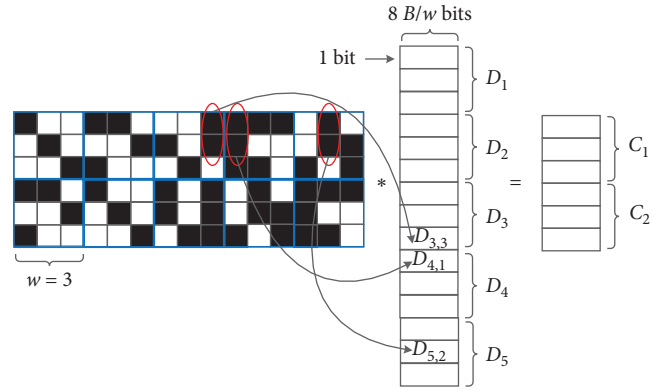


Figure 5: When carrying out two adjacent rows elimination scheme, the two relative rows can eliminate the same XOR operations in the pair of red circle in the matrix encoding operation in the first step.

Table 1: Relative parameters.

Parameters	Description
N_{one}	The number of *one*
N_{xor}	The number of XOR operations
Pair	The relative rows with *one* in the same column
Group	The original data divided by GF(2^w)

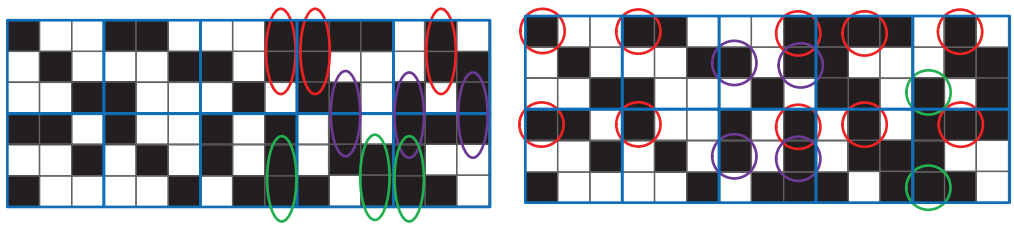
Through the above operation, XOR total numbers have been decreased to $N_{xor} = 41 - 2 - 2 - 3 = 34$. Since all the schemes are eliminated in two rows, we called it two rows elimination scheme.

From the scheme mentioned above, we use the rows in the adjacent rows to eliminate the same operation. Actually, there is a lot of situation between different rows in the two rows elimination scheme. We select the two rows in the adjacent groups at this time. A detailed example is shown in the same matrix. We can see this in Figure 6(b).

There is the same XOR subset $D_{1,1} \oplus D_{2,1} \oplus D_{3,3} \oplus D_{4,2} \oplus D_{5,2}$ in $C_{1,1}$ and $C_{2,1}$. We can reduce four XOR operations when calculating $C_{1,1}$ and $C_{2,1}$. Similarly, there is the same XOR subset $D_{3,1} \oplus D_{3,3}$ and $D_{3,2} \oplus D_{4,1} \oplus D_{5,1}$ in $C_{1,2}$, $C_{2,2}$ and $C_{1,3}$, $C_{2,3}$, which reduces the XOR operation one and two times, respectively. The total reduced numbers are $N_{xor} = 41 - 4 - 1 - 3 = 34$ times, which is equal to the result used by the two rows elimination scheme mentioned above.

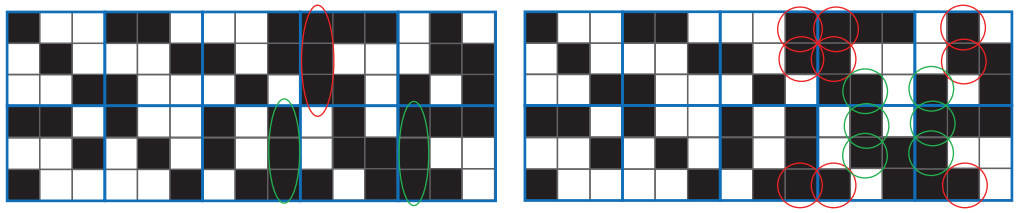
3.4. More than Two Rows Elimination Scheme. Three rows elimination scheme means that we can eliminate the same values from three rows. We can see that the group number is 3 from Figure 7.

We can see that there is not the same XOR subset among $C_{1,1}$, $C_{1,2}$, and $C_{1,3}$, while there is the same XOR subset $D_{3,3} \oplus D_{5,1}$ in $C_{2,1}$, $C_{2,2}$, and $C_{2,3}$. When calculating $C_{2,1}$, $C_{2,2}$, and $C_{2,3}$, the XOR operation can be reduced by $1 * 2 = 2$ times which is also the total numbers of reduced XOR.



(a) (b)

Figure 6: (a) Two adjacent rows elimination scheme. The two adjacent rows can eliminate the same XOR operations in the pair of different color circles. (b) Two rows elimination scheme in the adjacent groups. The two rows can eliminate the same XOR operations in the pair of different color circles.



(a) (b)

Figure 7: (a) Three adjacent rows elimination scheme. The three adjacent rows can eliminate the same XOR operations in the pair of different color circles. (b) Three rows elimination scheme in the adjacent groups. The three rows can eliminate the same XOR operations in the pair of different color circles.

From the previous discussion, we can conclude that the total number of the XOR subset will be more between the group of $C_{1,1}$, $C_{1,2}$, and $C_{2,3}$ that of $C_{2,1}$, $C_{2,2}$, and $C_{1,3}$. According to the elimination scheme, there is the same XOR subset $D_{3,3} \oplus D_{4,1} \oplus D_{5,2}$ in group $C_{1,1}$, $C_{1,2}$ and $D_{4,2} \oplus D_{5,1}$ in group $C_{2,1}$, $C_{2,2}$, and $C_{1,3}$. The total XOR numbers have been reduced by $2 * 2 + 1 * 2 = 6$.

