Research on Multi-fault-tolerant MDS Array Erasure code

Wei Yuan, Ying Yu, Yan Gao, Dan Tang*

Department of Software engineering, Chengdu University of Information Technology, Chengdu 610000, China
731702594@qq.com

Abstract. The array code has the advantages of high computational efficiency and simple structure. And the erasure code strategy based on the array code is one of the ideal fault-tolerant methods for distributed storage systems. However, low fault tolerance has always been a major obstacle to the practicality of array code. According to the known literature, the fault tolerance of MDS array code is no more than three. And the array code with the biggest fault tolerance is the Grid code. Its fault tolerance is usually only 15. In view of this situation, this paper proposes a kind of multi-fault-tolerant MDS array code, called Reed-Solomon Array (RA). The RA code has higher coding efficiency, whose encoding process and decoding process only involve binary XOR operations. What is more, the construction method is simple and easy to be implemented by software and hardware. And then the RA code is a MDS code with optimal storage efficiency. Finally the size of the storage array of the RA code is not strictly limited. The extensibility of the distributed storage system is met.

1. Introduction

With the continuous development of information technology in various industries and fields, the data volume shows an exponential growth trend. With the rapid increase of data volume, the problem of the reliability of data storage comes along. The continuous growth of storage data in various industries leads to the increasing scale of distributed storage system and the increasing number of nodes. Companies such as Google and Facebook already have storage systems with more than 3,000 nodes[1]. As the number of nodes increases, the probability of node failure increases. Therefore, how to provide a secure environment for distributed storage system and keep the high reliability and high availability of data has become an essential issue of distributed secure storage at this stage.

At present, the most common data fault-tolerant technology in distributed storage system is multi-copy strategy. The strategy is to store n copies of data replication on n different nodes to achieve redundant back-up. When n-1 nodes failed at the same time, the data could be recovered effectively. Most of the data reliability enhancement methods used in commercial storage systems are multi-copy strategy. The famous distributed storage system GFS [2] and Hadoop [3] all adopt this method. This method does not need special encoding algorithm or decoding algorithm, and it has good fault tolerance but low space utilization. If the fault-tolerant capability is $n - 1$, the space utilization rate is only $1/n$. As fault tolerance increases, declining storage efficiency becomes a major shortcoming of distributed storage systems.

Erasure code strategy is a method to enhance the reliability of storage system which has attracted much attention in recent years. Compared with the multi-copy strategy, the biggest advantage of the erasure code strategy is that it can effectively reduce the update cost and improve the storage efficiency while guaranteeing the fault-tolerant ability. For the research on fault tolerance mechanism
of building storage system based on erasure code strategy, there are the Plank team of Tennessee University and the Blaum team of IBM in foreign countries [4,5], Shu Jiwu team of Tsinghua University and Xuxulong team of China University of Science and Technology in domestic [6].

At present, there are two main aspects in the research of erasure code strategy. One is RS (Reed-Solomon) erasure code. RS code is MDS code, which can correct arbitrary errors. However, the computation of RS code involves operations over multivariate finite field with high computational complexity, and the computational cost is too large in large-scale distributed systems. In order to solve this problem, some scholars have made in-depth research on it and proposed some improved methods. The most typical method is to improve the calculating speed by converting multivariate finite field operations to binary field operations[7,8]. Other scholars have also made outstanding contributions in improving the efficiency of multivariate finite fields, such as GF-complete proposed by Plank [9], which improves the computation speed of multivariate finite fields by modifying the instruction set. The other research of erasure code strategy is the array code. The encoding process of the array code is only based on binary exclusive XOR operation. So it has the advantages of high computational efficiency and simple implementation process. These advantages make array code ideal for large-scale distributed storage systems. The typical MDS codes have EVENODD code [5] and X code [10] that can tolerate two faults, and star code [11] that can tolerate three faults. However, there has been no MDS array code with a fault tolerance greater than 3 so far. In order to improving the fault tolerance, the researchers designed some non-MDS code at the expense of certain storage efficiency, which can tolerate more than three disk errors. Among them, the Weaver [12] code and the Grid [6] code are typical. Although this kind of array code has improved its fault-tolerant ability, its application in distributed storage system still has various problems. First, there are restrictions on the number of stripes or strips. Second, the storage efficiency decrease with the increase of fault tolerance. Therefore, the array code is less practical in the field of distributed storage systems due to these problems. In view of this situation, this paper proposes an array erasure code to form an array of Cauchy RS code. The array code can theoretically tolerate any number of faults and reaching the Shannon limit. It is a MDS code. And there is no limit to the number of strips or stripes. It is of great significance to improve the practicability of array code in the field of fault tolerance of distributed storage systems.

