Research on Fault Tolerance Enhancement of Array Erasure Codes

Dan Tang[1,a], Feng Xiao[1] and Di Fan[1]

[1]*Software Engineering Institute, Chengdu University of Information Technology,610225Chengdu, P. R, China*

Abstract. Array erasure coding technology is one of the most desirable methods to enhance the reliability of storage systems, but fault tolerance is always the bottleneck to the use of array codes. There are several improvement schemes for fault tolerance of array codes now, such as graph theory method, hybrid layout method, and so on. But improvement of the fault tolerance is pretty small, and always cost too much. In view of this, the paper presents a new method to improve fault tolerant ability of array erasure codes. The new method has a highly computing efficiency because all calculations are binary XOR operations, and it is convenient to implement because of its simple construction. Using this new method, we can model a horizontal array code with the optimal updating expenses. Although MDS array codes cannot be modeled by the new method, but storage efficiency can be improved by increasing the strip size. The most important contribution of this paper is that, this new method can construct an array erasure code which has the unconstrained fault tolerant ability in theory.

1 Introduction

At present, the data generated by all sectors of industries is growing explosively. To deal with the storage reliability problem caused by the rapid growth of the data quantity, an effective method is to build a storage system using multiple independent storage nodes together besides the simple expansion of the storage node capacity. This method can effectively increase the storage capacity, enhance the system reliability, improve the data parallel access efficiency and reduce the cost of storage system construction. In this case, we use the number of storage nodes to represent the scale of a storage system. At present, storage systems with more than 100 nodes have become more popular, some companies such as Google and Facebook even already have many storage systems with node quantity between 3000 to 4000[1]. However, under the condition that the average failure probability of a single node is certain, the increasing amount of nodes means that the number of nodes that may fail simultaneously in any time period would increase. In addition, earthquake, flood, fire, debris flow, and other natural disasters, hacker attacks, terrorist attacks, operational mistakes, intentional or unintentional destruction, and other emergencies may also cause multiple storage nodes to fail simultaneously. Therefore, it is extremely important to research the fault tolerant technology of storage systems for ensuring data security.

Replication is the most concise and intuitive fault tolerant method, which stores multiple copies of the same data to different storage nodes, and redundant data are copies of the original file. Because any copy in a node is equal to the original data, so when a node failure occurred, all data in the failure node could be recovered by any other normal node. This replication method is easy to implement and is convenient to deploy and expand dynamically. However, as the method which core idea is to enhance the reliability of the storage system, replication has great defects. The biggest problem is that it will lead to the low efficiency of storage space utilization. Using replication strategies to build a f fault tolerant storage system where f is a positive integer, it needs to copy each data to $f+1$ different storage node, in other words, the storage efficiency of the entire system is only $1/(f+1)$, and this is clearly unacceptable to a large scale storage system. In addition, the high updating expenses is another problem of replication strategy, any one of the smallest unit of data changes will involve all storage nodes which have the data duplicate. If the update operations occurred frequently, the operation efficiency of the whole system would be greatly affected.

The fault tolerant method based on erasure codes is a kind of storage reliability enhancement method which has attracted more and more attention in recent years. Compared with the fault tolerant method based on replication, in a storage system, erasure codes based methods can effectively balance fault tolerance, storage efficiency, repair bandwidth, updating expenses, and so on, so it has become an important approach to enhance the storage reliability. At present, erasure codes used in the storage reliability enhancement include RS erasure codes, LDPC erasure codes, array erasure codes, etc...

At present, the RS erasure code is the only one known MDS code which can correct any number of faults. It has achieved the Shannon bound, so the fault tolerant scheme based on RS codes has the advantages of unrestricted

[a] Corresponding author: tangdan@foxmail.com

fault tolerant capability and optimal storage efficiency. However, because of the complexity of RS codes over finite field operations, its computational cost is hard to be accepted by any large-scale practical system. In view of this, scholars have put forward some useful modified methods to improve the computing efficiency[2~3]. In recent years, scholars have made some great work on the efficiency improvement of operations over the finite field, in which the most typical is the GF-Complete proposed by professor Plank[4]. In spite of this, the operational efficiency of the RS code still cannot completely meet the needs of the storage system, especially the large-scale storage system. The LDPC(Low Density Parity Check) erasure code is a kind of erasure codes which encoding and decoding process are all based on XOR operations, so compared with RS codes, its operational efficiency is much higher. The LDPC erasure code is not a MDS code, but with the same fault tolerant capability, its code rate can be close to the MDS code rate. Although the LDPC code has many advantages, but it still faces a lot of problems when face to the storage system, including the irregularity of the code structure and the probability of successful decoding. And these features have greatly hindered the utilization of LDPC codes. And nowadays, the application of LDPC codes in the storage system is few[5].