We expand the value of w from 3 to i so that we can get the expanded matrix code conversion schematic diagram in Figure 8.

Two rows elimination scheme will be shown as follows. Comparing every two rows to draw the same XOR operations, we set the same XOR term for P_i, in which i means the serial number of rows. These steps can reduce the numbers of XOR P_{i-1} times, which is shown in the following equation:

$$\text{Num} = \begin{cases} \sum_{i=1}^{mw/2} p_{2i-1} - 1\,(mw\%2 = 1), \\ \sum_{i=1}^{(mw-1)/2} p_{2i-1} - 1\,(mw\%2 = 0). \end{cases} \tag{10}$$

Synthesizing all of the eliminated situations, we will get the conclusion.

Inference 1. In the case of $\text{GF}(2^W)$, let us suppose data block number as k and parity block number as m. If there is the same number of XOR operation in each t line, we define the number as P_i, in which i means the serial number of rows. The total numbers of XOR can be reduced as

$$\text{Num} = \begin{cases} (t-1) * \sum_{i=1}^{mw/t} p_{ti-1} - 1\,(mw\%t = 0), \\ (t-1) * \sum_{i=1}^{(mw-1)/t} p_{2i-1} - 1\,(mw\%t = 1), \\ \cdots\cdots \\ (t-1) * \sum_{i=1}^{(mw-1)/t} p_{2i-1} - 1\,(mw\%t = t-1). \end{cases} \tag{11}$$

All of the elimination schemes are constituted by two lines elimination scheme, three lines elimination scheme to mw lines elimination scheme. At the same time, it includes hybrid line elimination scheme which means the uncertain numbers of rows. There are $\text{GF}(2^W)$ elimination methods. Therefore, the elimination problem is an NP-hard problem, and there is not a best solution. The method proposed in this paper is the realization of the approximate optimal solution. We can also eliminate three or more lines at the same time, but it will cause a reduction of XOR operation numbers. Although it can be multiplied by the coefficient, which is the elimination line number $k - 1$, the achieved optimization results will be similar with two rows. In summary, we will use two elimination schemes in the following tests.

3.5. Row Elimination Scheme Algorithm Flow. Algorithm 1 shows the flow of row elimination scheme algorithm.

Since the row elimination scheme has no relationship with the encoded data, algorithm 1 can work after the encoding matrix comes into being. We will show how the row

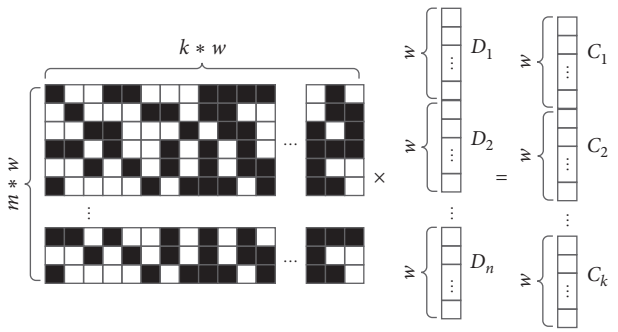


Figure 8: The matrix code conversion schematic diagram of the expanded w.

Require:
The row number in matrix from 1 to kw, i;
The column number in matrix from 1 to mw, i;
The middle result of XOR operations, Multi;
Ensure:
Null;
Require: $kw \geq 0 \vee mw \geq 0$
(1) $i \Longleftarrow 1$
(2) $j \Longleftarrow 1$
(3) **while** $j \leq km$ **do**
(4) **while** $j \leq kw$ **do**
(5) **if** $M_{i,j} == M_{i+1,j} == 1$ **then**
(6) Multi $\Longleftarrow M_{i,j} \oplus M_{i+1,j}$
(7) $N \Longleftarrow N/2$
(8) **else**
(9) $j \Longleftarrow j + 1$
(10) **end if**
(11) **end while**
(12) $i \Longleftarrow i + 2$
(13) **end while**

Algorithm 1: Row elimination scheme algorithm flow.

elimination scheme algorithm flow works according to the adjacent two rows elimination scheme. The matrix is divided into different groups. Firstly, every adjacent two rows are in the same group, which will be scanned through parallel processing. If there is *one* in adjacent two rows, it will be recorded into a pair. After scanning the whole group, all the pairs are recorded and the number of pairs has been calculated. The number of XOR operations is one less than the number of pairs. Secondly, we will perform XOR arithmetic operations in the pairs. After performing XOR arithmetic operations in the first row of the pair, the system will record the value of XOR arithmetic operations, which can be reused in calculating the second row of the pair directly. Lastly, we will perform XOR arithmetic operations which are not in the pairs.

4. The Performance Comparison between CRS Codes and OptRS Codes

4.1. Enumerating the Matrix When $w \leq 4$. In this section, we will compare the performance between CRS codes and OptRS codes. At first, let us define some variable. Let o be the average number of *one* per row in the distribution matrix. Then, the number of XORs to produce a word in each coding packet is equal to $o - 1$. We define optimal coding as $k - 1$ XORs per coding word [13, 14].

When $w \leq 4$, we can enumerate all the matrices of CRS codes and OptRS codes as the arbitrary parameter data block number k and the parity block number m, which satisfied $m + k \leq 2^W$. For example, the Cauchy matrix of Figure 4 indeed has the number of *one* 47. The average number of *one* per row o is $47/6 = 7.83$. Since that matrix requires $7.83 - 1 = 6.83$ XORs per coding word and optimal coding would require $5 - 1 = 4$, its factor is $6.83/4 = 1.71$. At the same time, the OptRS matrix of equation (8) indeed has the number of *one* 41. The average number of *one* per row o is $41/6 = 6.83$. Since that matrix requires $6.83 - 1 = 5.83$ XORs per coding word, and optimal coding would require $5 - 1 = 4$, its factor is $5.83/4 = 1.46$.