The organization of this paper is as follows: The next section will provide a brief introduction to the basic concepts to be used in the arraying process. We will detail the construction process of the RA code in Section 3, including encode and data recovery. Then We give the proof and the analysis of the basic characteristics of the RA code in Section 4, and conclude in Section 5.

2. Binary Matrix Representation of Elements in Finite Fields

In [14], a scheme is mentioned in which a binary matrix is used replaced a finite field element for multiplication, and the operation requires only a simple XOR operation. The binary matrix is composed of coefficient vectors of elements in a finite field.

Let the irreducible polynomial of degree w in $GF(2^w)$ be $q(x)$. Polynomial $f(x) = \sum_{i=0}^{w-1} f_i x^i, f_i \in GF(2)$ can represent all elements on $GF(2^w)$. The column vector $V(x) = (f_0, f_1, \ldots, f_{w-1})^T$ is called the coefficient vector of polynomial $f(x)$.

Definition 2.1: For any $e \in GF(2^w)$, let $\beta(e)$ be the matrix whose column i is the coefficient vector of $x^{-i}e \mod p(x)$.

Binary matrix $\beta(e)$ has the following lemma:

Lemma 2.1: $\beta$ is a field is isomorphic from $GF(2^w)$ to $\beta(GF(2^w))$. In particular,
- $\beta(0)$ is the matrix that all element are zero.
- $\beta(1)$ is the identity matrix.
- $\beta$ is the injective mapping.
For any two elements \( a, b \in GF(2^w) \), \( \beta(a + b) = \beta(a) + \beta(b) \).

For any two elements \( a, b \in GF(2^w) \), \( \beta(ab) = \beta(a)\beta(b) \).

For a detailed proof of lemma 2.1, please refer to the literature [9], but it is obvious that it is configurable, and the multiplication of the elements on the finite field can be converted into the multiplication of the matrix on \( \beta \).

3. RA code: A class of MDS array code with high fault tolerance

3.1. Encoding process

The RA code proposed in this paper replaces the elements in the Cauchy matrix of the Cauchy RS code into a binary matrix, and then converts the operation of the multiple finite field of the Cauchy RS code into a binary XOR operation, reducing the complexity of the operation. The replaced binary matrix is converted into an array, and then the array is simplified according to a certain method to obtain a final coded array. On \( GF(2^w) \), an Cauchy RS matrix of \((n, m, w)\) (\(n\) is the number of blocks of the data block, \(m\) is the number of blocks of the check block) is constructed and arrayed. The specific arraying steps are as follows:

Step 1: Construct a finite field \( GF(2^w) \) and corresponding binary matrix.

The primitive polynomial \( x^a + x^b + \cdots + x^c + 1, (a > b > \cdots > c) \) of the root \( \alpha \) is generated to generate a finite field \( GF(2^w) \), that is, \( \alpha^a = \alpha^b + \cdots + \alpha^c + 1 \). Construction process is as follows:

\[
\begin{align*}
\alpha^0 &= 1 \\
\alpha^1 &= \alpha \\
\alpha^{a+b} &= \alpha^a \cdot \alpha = \alpha^{a+b} + a + \alpha \\
&\vdots \\
\alpha^{a+b} &= \alpha^a \cdot \alpha^{a+b} = \alpha^a + a + \alpha \\
&\vdots \\
\alpha^{a+b} &= \alpha^a + \cdots + \alpha^c + 1 \\
\alpha^{a+b} &= \alpha^a + \cdots + \alpha^c + 1
\end{align*}
\]

The above is a polynomial element representation in the finite field. Next, according to definition 2.1, the binary matrix \( \beta(e) \) is constructed.