Only simple binary XOR operations are used in the encoding and decoding process of array erasure codes (simply as array codes) , so it has a high computational efficiency. In addition, which has many other advantages such as simple realization, two-dimensional coding structure. These features make array codes suitable for being used in the storage system with multiple nodes. According to the storage efficiency, array codes can be subdivided into MDS codes and non MDS codes. Among them, 2 fault tolerance codes such as EVENODD codes[6] and X codes[7], 3 fault tolerance codes such as Star codes[8] and extended X codes[9] are all typical MDS array codes. However, the research work of MDS array codes with higher fault tolerance has not made a breakthrough. Researchers have modeled some array codes that have higher fault tolerance by giving up some storage efficiency, including Weaver code[10], Hover code[11], Grid code[12], and E code[13], and so on. The fault tolerant capability of these array codes has a certain improvement compared to the MDS array code, but it can only reach 10 or more, and such fault tolerant ability is still difficult to fully meet the application requirements.

This paper focuses on methods to improve the fault tolerant capability of array codes, and gives a method to effectively improve the fault tolerance. The new method can balance some important performance of the storage system. And it is proved that the proposed method can make the array code have the theoretically unrestricted fault tolerance. The organizational structure of this paper is as follows: In the first chapter, some general concepts in the fault tolerant area are given, and some common methods of improving the fault tolerance of array codes are presented. The second chapter will give a new method to improve the fault tolerance of array codes, as well as restrictions. In the third chapter, some important properties of the new method are analyzed. The last chapter is the summary of this article.

2 Basic concepts and related works

2.1 Basic concepts

Data, parity, element, stripe, strip, and other common terms can refer[6]-[13], while some of the symbols and concepts given below are prepared to explain the new method. It is important to note that the fault tolerance enhancement method proposed by this paper is based on the horizontal data layout.

Coding chain and slope: The data elements array of the storage system can be regarded as a two-dimensional coordinate system, the element at the first row and the first column be the origin of the coordinates, the horizontal rightward axis be the positive X axis, the vertical downward axis be the positive Y axis, and all data elements can be seen as a point in the coordinate system, so all data elements that need to calculate one parity can be considered as a chain on the coordinate system. What needs to explain specially is that, the coding chain is not first proposed in this paper, and such concepts have been mentioned in several previous research papers of array codes, for example, the X code call the coding chain as decoding chain[7]. Supposing the coordinates of all elements of a coding chain are $(x_1, y_1), (x_2, y_2) \dots (x_m, y_n)$, then these coordinates must meet the following conditions:

$$\begin{aligned} &(y_2 - y_1)\%n /(x_2 - x_1)\%n \\ &= (y_3 - y_2)\%n /(x_3 - x_2)\%n \cdots \\ &= (y_m - y_{m-1})\%n /(x_m - x_{m-1})\%n \end{aligned} \tag{1}$$

And the slope of the coding chain can be defined as $s = (y_m - y_1)\%n /(x_m - x_1)\%n$. In the process of encoding and decoding, the coding chain is used as a basic unit to participate in. Obviously, the XOR sum of all elements in a coding chain will produce a parity element(or check element). This parity element will also be involved in the encoding and decoding process as data elements in a coding chain, so it can be called as coding chain tail. When deploying a coding chain, it is just need to select a location from the parity array section in a certain sequence to store the parity element, namely the XOR sum of all data elements in a coding chain. For example, Figure 1 shows selections of corresponding parity elements when deploying a coding chain with the slope 1, while the XOR sum of all elements with the same texture is zero. In addition, if there were only one failure element in a coding chain and its tail, this failure element can be recovered obviously, so this coding chain can be called as the valid decoding chain.

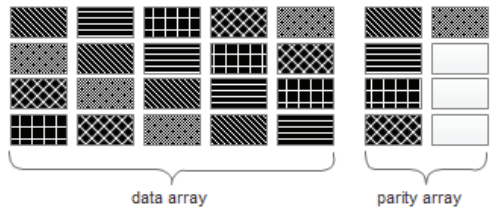

Figure 1. The coding chain of slope 1 and parity elements.

$El(i,j)$: The arithmetic expression $El(i,j)$ denotes the element of *i*-th row and *j*-th column, where *i* and *j* are all positive integers. When there are a number of elements that need to be represented, the symbol ':' can be used to show a range, such as $El(i,1:4)$ represents the element collection of *i*-th row and 1st to 4th column in the storage array, and $El(i,:)$ represents all elements of *i*-th row.

$Chain^{s}_{El(i,j)}$: The coding chain which includes the element $El(i,j)$ and has the slope *s*. The superscript and subscript of that expression can be omitted according to the actual situation. For example, $Chain_{El(:,j)}$ represents a collection of coding chains with any slopes and contains any element of the column *j*, $Chain^{s}$ refers to all coding chains with the slope *s*, while *s* can be a certain number or a collection.