From Figure 9, we can see that there are three features worth mentioning. First, there is a significant difference in the performance of the minimum and maximum matrices both for CRS codes and OptRS codes. When k is smaller, there is a greater variety of possible values in the Cauchy matrix. But, all the values of CRS codes are bigger than those of OptRS codes among the minimum and the maximum values. Second, both the performance of CRS codes and OptRS codes get worse as k grows because the matrix of CRS codes and OptRS codes must contain more values from $\mathrm{GF}(2^W)$ when it grows.

According to calculation, we can conclude that the last feature of the matrix is better when w is bigger and when k is equal to m. For example, we suppose $k = 2$, $m = 2$, and when $w = 2$, the minimum matrix of CRS has 10 *one*, while the minimum matrix of OptRS codes has 9 *one*. This means that CRS codes have $10/4 - 1 = 1.5$ XORs per coding word and OptRS codes have $9/4 - 1 = 1.25$ XORs per coding word. When $w = 3$, the minimum matrix of CRS codes has 14 *one*, while the minimum matrix of OptRS codes has 13 *one*. This means that CRS codes have $14/6 - 1 = 1.33$ XORs per coding word and OptRS codes have $13/6 - 1 = 1.17$ XORs per coding word. When $w = 4$, the minimum matrix of CRS codes has 18 *one*, while the minimum matrix of OptRS codes has 16 *one*, which means 1.25 and 1 XORs per coding word, respectively.

4.2. Analyze OptRS Codes When $w > 4$. When $w > 4$, it is impractical to enumerate all the matrices of CRS codes and OptRS codes to find optimal matrices. We expand the value of w from 4 to i so that we can get the regular pattern of the relative matrices. Since the less the number of *one* the matrix has, the best the performance is. Our purpose is how to find the least number of *one* in the matrix when in different w.

We call the matrices that it produces optimal matrices (OM in short) and parameterize OM with k, m, and w. According to the results when $w \leq 4$, we can conclude that the OM (k, m, w) matrices where $k = m$ are optimal. We are going to actually prove this below for the given parameter k, m, and w by mathematical induction.

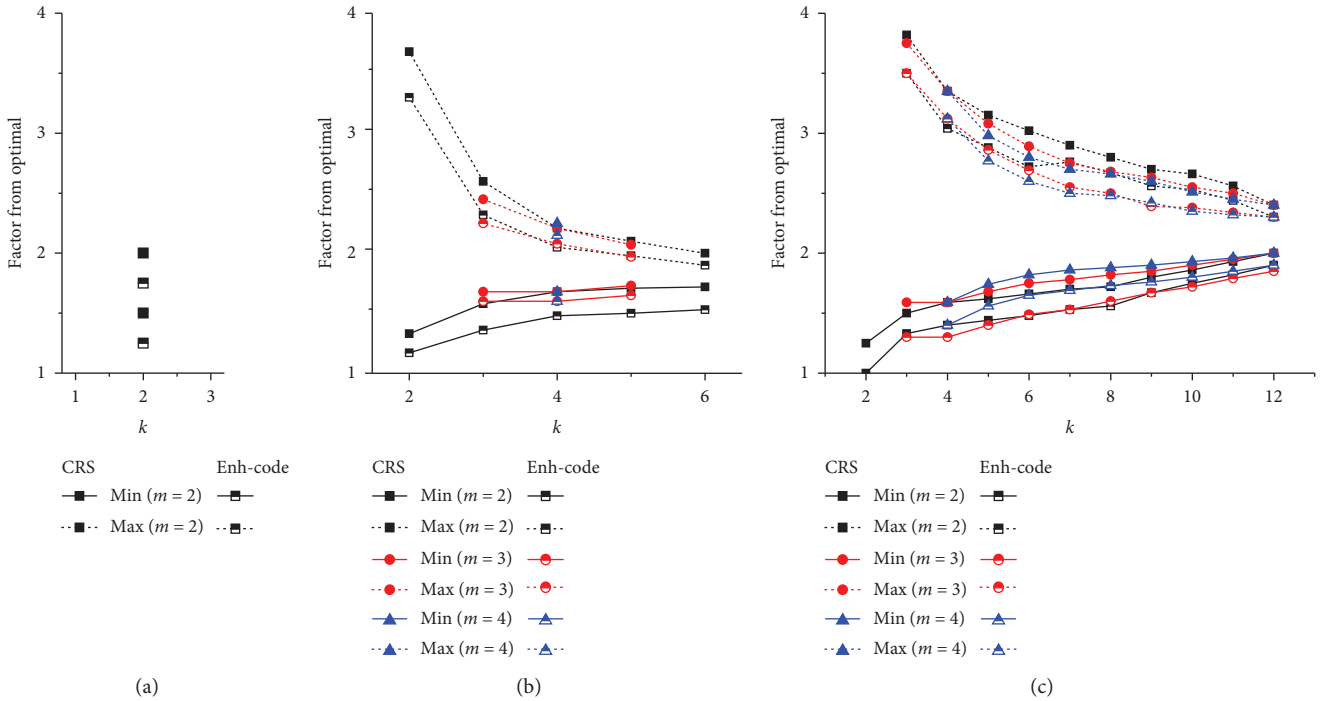


Figure 9: Minimum and maximum Cauchy matrices for $w \leq 4$ with different k and m.

We divide all the OM matrices when $w > 4$ into three situations: first, $\mathrm{OM}(k, m, w)$ when $k = m$ and k is an even number; second, $\mathrm{OM}(k, m, w)$ when $k = m$ and k is an odd number; third, $\mathrm{OM}(k, m, w)$ when $m \geq k$.

Theorem 1. $\mathrm{OM}(k, m, w)$ *when* $k = m$ *and* k *is an even number. We can translate* $\mathrm{OM}(k, m, w)$ *into* $\mathrm{OM}(k, k, w)$.