Step 2: Building a Cauchy Matrix.

And, \(m\) elements and \(n\) elements are selected from \( GF(2^w) \) to form \( X = \{x_1, x_2, \ldots, x_m\} \) and \( Y = \{y_1, y_2, \ldots, y_n\} \), respectively, and the Cauchy matrix is constructed.

Step 3: Extending Cauchy Matrix to Binary Matrix.

Replacing the elements in the Cauchy matrix obtained in step 2 with the corresponding binary matrix \( \beta(e) \) obtained in step 1. The Cauchy matrix of size \( m \times n \) is expanded into a binary matrix of size \( mw \times nw \). Let the expanded binary matrix be \( R \).

Step 4: Building the array.

First, each of the \(n\) data blocks is divided into \(w\) shares, such as dividing \( D_1, D_2, D_3, \ldots, D_n \) into \( D_{1,1}, D_{1,2}, \ldots, D_{1,w}, D_{2,1}, \ldots, D_{n,w} \).

\[
C = R \times D = \begin{bmatrix}
R_{1,1} & R_{1,2} & \cdots & R_{1,w} \\
R_{2,1} & R_{2,2} & \cdots & R_{2,w} \\
\vdots & \vdots & \ddots & \vdots \\
R_{w,1} & R_{w,2} & \cdots & R_{w,w}
\end{bmatrix} \times
\begin{bmatrix}
D_{1,1} \\
D_{1,2} \\
\vdots \\
D_{1,w} \\
D_{2,1} \\
\vdots \\
D_{n,1} \\
D_{n,w}
\end{bmatrix} = \begin{bmatrix}
C_{1,1} \\
C_{1,2} \\
\vdots \\
C_{1,w} \\
C_{2,1} \\
\vdots \\
C_{w,1} \\
C_{w,w}
\end{bmatrix}
\]
\[ C_{1,1} = (R_{1,1} \cdot D_{1,1}) \oplus (R_{1,2} \cdot D_{1,2}) \oplus \cdots \oplus (R_{1,w} \cdot D_{1,w}) \]

The resulting array is shown below:

\[
Z_1 = \begin{bmatrix}
D_{1,1} & D_{1,2} & \cdots & D_{1,n} & C_{1,1} & C_{2,1} & \cdots & C_{m,1} \\
D_{1,2} & D_{1,2} & \cdots & D_{1,2} & C_{1,1} & C_{2,2} & \cdots & C_{m,2} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
D_{1,w} & D_{2,w} & \cdots & D_{n,w} & C_{1,w} & C_{2,w} & \cdots & C_{m,w}
\end{bmatrix}
\]

Step 5: Operation optimization.

Calculating the number of occurrences of the \( D_{i_1, j_1} \oplus D_{i_2, j_2} \) and \( D_{i_1, j_1} \oplus D_{i_2, j_2} \oplus D_{i_3, j_3} \) \((i_1, i_2, i_3 \in \{1, 2, \cdots, n\}, j_1, j_2, j_3 \in \{1, 2, \cdots, w\})\) types in the array.

The expression in which the number of occurrences exceeds one is multiplied by 2 according to the number of times of the \( D_{i_1, j_1} \oplus D_{i_2, j_2} \) type, and the result of multiplying the number of times of the \( D_{i_1, j_1} \oplus D_{i_2, j_2} \oplus D_{i_3, j_3} \) type by 3 is sorted from large to small. The array is replaced by the sort result, the replacement success is marked as \( S_i \), and the \( i \) is the replacement sequence number. After the replacement is completed, the replacement list \( S \) and the last optimized array \( Z_2 \) are obtained.