D^{i}_{j}: D^{i}_{j} represents the distance from the column *i* to the column *j* in a storage array, and which definition is as below: $D^{i}_{j}=(j-i-1)\%n+1$, where *n* is the number of columns of the storage array. In the research of array codes, a column of a storage array is usually used to represent a node in the actual storage system, and node failure that means all elements lost in the corresponding column. A special note is the distance between failure columns, if one or more failure columns existed between the failure column *i* and *j*, there were no distance from column *i* to *j*, that is denoted as $D^{i}_{j}=0$.

R: Symbol *R* indicates that failure elements can be recovered with a coding chain or a set of coding chains. For instance, $R(EL(i,j))=Chain^{s}_{El(i,j)}$ indicates that the failure element $El(i,j)$ can be recovered by one or more coding chains which contain the element $El(i,j)$ itself and have the slope *s*, and can be denoted as the expression $R(EL(i,j))=Chain^{s}$ simply. Using multiple *R* expressions, the recovery process for failures can be displayed clearly.

2.2 Review of related works

Low fault tolerance is one of the most important factors that restrict the development of array codes, so how to improve the fault tolerance of array codes is an important research subject. At present, main methods to improve the fault tolerance of array codes can be summarized as follows.

1. Graph theoretical method. All data and parity elements in the same stripe are used as nodes, connections among these elements are used as edges, then an undirected graph set up. In the establishment of relationships among elements, namely adding edges among nodes should strictly control the out-degree and in-degree to meet some conditions, and based this to find approaches to improve the fault tolerance of array codes. If constraint conditions of the elements in the graph were reasonable, array codes that could tolerate 10 or more erasures can be constructed. However, this method is usually unable to construct a general fault tolerant capability model, which restricts the application range of which using this kind of array codes. In addition, using this method to improve fault tolerance will pay a high cost of storage efficiency. The weaver code is a typical representative of using graph theory to improve fault tolerance, the fault tolerant ability of weaver codes can be up to 12 or even higher. But when the weaver code is used to construct a 10 fault-tolerant storage system, the storage efficiency will be less than 20%, which is only slightly higher than the replication strategy[10].

2. Hybrid layout method. As previously mentioned, according to the layout relation of parity elements and data elements in a storage array, array codes can be divided into the vertical array code and the horizontal array code. For the vertical array code and horizontal array code, the layout relationship between the parity elements and the data elements is fixed. The hybrid layout method refers to a combination of vertical and horizontal data layout in the design of array codes to achieve the purpose of improving the fault tolerant ability. Compared with the Graph theoretical method, using the hybrid layout method to model array codes has a smaller cost of storage efficiency, and the fault tolerant ability can be proved in theory. However, this method usually has strict restrictions for stripe and stripe size, so it is not easy to extend, nor fully consider the update penalty and other factors. The grid code is a typical representative of using hybrid layout method to improve fault tolerance, and the fault tolerant ability can be up to 15 or even higher[12]. However, the array size is subject to the traditional horizontal and vertical codes used in the construction of a grid code, and a write operation of a minimum unit involves data access of all storage nodes, so it is hard to apply to large scale storage systems.

3. Random parities method. To improve fault tolerance while maintaining the balance of many other performance of the storage system, the random parity method gives up the practice of establishing the linear relationship between the data and the parity, and the stochastic method is used to establish connections between the data and the parity elements. At present, a number of array codes constructed by this method can tolerate 30 or even higher failures. However, there are more problems in the random parities method, the fault tolerant ability of array codes constructed by this kind of method has not obtained a mathematical proof, it is just gotten by the computer experiment usually, confirmed by the exhaustive method, or using a sampling method to prove the fault tolerant ability can achieve in a high probability, so these make this method difficult to apply

in various practical applications. The E code and GE code are typical representatives of using the random parity method to improve fault tolerance, and experiments prove that the fault tolerant ability of E codes or GE codes can be up to 30 or even higher in a high probability, but the fault tolerant ability is less than 10 under certain conditions, so this feature greatly limits the popularization and application of this kind of method[13].

3 Fault tolerance enhancement method based on coding chains

This chapter will propose a new method to improve the fault tolerant capability of array codes. In order to ensure the high efficiency of encoding and decoding process, all operations in the new method are XOR operations over the binary field. On the data layout of the vertical array codes, data elements and parity elements exist in the same strip, storage nodes can have a good load balance, but elements in different nodes are highly dependent on each other, and it's not easy to extend. This new method uses the typical horizontal data layout. At present, most array codes with similar technical routes organize the relationship between data elements and parity elements by using horizontal, diagonal and negative diagonal three kinds of coding chains, but the maximum fault tolerance of that strategy is not more than 3 in theory. Even more serious is that, in order to adapt to the large scale storage system, the horizontal coding chain is no longer applicable, otherwise, a minimum unit data update on one node will involve read operations of all the other data nodes. Thus, in order to improve the fault tolerance of array codes, this paper builds the correlation between data elements and parity elements by using coding chains with multiple disparate slopes.