Proof. From Figure 9, we can see that $\mathrm{OM}(2, 2, w)$ is the best performance matrix when $w \leq 4$. When $k > 2$ and k is an even number, we construct $\mathrm{OM}(k, k, w)$ from $\mathrm{OM}(k/2, k/2, w)$. For example, we construct $\mathrm{OM}(6, 6, w)$ from $\mathrm{OM}(6/2, 6/2, w)$, which is described above.

We can prove $\mathrm{OM}(6/2, 6/2, w)$. which equals to $\mathrm{OM}(3, 3, w)$ is the optimal matrix. □

Theorem 2. $\mathrm{OM}(k, m, w)$ *when* $k = m$ *and* k *is an odd number. We can also translate* $\mathrm{OM}(k, m, w)$ *into* $\mathrm{OM}(k, k, w)$.

Proof. hen $k > 2$ and k is an odd number, we construct $\mathrm{OM}(k, k, w)$ from $\mathrm{OM}(k - 1, k - 1, w)$, for example, we construct $\mathrm{OM}(3, 3, w)$ from $\mathrm{OM}(3 + 1, 3 + 1, w)$, and so on. Thus, we construct $\mathrm{OM}(k, k, w)$ by constructing $\mathrm{OM}(k + 1, k + 1, w)$ and deleting the row and column that result in a minimal weight matrix. □

Theorem 3. *We define* $\mathrm{OM}(k, m, w)$, *where* $k! = m$ *to be the minimum weight super matrix of* $\mathrm{OM}(\min(k, m), \min(k, m), w)$.

Proof. Suppose $k > m$. One constructs $\mathrm{OM}(k, m, w)$ by first constructing $\mathrm{OM}(m, m, w)$, then sorting the weights of the rows that can potentially be added to $\mathrm{OM}(m, m, w)$ to create $\mathrm{OM}(k, m, w)$, and then adding the $k - m$ smallest of these rows. The construction when $m \geq k$ is analogous, except columns are added rather than rows. For example, $\mathrm{OM}(3, 4, 3)$ is constructed by adding a column to $\mathrm{OM}(3, 3, 3)$. If $k < m$, the situation is similar, which will not be proved in this paper again. □

5. The Architecture of the Storage System

5.1. The Architecture of the Distributed Storage System. In this paper, we add erasure codes to the distributed storage system. The specific system framework is shown in Figure 10.

The core module in the system is the cluster management module, which manages all the nodes of the system. It can read and write data from locating the object in erasure codes or replicas. If the system nodes fail or there are new nodes added to the system, it will inform upper level and manage operations to maintain consistency information among nodes.

The node management module receives the request from the gateway and performs the read/write operation according to the request type. If the requested data is in the form of erasure codes, they will be encoded and decoded in the EC module. When it is in writing operation, the EC module will cut the data and encode them into strips. Part of the data will be stored in local disks, while other data fragments and encoded parity values are transmitted to other nodes through the gateway. If the operation is read, the situation will be more complicated. The gateway can only

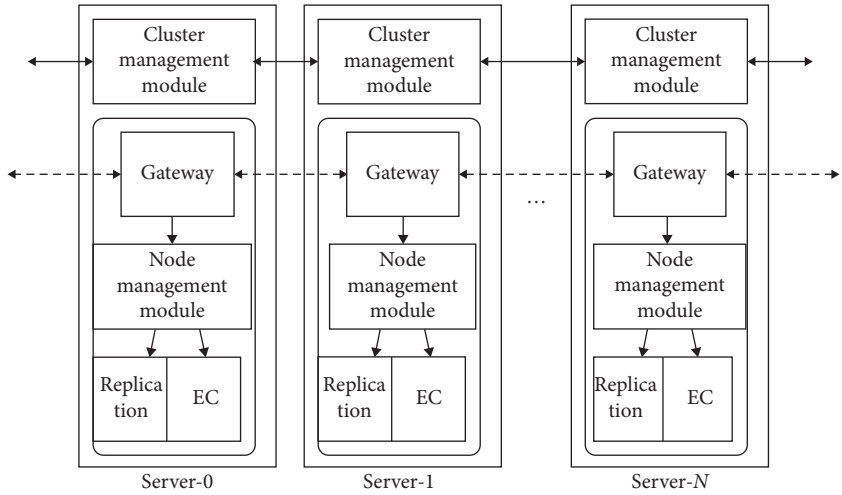


Figure 10: Architecture of the distributed storage system.

know the location where the data are stored, but the gateway cannot know whether the data have been damaged or not. When the node management module reads the data corrupted, it will return a read failure message to the gateway. The gateway will extract data from the other nodes and recover the data on the node by encoding rules. After the data have been recovered, they are passed to the application.

5.2. The Design of the Coding System. The overall architecture of the storage system is shown in Figure 11. The system has two storage methods: replication and erasure code. After the application initiates a read/write request, the local gateway receives the request and locates the server node where the data block is located through a consistent hash algorithm. If it is on another server node, request is forwarded to another node. If it is on the local node, it will enter the encoding process.

The encoding process is mainly divided into two processes: striping processing of data and encoding/decoding of data.

Striping of data refers to cutting user data into equal-sized blocks of data. Each block of data is called a strip, and multiple consecutive strips form a stripe. Assuming that the data block is $(D_1, D_2, D_3, \ldots, D_k)$, the data block is encoded with an encoding matrix (erasure matrix), and the obtained parity data block is $(P_1, P_2, \ldots, P_m)$. Both D_i and P_i correspond to a strip, where P_i is represented as a parity block. The $k+m$ strips obtained after the verification are, respectively, stored on $k+m$ nodes in the cluster.

When the write operation is performed, after the user data is stripped and after processing by the encoding matrix, $k+m$ strips are, respectively, stored to the corresponding nodes. As shown in step (3) in the figure, only one strip data block is stored to the local disk, and the other $k+m-1$ strips are stored to other nodes of the system.