The arrayed process only needs to be executed once to get arrays \( Z_1 \) and \( Z_2 \) without repeated execution. Based on the replacement list, the values of the individual substitution formulas are calculated. After that, according to the array, redundant data is calculated and the encode is completed.

### 3.2. Decoding process

In this paper, the array obtained by arraying has no fixed law, so there is no decoding method can operate according to the rules of its own array. If the decoding algorithm based on the RS Cauchy code still involves the operation of the multivariate finite field and the matrix inversion, the operation complexity is large. In [15], a general decoding method for array code is proposed. The decoding procedure is simple, and the size of the array is not limited. The calculation is binary-XOR operation. It is an ideal decoding algorithm for RA code. Therefore, this paper appropriately modifies the matrix decoding method proposed in [15] as the decoding method of RA code. The general steps are as follows:

**Step 1: Constructing a Decoding Matrix.**

First, according to the array \( Z_1 \) is divided into \((n+m)\times w\) blocks, the matrix \( B \) of the size \((n+m)w\times(n+m)w\) is created. The structure of the matrix \( B \) is \( B = \left( \frac{I}{O} \right) \). And the \( I \) is an identity matrix of size \( nw\times nw \). The \( O \) is a zero matrix of size \( nw\times mw \). The \( H \) is a parity check matrix of \( Z_1 \).

**Step 2:** Calculating the data needed for lost data recovery.

**Step 2-1:** When the node fails, the data of the entire data column is lost. So when restoring data, a corresponding list \( L \) of missing elements is created according to the missing nodes.

**Step 2-2:** The element \( e \) in the missing element list \( L \) is iterated through. If it belongs to the data element, proceed to step 2-3. If it belongs to the redundant element, skip it.

**Step 2-3:** The \( r \) is the set of rows whose column \( e \) is 1 in the check matrix \( H \). If there is no element in \( r \), set row \( e \) to zero (indicating that the element cannot be restored theoretically). If there is an element in \( r \), then we select the row with the smallest weight of Hamming after removing the missing element from \( r \) as \( r_j \). The row corresponding to the element in \( r \) and the row \( e \) are XOR with \( r_j \), and finally the value in \( r_j \) is set to 0.

**Step 2-4:** After all the data elements in \( L \) are processed, the values in the columns corresponding to the redundant elements in \( L \) are all set to zero.
Step 3: Recovering data.

After completing step 2, row $r_e$ is the row corresponding to the element $e$ of the missing element list $L$ in the corresponding matrix $B$. The $c_e$ is a set of columns with a value of $1$ in $r_e$. The data block corresponding to the element in $c_e$ is the data block required for the recovery element $e$. According to the missing element list $L$ and the matrix $B$, data blocks needed to recover all lost data can be obtained. Corresponding lost data can be obtained by XOR calculation of required data block.

3.3. Specific examples

The construction process of the $(3,2,3)$ (the number of data blocks is $3$, the number of check blocks is $2$, and the Hamming distance is $3$) RA code on the finite field $GF(2^3)$ where the primitive polynomial is $x^3 + x + 1$.

(1) Encoding process

The root of primitive polynomial $x^3 + x + 1$ with finite field $GF(2^3)$ is $\alpha$. And $\alpha^3 = \alpha + 1$. The construction process is as follows:

\[
\begin{align*}
\alpha^0 &= 1 & \alpha^4 &= \alpha^2 \times \alpha = \alpha^2 + \alpha \\
\alpha^1 &= \alpha & \alpha^5 &= \alpha^4 \times \alpha = \alpha^4 + \alpha^3 + \alpha + 1 \\
\alpha^2 &= \alpha^2 & \alpha^6 &= \alpha^5 \times \alpha = \alpha^5 + \alpha^4 + \alpha + 1 \\
\end{align*}
\]

A comparison table of the finite field element $e$ and the coefficient vector $V(e)$, the binary matrix $\beta(e)$ is constructed.