3.1. Basic encoding and decoding method

As mentioned before, in order to reduce the correlation strength among data and parity elements, the article uses a typical horizontal data layout. Assuming that the fault tolerant ability of storage nodes is *f*, the number of rows of the storage array is *m*, the number of columns of the data array portion is *n*, and the number of columns of the parity array portion is *t*, then the size of the entire storage array can be expressed as $m\times(n+t)$, obviously, *f*, *m*, *t* are all positive integers, and $m\geq 2$. If there is no special description, these symbols *m*, *f*, *n*, *t* in later sections have the same meanings as described here.

Deploying *f* coding chains in the $m\times n$ data array portion, and there are two situations will be discussed respectively. When *f* is odd, deploying coding chains with slopes $\pm 1,\pm 2\cdots\pm\lfloor f/2\rfloor,\lceil f/2\rceil$ in the data array portion, and when *f* is even, deploying coding chains with slopes $\pm 1,\pm 2\cdots\pm\lceil f/2\rceil$ in the data array portion. The XOR sum of all elements in a coding chain is placed in the parity array as the column-major sequence. As noted above, the size of the data array portion is $m\times n$, all elements in the data array can be denoted as $El(:,1:n)$, namely, the symbolic representation $El(i,j)$ can denote every element of the data array portion when $1\leq i\leq m, 1\leq j\leq n$. Similarly, all elements of the $m\times n$ parity array portion can be denoted as $El(:,n+1:n+t)$, or as $El(i,j)$ when $1\leq i\leq m, n+1\leq j\leq n+t$. Each parity element is derived from XOR sum of *m* data elements, now still assuming that *f* is the maximum fault tolerant ability, then every parity element can be expressed and calculated by formula (2).

$$El_{check}=\sum_{row=1}^{m}El(row,((row-1)\cdot sign\cdot slope+1+beta)\%n) \tag{2}$$

In the formula (2), the symbol El_{check} represents any parity element, the symbol *sign* indicates that the slope of the current coding chain is positive or negative, the absolute value of the current coding chain is represented by the symbol *slope*, and the symbol *beta* represents the increment value between the column number of the current data element in a coding chain and the column number of the previous element. And all these symbols are defined as follows.

$$slope=\lceil i/2\rceil,\ beta=j-1, \tag{3}$$

$$sign=\begin{cases}-1 & ,i\%2=0\\ 1 & i\%2=1\end{cases}, \tag{4}$$

In above definitions, symbols *i* and *j* indicate that the *j*-th coding chain of maximum *i*-fault-tolerance is deploying, where $1\leq i\leq f$, $1\leq j\leq n$. Therefore, all parity elements can be derived through the formula (2).

The above shows that, the fault tolerant capability of the storage system increased by 1, you need to deploy *n* more coding chains, and to increase *n* more parity elements correspondingly, so the number of parity nodes is $t=\lceil f\cdot n/m\rceil$.

Array erasure codes with unrestricted fault tolerance can be constructed by the above method according to the application environment. Then we will explain how to recover failure elements on failure nodes. This paper only considers the storage node failure, namely all elements lost in a failure nodes, not consider partial data loss in a node. Node failures can be divided into three categories according to the type of failure nodes, including all failures occurred in the parity array portion, all failures occurred in the data array portion, the data array and the parity array all have failures. All failures occurred in the data array portion is most difficult to recover, so assuming all failures occur in the data array portion. The core idea of this method is to find valid decoding chains to recover the loss elements on failure nodes, which is similar to the Zig-Zag decoding method[8] but a more concise valid decoding chains searching algorithm.

Supposing there are *f* failure nodes in the data array portion, and each node in a storage system can correspond to one column in a storage array, so label all *f* failure columns as $1,2\cdots f$ sequentially. The distance between column *f* and column 1 can be denoted as D_1^f, then supposing D_1^f is maximum among all distance values of adjacent failure columns. The core idea of this

algorithm describes as follows: searching a coding chain and its corresponding tail which have only one lost element, namely a valid decoding chain, then XOR sum of all the other elements of this coding chain and its tail is just the lost element value. As stated earlier, all deployed coding chains have the slope symmetry property, and using this property, the search range of valid decoding chains can be reduced greatly. For example, if $Chain^{s}_{El(i,j)}$ was a valid decoding chain, then $Chain^{-s}_{El(m+1-i,j)}$, $Chain^{-s}_{El(i,f+1-j)}$ and $Chain^{s}_{El(m+1-i,f+1-j)}$ are all valid decoding chains, where $1 \le i \le \lfloor f/2 \rfloor$, $1 \le j \le \lfloor m/2 \rfloor$. In other words, if the loss element $El(i,j)$ can be recovered by the coding chain $Chain^{s}_{El(i,j)}$, the loss elements $El(i,f+1-j)$, $El(m+1-i,j)$ and $El(m+1-i,f+1-j)$ can be recovered by coding chains $Chain^{-s}_{El(m+1-i,j)}$, $Chain^{-s}_{El(i,f+1-j)}$ and $Chain^{s}_{El(m+1-i,f+1-j)}$ respectively.