The process of reading operations is relatively complicated because it involves data corruption. When the data are intact, as shown in steps (5) and (6), the corresponding data block is read from the local disk and other nodes, and the data are directly stripped and processed to obtain the original data without decoding operation. However, when the local data fail, the data blocks and the parity blocks are read from other nodes at first, and then the data blocks and the parity blocks are decoded by the coding matrix to obtain the data on the local disk. Then, the data are back-striped with the data blocks of other nodes to obtain the original data. Finally, the local data obtained by the decoding operation will be written back to the local disk.

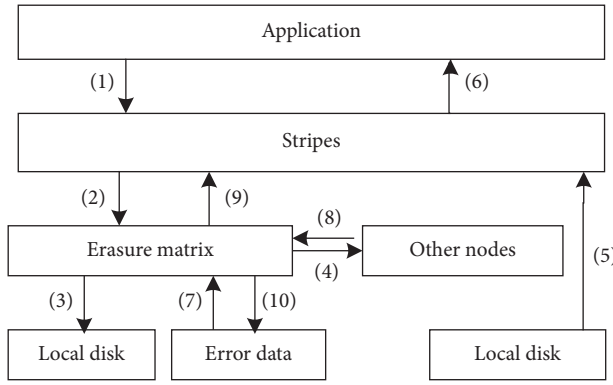


Figure 11: Architecture of the storage system: (1) divide data into trips; (2) encode operations; (3) write data to local disks; (4) write data to other nodes; (5) read data from the local disk and succeed; (6) splicing data; (7) read data from the local disk and fail because there are error data; (8) read data from other nodes; (9) decode operations; (10) encode operations and rewrite data to local disks.

6. Evaluation

We evaluate the erasure coding system along two dimensions, namely, encoding/decoding performance and read performance. Data are encoded in an erasure-coded storage system with three schemes, RS codes, CRS codes, and OptRS codes. The experiments are implemented on a four-node cluster of machines. Each machine is equipped with Xeon $E5606$ CPU with 16 GB $DDR3$ memory and 500 GB disks, and an Intel Xeon $E5-2660$ CPU with 32 GB memory and $1T$ disk is equipped in the server node.

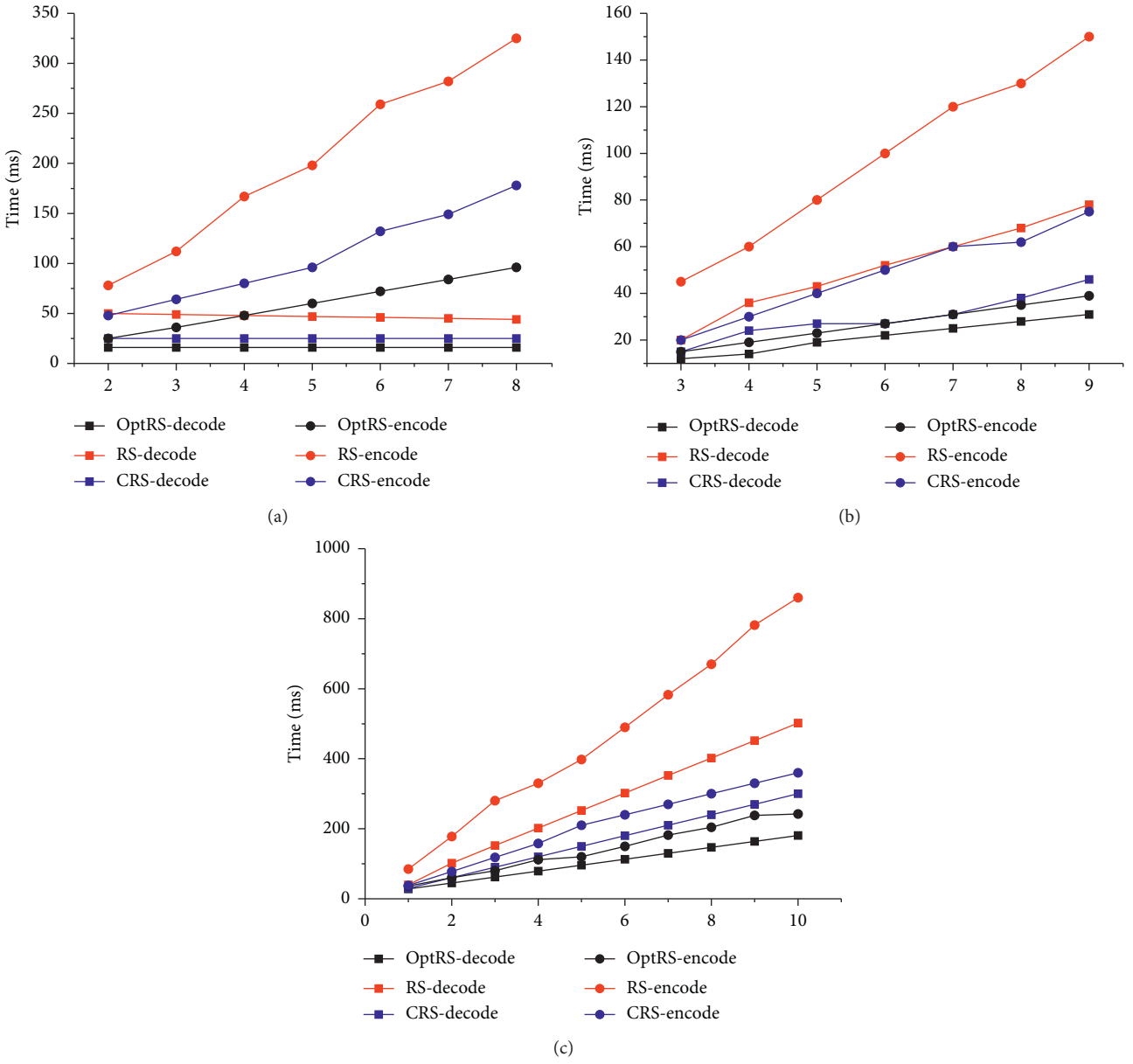


Figure 12: Encoding and decoding time with different parameters. (a) Encoding and decoding time with different parity block numbers. (b) Encoding and decoding time with different data block numbers. (c) Encoding and decoding time with different block size.