| $e$  | 0 | 1 | $\alpha$ | $\alpha^2$ | $\alpha^3$ | $\alpha^4$ | $\alpha^5$ | $\alpha^6$ |
|-----|---|---|---------|---------|---------|---------|---------|---------|
| Polynomial representation | 0 | 1 | $\alpha$ | $\alpha^2$ | $\alpha^3$ | $\alpha^4$ | $\alpha^5$ | $\alpha^6$ |
| Binary representation | 000 | 001 | 010 | 100 | 011 | 110 | 111 | 101 |
| $V(e)$ | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| $\beta(e)$ | 0 0 0 0 1 0 1 | 0 0 1 0 1 0 1 | 0 0 1 0 1 0 1 | 0 0 1 0 1 0 1 |

Selecting $X = \{1,2\}, Y = \{0,3,4\}$ from $GF(2^3)$. The constructed Cauchy matrix is $\begin{bmatrix} 1 & 5 & 2 \\ 5 & 1 & 3 \end{bmatrix}$.

We take out 12 raw data blocks, corresponding to $D_{1,1}, \cdots, D_{1,3}, D_{2,1}, \cdots, D_{4,3}$. Calculating the redundant array. $C = R \times D$.

The resulting array is shown below:

\[
ZI = \begin{bmatrix}
D_{1,1} & D_{2,1} & D_{3,1} & D_{1,2} \oplus D_{2,2} \oplus D_{3,2} & D_{1,3} \oplus D_{2,3} \oplus D_{3,3} \\
D_{1,2} & D_{2,2} & D_{3,2} & D_{1,1} \oplus D_{2,1} \oplus D_{3,1} & D_{1,3} \oplus D_{2,3} \oplus D_{3,3} \\
D_{1,3} & D_{2,3} & D_{3,3} & D_{1,1} \oplus D_{2,1} \oplus D_{3,1} & D_{1,2} \oplus D_{2,2} \oplus D_{3,2} \\
\end{bmatrix}
\]

The array is optimized for operation.

Calculating the number of occurrences of the $D_{i1,j1} \oplus D_{i2,j2}$ and $D_{i1,j1} \oplus D_{i2,j2} \oplus D_{i3,j3}$ ($i1,i2,i3 \in \{1,2,3,4\}, j1,j2,j3 \in \{1,2,3\}$) types in the array. The expression in which the number of
occurrences exceeds one is multiplied by 2 according to the number of times of the $D_{ij} \oplus D_{ij}$ type, and the result of multiplying the number of times of the $D_{ij} \oplus D_{ij} \oplus D_{ij}$ type by 3 is sorted from large to small. For reasons of space, it is not listed in the text. The array is replaced in order. The successful replacement formula is as follows:

$$S_1 = D_{1,1} \oplus D_{1,1} \oplus D_{2,1}; S_2 = D_{2,1} \oplus D_{3,1} \oplus D_{3,1}; S_3 = D_{1,3} \oplus D_{3,3};$$

The replaced array is as follows:

$$Z2 = \begin{bmatrix}
D_{1,1} & D_{1,1} & D_{2,1} & D_{2,1} & D_{3,1} & D_{3,1} \\
D_{1,2} & D_{1,2} & D_{2,2} & D_{2,2} & D_{3,2} & D_{3,2} \\
D_{1,3} & D_{1,3} & D_{2,3} & D_{2,3} & D_{3,3} & D_{3,3}
\end{bmatrix}$$

(2) Decoding process

This example is to build a $(3, 2, 3)$ RA code with a fault tolerance of 2. We assume that the 1 and 5 nodes fail. The following is the specific process of decoding:

First, we construct a decoding matrix $B1$ of size $15 \times 15$. The rows and columns of matrix B correspond to 15 data blocks in the array, respectively. According to the invalid nodes 1 and 5, we can get the list of missing elements $L = \{0, 1, 2, 12, 13, 14\}$. $[0, 1, 2]$ belongs to the data element. And $[12, 13, 14]$ belongs to the redundant element. We loop through the list of missing elements $L$ for data recovery. When the missing element $e = 0$, $0$ belongs to the data element. The $r$ is a set of rows with a value of 1 in the 0th column of the check matrix $H$. So $r = 9, 12, 14$. Because 12 and 14 is the missing element, it is excluded. The Hamming weighs of row 9 is the smallest, so we choose $r = 9$. After the 9th row is XOR with the 0th, 12th and 14th row, the value of the 9th line is set to 0. The result obtained is matrix $B2$.

We continue to loop through the list of missing elements $L$. However, due to space reasons, detailed steps are not given. When all missing data elements are looped, the value of the column corresponding to the missing redundant element is set to zero. Finally, the transformed matrix is $B3$.

The $r_e$ is the corresponding row of the element $e$ in the missing element list $L$ in the matrix $B$. The $c_e$ is a collection of columns with a value of 1 in row $r_e$. The data block corresponding to the element in $c_e$ is the data block required to recover the element $e$. The expression for recovering the missing element is as follows:

$$D_{1,1} = D_{1,2} \oplus D_{1,2} \oplus D_{1,3} \oplus C_{1,1} \quad D_{1,2} = D_{2,2} \oplus D_{1,1} \oplus D_{1,3} \oplus C_{1,2} \quad D_{1,3} = D_{3,2} \oplus D_{1,1} \oplus C_{1,3}$$

$$C_{1,1} = D_{2,2} \oplus D_{2,3} \oplus C_{1,2} \quad C_{2,2} = D_{1,1} \oplus D_{1,1} \oplus D_{1,3} \oplus C_{1,3} \quad C_{2,3} = D_{3,2} \oplus D_{2,3} \oplus D_{2,3} \oplus C_{1,3}$$

Finally, the missing element data is calculated according to the expression. The decoding is complete.
4. Performance analysis and comparison

This section will further analyze the performance of the RA code system for storage systems, especially distributed storage systems, for the requirements of fault-tolerant code.

4.1. MDS property

The RA code is evolved from the RS Cauchy code array. The RA code also inherits the MDS property of the RS Cauchy code.

Proof:

The check matrix of code is closely related to the fault tolerance of code. The structure of the check matrix of the RA code is \( H = [R \mid I] \), where \( I \) is the identity matrix. The structure of the matrix \( R \) is:

\[
R = \begin{bmatrix}
R_{1,1} & R_{1,2} & \cdots & R_{1,w} \\
R_{2,1} & R_{2,2} & \cdots & R_{2,w} \\
\vdots & \vdots & \ddots & \vdots \\
R_{w,1} & R_{w,2} & \cdots & R_{w,w}
\end{bmatrix}
\]

One node of the RA code corresponds to the elements on \( w \) rows and \( w \) columns of the matrix. When the node fails, it will affect the elements on \( w \) rows and \( w \) columns of the matrix. According to this property, the \( H \) can be transformed into \( H_1 \). The structure of the matrix \( H_1 \) is \( H_1 = [K \mid I] \).

And the structure of the matrix \( K \) is:

\[
K = \begin{bmatrix}
K_{1,1} & K_{1,2} & \cdots & K_{1,w} \\
K_{2,1} & K_{2,2} & \cdots & K_{2,w} \\
\vdots & \vdots & \ddots & \vdots \\
K_{m,1} & K_{m,2} & \cdots & K_{m,w}
\end{bmatrix}, \quad K_1 = \begin{bmatrix}
R_{1,1} & R_{1,2} & \cdots & R_{1,w} \\
R_{2,1} & R_{2,2} & \cdots & R_{2,w} \\
\vdots & \vdots & \ddots & \vdots \\
R_{w,1} & R_{w,2} & \cdots & R_{w,w}
\end{bmatrix}
\]

It is obvious that \( K_{i,j} (i \in \{1, 2, \cdots, m\}, j \in \{1, 2, \cdots, n\}) \) is the binary matrix corresponding to the Cauchy matrix elements calculated by the element sets \( X \) and \( Y \). So \( K \) is equivalent to a Cauchy matrix. The \( H_1 \) consists of a Cauchy matrix and an identity matrix, which is undoubtedly a full rank matrix. So \( m \) faults can be tolerated by RA code, equaling to the number of redundant blocks. Therefore, the RA code has MDS properties.