Example 1. A storage array, supposing the size of the data array portion is 4×7, and maximum fault tolerant ability requirement is 2, then 2 coding chains need to be deployed obviously. A total of 14 parity elements will be generated, so 4 parity columns are also needed. The deployment of two coding chains is showed in Figure 2, all elements with the same texture constitute the coding chain and its corresponding tail, and the XOR sum of those elements is zero.

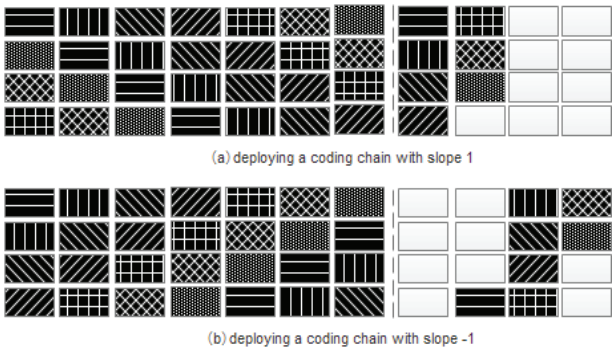

Figure 2. The deployment of coding chains to tolerate 2 failures.

In example 1 above, the second and fourth nodes are assumed to be failure, namely all elements in the second and fourth column lost. According to the data recovery method described in this chapter, all loss element can be recovered completely, and details of the recovery process are shown in Figure 5. An important point to be explained is that, the parity array portion is leaved out because all failures are in the data array portion. Furthermore, in subfigures of Figure 3, all elements with the same flag are coding chains used to recover loss elements.

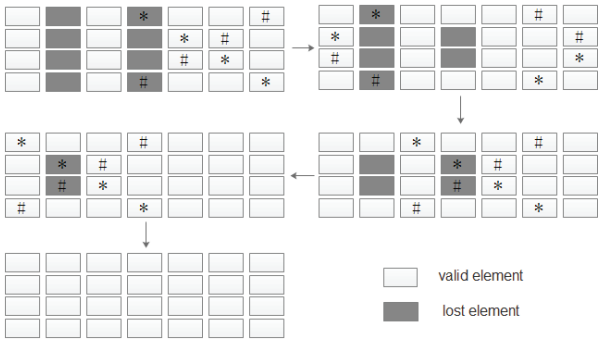

Figure 3. The recovery process of lost data.

In example 1, the size of the data array portion is 4×7, all lost data in column 2 and 4 are recovered successfully. Supposing the size of the data array portion is 4×5, the second and fourth nodes are also assumed to be failure, and two coding chains are deployed as before, shown in Figure 4, but now, any coding chain and its tail with only one lost element can not be found, all lost data can not be recovered. And more, another conclusion can be drawn from the experiment, deploying coding chains like this chapter, if the size of data array portion is 4×5, all lost data cannot be recovered if two or more nodes are failure.

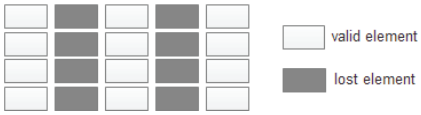

Figure 4. All lost data can not be recovered..

3.2 Constraint conditions of the data array size and fault tolerance

Place the figure as close as possible after the point where it is first referenced in the text. If there is a large number of figures and tables it might be necessary to place some before their text citation. If a figure or table is too large to fit into one column, it can be centred across both columns at the top or the bottom of the page.

From descriptions of loss data recovery of previous section, an intuitive conclusion can be drawn, just deploying coding chains cannot ensure the fault tolerant capability of array codes, the size of the storage array is also a constraint factor of fault tolerance. In other words, with the same number of rows, reducing the number of columns of the data array portion will decrease fault tolerant ability of array codes. Extending example 1 again, another conclusion can be obtained through exhaustive testing, in the case of 4 rows, 2 node failures can be recovered as long as the number of columns is equal or over 7. So using the method proposed in this paper to improve the fault tolerance of array codes, a prerequisite is that the size of the data array and the number of fault tolerance need to satisfy a certain relationship.

Symbols *m* and *n* still represent the number of rows of the storage array and the number of columns of the data array portion, for a *f*-fault-tolerant storage system, assuming that *f*,*m*,*n* satisfy the following constraints: $n \ge m \cdot f - f + 1$, where *m*, *n* and *f* are positive integers, and $m \ge 2$. Now we discuss fault tolerant ability of array codes under that condition.