6.1. Encoding and Decoding Performance. The encoding and decoding time will be determined by different numbers of data blocks k, parity blocks m, as well as the size of blocks s. We change one of the three parameters at one time and compare the performance with the baseline.

Firstly, let $k = 5$, $s = 1$ MB, and adjust the value of m from 2 to 8. Secondly, k is changed from 3 to 9, while $s = 4$ MB and $m = 2$. The encoding and decoding time is shown in Figures 12(a) and 12(b).

We can see that the encoding time is in direct proportion to the block numbers. As we all know, the encoding time is relative with the computational cost. It takes more time to calculate with the total number of blocks increasing, which leads to more encoding time.

When the system is decoding, it will pick up the blocks which they need and will not distinguish whether the block is the parity block or the data block. The minimum number of blocks to pick up is k commonly, which indicates that the increase in parity block numbers or data block numbers will not lead to the growth of the decoding time. This rule leads to the relative stable decoding curve among three schemes.

The overhead of the encoding is larger than that of the decoding so that the encoding time is more susceptible by the block numbers. The encoding curve is steeper than the decoding curve. Since all the blocks need to be figured in the encoding operations, the decoding operations only require k blocks. The more block number to calculate, the less the performance is.

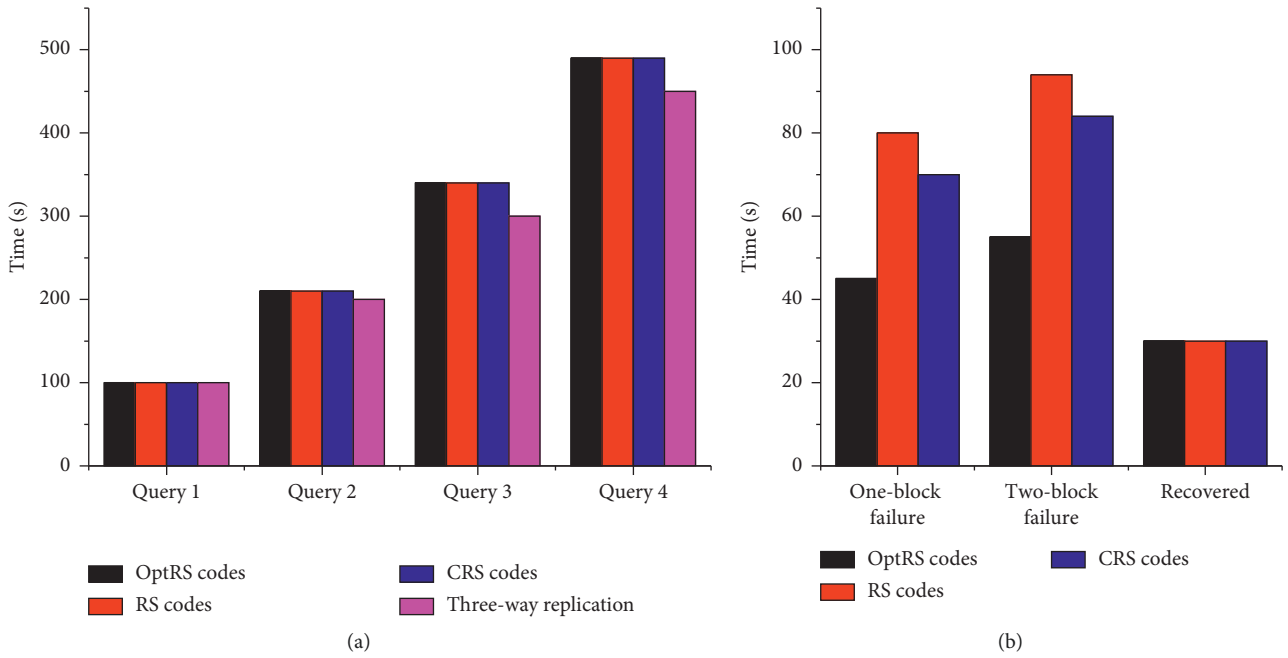


Figure 13: Reading performance. (a) Normal reading performance. (b) Degraded reading performance.

Since standard bit arithmetic (addition is XOR and multiplication is bitwise AND) is used in OptRS codes and CRS codes, they require lower encoding/decoding time than RS codes. The reason why OptRS codes have lower encoding/decoding time than CRS codes is that OptRS codes adopt the coding optimization and adaptively select different RE schemes. OptRS codes are better than the other schemes in both the encoding time and the decoding time, which indicate OptRS codes are more efficient than the others.

At last, we set $k = 5$, $m = 2$, and let s range from $1M$ to $10M$, while the results are shown in Figure 12(c). From the figure, we can see that the encoding and decoding time increase with the block size rises because of more data to deal with. For the same block size, decoding time is less than encoding performance which is the same reason as the last two experiments.

Figure 12 shows the encoding and decoding time among RS codes, CRS codes, and OptRS codes curve with different parameters. The relationship is obviously positive between encoding/decoding time and data block numbers, as the same as to the block size. When data block number increases, encoding/decoding time increases, while the increment is less than the case when block size increases. The parity block number is the least influential one in all the parameters. Encoding time is more than decoding time and it fluctuates even more. Since it is more suitable to be deployed on distributed systems, we should pay more attention to improve the encoding performance in the future.