4.2. Theoretically unrestricted fault tolerance and weak constraints

The RA code is the code obtained by arraying the RS Cauchy code. The fault tolerance of RS Cauchy code is theoretically unrestricted. The fault tolerance of the RA code is the same as that of the RS Cauchy code and is not limited. Fault tolerance is an important factor affecting the usability of array code. Table 1 lists the fault tolerance capabilities of the main types of array code currently in the storage fault tolerance field and whether they have MDS properties. It can be seen from Table 1 that the fault tolerance of the MDS array code is less than or equal to 3, and the array code with a fault tolerance greater than 3 does not have the MDS property. Current multi-fault-tolerant array codes are at the expense of storage efficiency for fault tolerance. As the fault tolerance is greater, the storage efficiency is lower, even with the storage efficiency of the multi-copy strategy.

Another factor affecting the usability of the array code is the limitation of the size of the array during the construction of the array code, that is, the constraint on the stripe or the strip. Table 1 lists the comparison of the size requirements about stripe and strip for the major types of array code in the storage fault tolerance field. As can be seen from the table, most of the array codes have strict requirements on the stripe size of the storage array and the strip size of the storage array are strictly limited, and most of them are prime numbers. This requirement has a great limitations on the application of array code to distributed storage systems, which greatly affects the extensibility of distributed storage systems and is contrary to the extensibility of distributed storage systems. The Weaver code has high fault tolerance and does not have specific requirements for stripe size. However,
this code does not have a systematic encoding method and lacks theoretical support. It is not suitable for large-scale distributed storage systems. Unlike other array code, the RA code proposed in this paper has no limitation on the stripe size. In construction, only the number of data nodes and the number of redundant nodes (i.e., fault tolerance) need to be considered. The distributed storage system that uses it has great extensibility and practicability.

| Array code | Fault tolerance | Whether it has MDS properties | stripe size limits | stripe size limits |
|------------|-----------------|-------------------------------|-------------------|-------------------|
| EVENODD code | 2               | yes                           | Prime number      | stripe size-1     |
| Star code   | 3               | yes                           | Prime number      | stripe size-1     |
| X code      | 2               | yes                           | Prime number      | stripe size       |
| Weaver code | 12              | no                            | None              | None              |
| Grid code   | 15              | no                            | Depending on the stripe size of the matched code | Depending on the stripe size of the matched code |
| RA code     | arbitrarily     | yes                           | None              | None              |

4.3. Higher computing efficiency
The computational efficiency of encoding and decoding greatly affects the usability of array code in distributed storage systems. Therefore, the computational efficiency of encoding and decoding becomes an important indicator for evaluating erasure code. In terms of computational efficiency, array code is undoubtedly the current best choice in erasure codes. Most of the array code operations are XOR operations, and the computational complexity is low.

Fig. 1 is the comparison of the encoding efficiency of two codes encoding 1MB-10MB files respectively. (a) is the comparison of the encoding efficiency between RA code with 5 data blocks and 2 fault tolerance and the EVENODD code with 5 data blocks. (b) is the comparison of the encoding efficiency between RA code with 5 data blocks and 3 fault tolerance and the STAR code with 5 data blocks. Each of these files is coded 10 times, and the average value after removing the maximum and minimum values. It is seen from the results shown in Fig.1 that the RA code is slightly more efficient than the EVENODD code and the Star code. And the RA code's fault tolerance is not limited, which further reflects the practicality of the RA code.