As mentioned before, the symbol t is denoted as the column number of parity array portion, then $t = \lceil f \cdot n / m \rceil$, thus $n > m$ and $t > m$. For horizontal array codes, the occurrence of failures can be divided to three types as follows: all failures occurred in the parity array portion, both data array portion and parity array portion have failures, and all failures occurred in the data array portion. The following will discuss these three cases respectively.

All f failures occur in the parity array portion is the simplest case of data recovery, all lost elements can be recovered by simply recalculating the parity elements. The case that both data array portion and parity array portion have failures is also easy to deal with. From the deployment of coding chains, in an array code with f fault tolerant capability, each data element belongs to f coding chains, the XOR sum of all elements is corresponding to only one parity element, so one data element is associated with f parity elements. The row-major sequence is used in coding chains deployment, and the column-major sequence is used when arranging the corresponding parities, and by the aforementioned restriction $n \geq m \cdot f - f + 1$, n is strictly larger than m, so two parity elements that associated with the same data element be not in the same column affirmatively. Supposing there were i failures in data array portion, and $f - i$ failures in parity array portion, in other words, $m \cdot (f - i)$ data elements and $m \cdot i$ parity elements are failure. Then for any data element, there are $f - 1$ correlation parity elements lost at most. Thus all failures occurred in the data array portion can be recovered, and then other failures occurred in the parity array portion can be recovered. The following discussion is the most difficult case of data recovery, this is the case that all failures occurred in the data array portion.

Using the mathematical induction to analysis, for convenience, assuming all discussions of this article below will follow the following conditions: all f failure columns are sequential denoted as $E_1, E_2 \cdots, E_f$ respectively, denote the distance between failure column as E_i and its right failure column as d_i, namely $D^{E_i}_{E_{(i-1)\%f+1}} = d_i$. Supposing $\max(d_1, d_2 \cdots, d_f) = d_f$, $i = 1,2 \cdots, f$. because of $n \geq m \cdot f - f + 1$, the inequality $d_f \geq m$ is established according to the pigeonhole principle.

Denote the only failure column as E_1 when $f = 1$, because of $n \geq m \cdot f - f + 1$, $n \geq m$ can be obtained. By the deployment of coding chains, a coding chain with slope 1 is deployed in the data array portion, so every failure element is passed through by a coding chain with slope 1. There is only one failure element on every coding chain contained failures, so each failure element can be recovered, namely the expression $R(El(:, E_1)) = Chain^1$ is established. Thus a conclusion can be drawn, all failures can be recovered when $f = 1$.

Denote two failure columns as E_1 and E_2 when $f = 2$, because of the precondition $n \geq m \cdot f - f + 1$, $n \geq 2m - 1$ can be obtained. All distances among columns are all positive integers, so expressions $R(El(1, E_1)) = Chain^{-1}$, $R(El(1, E_2)) = Chain^1$ hold. Then all failures in the first row of the storage array can be recovered, namely the expression $R(El(1,:)) = Chain^{\pm 1}$ holds. Repeat steps above, all failures can be recovered. Thus a conclusion can be drawn, all failures can be recovered when $f = 2$.

Supposing all failures can be recovered when $f = 2k$ .The following is a discussion about $f = 2k + 1$ and $f = 2k + 2$. Denote all $2k + 1$ failure columns as $E_1, E_2 \cdots E_{2k}, E_{2k+1}$ when $f = 2k + 1$. The distance $d_{2k+1} \geq m$ is the maximum among all distances of failure columns from the prerequisite. When $d_1 \geq \lceil m / 2 \rceil$ holds, the equality $R(El(:, E_1)) = Chain^{\pm 1}$ has been obtained. For the same reason, when $d_{2k} \geq \lceil m / 2 \rceil$ holds, the equality $R(El(:, E_{2k+1})) = Chain^{\pm 1}$ has been obtained. From two cases above, all failures can be recovered when $d_1 \geq \lceil m / 2 \rceil$ or $d_{2k} \geq \lceil m / 2 \rceil$. While inequalities $d_1 < \lceil m / 2 \rceil$ and $d_{2k} < \lceil m / 2 \rceil$ holds, these equations can be obtained obviously, $R(El(1, E_1 : E_k)) = Chain^{-1 \geq s \geq -k}$, $R(El(1, E_{k+2} : E_{2k})) = Chain^{1 \leq s \leq k}$, $R(El(1, E_{k+1})) = Chain^{k+1}$. So all failures in the first row of the storage array can be recovered, repeat the same steps above, all failures can be recovered when $f = 2k + 1$. Denote all $2k + 2$ failure columns as $E_1, E_2 \cdots E_{2k+1}, E_{2k+2}$ when $f = 2k + 2$. The distance $d_{2k+1} \geq m$ is the maximum among all distances of failure columns because of the prerequisite. When $d_1 \geq \lceil m / 2 \rceil$ holds, the equality $R(El(:, E_1)) = Chain^{\pm 1}$ has been obtained. For the same reason, when $d_{2k+1} \geq \lceil m / 2 \rceil$ holds, the equality $R(El(:, E_{2k+1})) = Chain^{\pm 1}$ has been obtained. From two cases above, all failures can be recovered when $d_1 \geq \lceil m / 2 \rceil$ or $d_{2k+1} \geq \lceil m / 2 \rceil$. While inequalities $d_1 < \lceil m / 2 \rceil$ and $d_{2k+1} < \lceil m / 2 \rceil$ holds, these equations can be obtained obviously, $R(El(1, E_1 : E_k)) = Chain^{-1 \geq s \geq -k}$, $R(El(1, E_{k+3} : E_{2k})) = Chain^{1 \leq s \leq k}$, $R(El(1, E_{k+2})) = Chain^{k+1}$, $R(El(1, E_{k+1})) = Chain^{-k-1}$. So all failures in the first row of the storage array can be recovered, repeat the same steps above, all failures can be recovered when $f = 2k + 2$.