In a word, OptRS codes have a higher encoding and decoding performance than CRS codes and RS codes in distributed systems. We can conclude that the encoding performance of OptRS codes is 36.1% higher on average than that of CRS codes and 58.2% higher than that of RS codes. The decoding performance of OptRS codes is 19.3% and 33.1% higher than that of CRS codes and RS codes, respectively.

6.2. Reading Performance. In the first experiment, four kinds of query sets in different sizes are used to test the normal reading performance with TPC-C benchmark and 1 GB workload. We add three-way replication for comparison, and the results are shown in Figure 13(a). Three-way replication provides a higher reading performance than erasure codes, while the three erasure codes have the same reading performance. This is because the three-way replication keeps three copies of original data, which can shorten the read latency a lot. But erasure codes only keep one copy. At the same time, the system's reading performance benefits from light workloads so that the time of *query*1 set is the least.

When one or more blocks are corrupted, the system enters the degraded reading mode. We compare the reading time from one-block corrupted, two-block corrupted, and block recovered with a 256 MB workload. The results are shown in Figure 13(b). The time of one-block corrupted is lower than that of two-block corrupted and higher than that of block recovered. The reason why one-block corrupted has a higher reading performance than two-block corrupted is that two-block corrupted needs more time to reconstruct the data. After all the data are recovered, the system reading performance becomes normal. OptRS codes have good enough decoding performance to recover most corrupted data blocks before users read them so that the degraded reading performance could be much better than that of RS codes and CRS codes in the first two situations.

7. Related Works

Nowadays, many international companies have applied erasure codes to save storage costs in their distributed storage systems, such as Google [15], IOM [16], and Facebook [17]. The problem of efficient node repair has already been studied and most solutions proposed have used MDS codes, which can recover from any $n - k$ nodes. MDS codes have the smallest storage overhead n/k when k is fixed and are the popular family of codes in distributed storage systems.

A big family MDS codes are RS codes, which can reach higher reliability than RAID so that they are popular in many fields.

However, the solutions usually have common disadvantages: high recovery cost. A lot of methods have been proposed to decrease this. Rashmi et al. [18] have developed Hitchhiker, a novel encoding and decoding technique that reduces both network traffic and disk I/O a lot during reconstruction. At the same time, Hitchhiker can reduce the latency of degraded reads and perform faster recovery from failed or decommissioned machines. Khan et al. [19] have presented an algorithm that found the optimal number of codeword symbols needed for recovery for any XOR-based erasure code and produced recovery schedules that use a minimum amount of data. They also demonstrated that the differences improve I/O performance in practice for the large block sizes used in cloud file systems.

Our team has done a lot of research already [20]. We have presented a storage system based on random network coding that optimizes resources consumed during reconstruction in [21]. The data were only functional reconstructed to the failed data so that the system needed to read the least data to reconstruct when data loss. Robot [22] is a solution based on coding technologies in the big data system that stores a lot of cold data. By studying Reed–Solomon coding technologies and big data systems, robot can not only maintain the system reliability but also improve the security and the utilization of storage systems.

Besides RS codes, there are many other popular code families in distributed storage systems, such as LRC [23] and minimum storage regeneration (MSR code in short) [24].

In [25], LRC has been used to store warm data and to store cold data based on the access characteristics of the data in distributed storage systems. At the same time, an RS code is used to store cold data after optimizing for storage overhead to reduce the storage burden. Mingyuan Xia et al. [26] have invented a new erasure-coded storage system that instead uses two different erasure codes and dynamically adapts to workload changes. It used a fast code to optimize for recovery performance and a compact code to reduce the storage overhead. A novel conversion mechanism was used to efficiently upcode and downcode data blocks between fast and compact codes. HACFS design techniques were generic and successfully applied to two different code families: product and LRC codes.

MSR code is now more popular because it has the smallest possible repair bandwidth in MDS codes. It acts as a superior alternative to RS codes and other codes. Assuming there are a bytes in each failed node, MSR code introduces an arbitrarily chosen subset of d helper nodes. Comparing the traditional MDS code, the total amount db of bytes downloaded is typically much smaller than the total amount ka bytes of data stored in k nodes. KV Rashmi et al. [27] built MSR code into consideration as a superior alternative to RS codes. MSR code can minimize network transfers during reconstruction while guaranteeing reliability. They can also be simultaneously optimal in terms of I/O, storage, and network bandwidth. In [28], MSR code has been used to repair invalid nodes and adding new nodes suggests in single- and multiple-node storage situations. The two processes can be unified in both concurrent and sequential for constructing multiple nodes.

The Butterfly code [29] has been implemented and experimentally validated the theoretically proven benefits of reduced data download for node repair. However, the test results told that the interaction between different distributed system layers had a significant impact on the code repair performance. HTECs [30] have been invented to provide the lowest data-read, data-transfer, and repair time when the repair process was linear and parallel. HTECs can permit flexibility at the expense of repair bandwidth.

MSR and the bandwidth optimization will be integrated into OptRS code frameworks after inspiring some ideas in our future works.

8. Conclusion

Erasure codes bring the low storage overhead in distributed storage systems meanwhile they take many performance problems. Focusing on this, we have proposed OptRS codes that extend the theoretical construction with practical considerations that lead directly to implementation. OptRS codes can optimize matrix calculation by mapping the calculation of the matrix into the XOR operation, which improves the performance of encoding and decoding greatly. Two rows and three rows elimination schemes can decrease the times of XOR operations theoretically, and test result shows that the effect is good. It is our belief that OptRS codes are well poised to make the leap from theory to practice.

We believe our OptRS codes are a first step towards improving the performance of storage systems. Several challenges remain as future work, such as reducing the decoding time further and integrating OptRS codes into HDFS. Since MSR code is more and more popular, we will consider doing some work on it.

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 work was supported in part by the National Natural Science Foundation of China under Grant 61662038 and Grant 61662039, in part by the fund of the Science and Technology Project of the Jiangxi Provincial Department of Education under Grant GJJ151081 and Grant GJJ161072, in part by the Visiting Scholar Funds by China Scholarship Council, in part by the JiangXi Association for Science and Technology, and in part by the Jiangxi Province Key Natural Science Foundation under Grant 20192ACBL20031.