Fig. 1 Comparison of encoding efficiency

5. Conclusion
For a fault-tolerant solution applied to a distributed storage system, not only the size of the fault-tolerant capability and storage efficiency, but also the decoding time and the internal communication
mechanism between the nodes are considered. The starting point of this paper is to construct a class of MDS array codes whose fault tolerance is theoretically unrestricted, and to avoid the strong constraints that most array codes need to meet during the construction process, thus improving the practicability of array codes. The experimental results show that the RA code proposed in this paper can achieve this goal. The encoding and decoding operations are binary domain XOR operations, which have high computational efficiency, MDS properties and optimal storage efficiency. From the currently literature and reference materials, the RA code proposed in this paper is the first MDS array code that completely uses the binary domain exclusive XOR operation and theoretically has unlimited fault tolerance. Therefore, the proposed code will play a positive role in the application of array codes in large-scale distributed storage systems.

Acknowledgments
This work was financially supported by the Project of Science and Technology Department of Sichuan Province (2018CC0093 and 2018GZ0102)

References
[1] Asteris M, Asteris M, Papailiopoulos D, et al. XORing elephants: novel erasure codes for big data[J]. 2013, 6(5):325-336.
[2] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung. The Google file system[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5):29-43.
[3] Shvachko K, Kuang H, Radia S, et al. The Hadoop Distributed File System[C]// IEEE, Symposium on MASS Storage Systems and Technologies. IEEE Computer Society, 2010:1-10.
[4] Plank J S, Blaum M. Sector-Disk (SD) Erasure Codes for Mixed Failure Modes in RAID Systems[J]. Acm Transactions on Storage, 2013, 10(1):4.
[5] Blaum, M., Brady, J., Bruck, J., Menon, J. EVENODD: an optimal scheme for tolerating double disk failures in RAID architectures[P]. Computer Architecture, 1994., Proceedings the 21st Annual International Symposium on,1994.
[6] Li M, Shu J, Zheng W. GRID codes:Strip-based erasure codes with high fault tolerance for storage systems[J]. Acm Transactions on Storage, 2009, 4(4):1-22.
[7] Zheng Z, Sinha P. XBC: XOR-based buffer coding for reliable transmissions over wireless networks[C]// International Conference on Broadband Communications, Networks and Systems, 2007. Broadnets. IEEE, 2007:76-85.
[8] Plank J S, Xu L. Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Network Storage Applications[C]// IEEE International Symposium on Network Computing and Applications. IEEE Computer Society, 2006:173-180.
[9] Plank J S, Greenan K M, Miller E L. Screaming Fast Galois Field Arithmetic Using Intel SIMD Instructions[J]. 2013.
[10] Xu L, Bruck J. X-code: MDS array codes with optimal encoding[J]. Information Theory IEEE Transactions on, 1999, 45(1):272-276.
[11] Huang C, Xu L. STAR : An Efficient Coding Scheme for Correcting Triple Storage Node Failures[J]. IEEE Transactions on Computers, 2008, 57(7):889-901.
[12] Hafner J L. WEAVER codes:highly fault tolerant erasure codes for storage systems[C]// FAST ’05 Conference on File and Storage Technologies, December 13-16, 2005, San Francisco, California, Usa. DBLP, 2005:16-16.
[13] Reed I S, Solomon G. Polynomial Codes Over Certain Finite Fields[J]. Journal of the Society for Industrial & Applied Mathematics, 1960, 8(2):300-304.
[14] Plank J S. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems[M]. John Wiley & Sons, Inc. 1997.
[15] Roth R M, Lempel A. On MDS codes via Cauchy matrices[J]. IEEE Transactions on Information Theory, 1989, 35(6):1314-1319.
[16] Blomer J. An XOR-based erasure-resilient coding scheme[J]. Technical Report at Icsi, 1995.
[17] Fan Di, Xiao Feng, Tang Wei. Research on Improvement Based on Matrix Decoding Algorithm[J]. Computer applications and software, 2018, 35(08): 295-302.