Thus we can conclude that, there are f failures in a $m \times n$ storage array, if f, m, n can satisfy the restrictive condition $n \geq m \cdot f - f + 1$, f failures can be recovered under the array coding scheme proposed in this chapter, where f, m, n are positive integers and $m \geq 2$.

3.3 Colour illustrations

You are free to use colour illustrations for the online version of the proceedings but any print version will be printed in black and white unless special arrangements have been made with the conference organiser. Please check with the conference organiser whether or not this is the case. If the print version will be black and white only, you should check your figure captions carefully and remove any reference to colour in the illustration and text. In addition, some colour figures will degrade or suffer loss of information when converted to black and white, and this should be taken into account when preparing them.

4 Performance analysis

Equations should be centred and should be numbered with the number on the right-hand side.

$$T_s(l,t) = T_g(l,t) \quad (1)$$

$$T_s(l,t) = T_g(l,t)\, T_b(x \quad , t) = 0 \quad (2)$$

Use italics for variables (*u*) and bold (u) for vectors. The order for brackets should be {[()]}, except where brackets have special significance.

This paper proposes a new improvement scheme for a class of array code tolerance ability, and the specific implementation steps and some restrictions are described above. However, for the entire storage system, it is undesirable to pursue only high fault tolerant capacity and ignore other performance. The previous article has demonstrated that this method can effectively improve the array code fault tolerance, and a chapter of the restrictions of ranks of the array codes on using the new method after are described. This chapter will describe the major performance costs overhead resulting from the use of new fault tolerance enhancement method.

4.1 Operational workload

About fault tolerance, compared with replication method, a major disadvantage of encoding method is that operations is necessary when storing data, just like RS erasure codes with so many advantages can not be applied to large-scale storage systems, one of the most direct reason is the high computational complexity required for encoding and data reconstruction. For storage systems, especially a large-scale storage system, the practicability of erasure codes largely depends on the operational workload. And the calculation load of encoding and decoding is an important indicator of erasure codes. For most non-MDS array codes, their structures are not normative usually, so it is difficult to use a certain formula to express calculations load of encoding or data recovery. Thus the number of binary XOR operations which are needed to generate a parity element is used to measure the complexity of encoding, and the number of binary XOR operations which are needed to recover a lost element is used to measure the complexity of data recovery.

From the previous chapter, based on the code-string array fault tolerance enhancement method, the amount of XOR operations required to generate a single parity element, and the amount of XOR operations required to recover a single lost element are all $m-1$, where m is the number of rows of the storage array, namely the strip size. With the most important prerequisite $n \geq m \cdot f - f + 1$ described in the previous chapter, when m is a given value, the more fault tolerance number means that the more parity elements need to be generated. In the case of m is 8, Figure 5 shows the amount of XOR operations required to store a 100M file when f is from 2 to 50. It is not difficult to see from this figure, the required binary XOR operations to store a 100M file is less than 13×10^8 even when the fault tolerant ability is as high as 50, and all of these operations can be completed by a most common computing node soon.

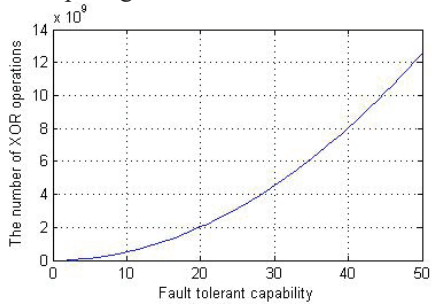

Figure 5. Relationship between fault tolerance and calculations.