References

[1] D. Ford, F. Labelle, F. I. Popovici et al., "Availability in globally distributed storage systems," in *Proceedings of the Operating Systems Design and Implementation*, pp. 61–74, Vancouver, BC, Canada, October 2010.

[2] W. Jiang, C. Hu, Y. Zhou, and A. Kanevsky, "Are disks the dominant contributor for storage failures?," *ACM Transactions on Storage*, vol. 4, no. 3, pp. 1–25, 2008.

[3] B. Schroeder and G. A. Gibson, "Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you?," in *Proceedings of the FAST'07: 5th USENIX Conference on File and Storage Technologies*, pp. 1–16, Berkeley, CA, USA, February 2007.

[4] Amazon simple storage service (S3), 2019, https://aws.amazon.com/s3/?nc1=h_ls.

[5] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *MSST'10 Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies*, pp. 1–10, Lake Tahoe, NV, USA, 2010.

[6] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google file system," in *Proceedings of the SOSP*, pp. 29–43, Bolton Landing, NY, USA, October 2003.

[7] 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.

[8] J. Blomer, M. Kalfane, M. Karpinski, R. Karp, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme," Technical Report TR-95-048, International Computer Science Institute, Berkeley, CA, USA, 1995.

[9] S. Muralidhar, W. Lloyd, S. Roy et al., "Facebook's warm BLOB storage system," in *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, pp. 383–398, Broomfield, CO, USA, October 2014.

[10] J. S. Plank, K. M. Greenan, and E. L. Miller, "Screaming fast Galois Field arithmetic using Intel SIMD instructions," in *Proceedings of the FAST*, pp. 299–306, San Jose, CA, USA, 2013.

[11] J. S. Plank and L. Xu, "Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications," in *Proceeding of the Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)*, pp. 108–116, Cambridge, MA, USA, 2006.

[12] J. S. Plank, S. Simmerman, and C. D. Schuman, "Jerasure: a library in C/C++ facilitating erasure coding for storage applications," Tech. Rep. CS-08-627, University of Tennessee, Knoxville, TN, USA, 2008.

[13] J. L. Hafner, "HoVer erasure codes for disk arrays," Technical Report RJ10352 (A0507-015), IOM Research Division, Grand-Saconnex, Switzerland, 2005.

[14] L. Xu and J. Bruck, "X-Code: MDS array codes with optimal encoding," *IEEE Transactions on Information Theory*, vol. 45, no. 1, pp. 272–276, 1999.

[15] Colossus, Successor to Google File System, 2010, http://static.googleusercontent.com/media/research.google.com/en/us/university/relations/facultysummit2010/storage_architecture_and_challenges.pdf.

[16] An introduction to GPFS Version 3.5, 2015, http://www-03.iOM.com/systems/resources/introduction-to-gpfs-3-5.pdf.

[17] Facebooks Erasure Coded Hadoop Distributed File System (HDFS-RAID), 2015, https://github.com/facebook/hadoop-20.

[18] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A Hitchhikers guide to fast and efficient data reconstruction in erasure-coded data centers," in *Proceeding of the SIGCOMM*, pp. 331–342, London, UK, 2015.

[19] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads," in *Proceeding of the FAST*, pp. 251–264, San Jose, CA, USA, 2012.

[20] C. Yin, J. Wang, H. Lv et al., "An optimized algorithm based on CRS codes in big data storage systems," in *Proceedings of the Algorithms and Architectures for Parallel Processing*, pp. 228–240, Zhangjiajie, Hunan, China, 2015.

[21] C. Yin, C. Xie, J. Wan, C.-C. Hung, J. Liu, and Y. Lan, "BMCloud: minimizing repair bandwidth and maintenance cost in cloud storage," *Mathematical Problems in Engineering*, vol. 2013, Article ID 756185, 11 pages, 2013.

[22] C. Yin, J. Wang, C. Xie, J. Wan, C. Long, and W. Bi, "Robot: an efficient model for big data storage systems based on erasure coding," in *Proceedings of the 2013 IEEE International Conference on Big Data*, pp. 163–168, Santa Clara, CA, USA, October 2013.

[23] 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.

[24] 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.

[25] B. Wei, L.-M. Xiao, W. Wei, Y. Song, and B.-Y. Zhou, "A new adaptive coding selection method for distributed storage systems," *IEEE Access*, vol. 6, no. 4, pp. 13350–13357, 2018.

[26] M. Xia, M. Saxena, and M. Blaum, "A tale of two erasure codes in HDFS," in *Proceedings of the FAST*, pp. 213–226, Santa Clara, CA, USA, 2015.

[27] K. V. Rashmi, P. Nakkiran, J. Wang, N. B. Shah, and K. Ramchandran, "Having your cake and eating it too: jointly optimal erasure codes for I/O, storage, and network bandwidth," in *Proceedings of the FAST*, pp. 81–94, Santa Clara, CA, USA, 2015.

[28] H. Zhang, H. Li, B. Zhu, X. Yang, and S.-Y. R. Li, "Minimum storage regenerating codes for scalable distributed storage," *IEEE Access*, vol. 5, no. 4, pp. 7149–7155, 2017.

[29] L. P. Juarez, F. Blagojevic, R. Mateescu, C. Guyot, E. E. Gad, and Z. Bandic, "Opening the chrysalis: on the real repair performance of MSR codes," in *Proceedings of the FAST*, pp. 81–94, Santa Clara, CA, USA, 2016.

[30] K. Kralevska, D. Gligoroski, R. E. Jensen, and H. verby, "HashTag erasure codes: from theory to practice," *IEEE Transactions on Big Data*, vol. 4, no. 4, pp. 226–239, 2018.