4.2 Update penalty

The update penalty of an erasure coding scheme is the number of parity nodes whose contents must be changed when a minimal change is made in the contents of a given information node. Because parallelism is the reason we want to use storage arrays, the number of storage nodes of access required to effect a small data update must be minimized. By information theory, at least f parity nodes are involved when a minimum data update of a f-fault-tolerant array code with horizontal data layout, namely the minimum update penalty is f.

From steps of the new method, in using this method to model a f-fault-tolerant array code, any data element is belong to just f coding chains, namely every data element is interrelated to f parity elements, and these f parity elements are in different nodes certainly from proof of the previous chapter, it would result access operations in just f different parity nodes when a minimum data update, achieve The theoretical minimum update penalty.

4.3 Storage efficiency

A biggest drawback of the fault tolerant method based on replication strategy is the extremely low storage efficiency, the waste of storage space will be unacceptable especially when the storage scale becomes large. While the erasure codes based fault tolerant method can improve storage efficiency under the condition of the same fault tolerant ability. Storage efficiency is an important performance indicator of fault

tolerant codes. Under the equivalent fault tolerance, if the codes based method had an approximate or even lower storage efficiency, the coding scheme is fail, at least from the perspective of that computational cost versus storage efficiency.

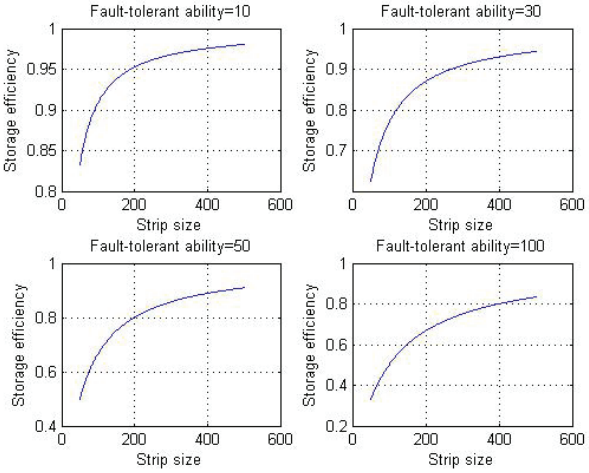

Figure 6. Correlation between the strip size and the storage efficiency.

MDS array codes cannot be modeled by the new method proposed in this paper, so an optimal storage efficiency can not be obtained. And even worse, increased fault tolerance can also lead to a decrease of storage efficiency. How to improve storage efficiency while ensuring fault tolerance is also a key point in the process of constructing array codes based on the new method. Under the new method, there are $f \cdot n$ data elements and $m \cdot n$ parity elements, in addition, the inequation $n \geq m \cdot f - f + 1$ must satisfy, so the storage efficiency can be improved by increasing m. Figure 6 shows the correlation between strip size and the storage efficiency.

5 conclusions

Low fault tolerance is one of the most important factors that restrict practical applications of array codes. However, considering the failures recovery time, internal communication mechanism between nodes, the operational efficiency of the entire storage system, and many other factors, for a fault tolerant scheme of a storage system, it can not be said that a higher fault tolerance array code has a higher practical value certainly. The starting point of this paper is to find a method to improve the fault tolerance of array codes, in other words, following this method, an array code which fault tolerance number can be preestablished. This method can also avoid strong constraint conditions that are needed for most array codes, and limit performance costs paid for the fault tolerance in an acceptable range. The work in this paper achieve these targets, the method proposed in this paper is also the first one that model array codes can tolerate arbitrary number of failures from the current known literature and reference materials. Therefore, it is reasonable to assume that our work will play a positive role in improving the range of application and practical value of array codes.

References

1. Sathiamoorthy M, Asteris M, Papailiopoulos D, et al. Xoring elephants: Novel erasure codes for big data. In: Proceedings of the VLDB Endowment. VLDB Endowment, 2013, 325–336.
2. Zheng Z, Sinha P. XBC: XOR-based buffer coding for reliable transmissions over wireless networks. In: Proceedings of Broadband Communications, Networks and Systems, Raleigh, 2007. 76–85.
3. Plank J S, Xu L. Optimizing Cauchy Reed-Solomon codes for fault-tolerant network storage applications. In: Proceedings of Network Computing and Applications, Massachusetts, 2006. 173–180.
4. Plank J S, Greenan K M, Miller L. Screaming fast Galois field arithmetic using Intel SIMD instructions. In: Proceedings of 11th USENIX Conference on File and Storage Technologies (FAST 13), San Jose, 2013. 298–306.
5. Harihara G, Janakiram B, Chandra M G, et al. SpreadStore: a LDPC erasure code scheme for distributed storage system. In: Proceedings of the 2010 International Conference on Data Storage and Data Engineering, Bangalore, 2010. 87–93.
6. Blaum M, Brady J, Bruck J, et al. EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures. IEEE Trans Comput, 1995, 45: 192–202.
7. Xu L, Bruck J. X-code: MDS array codes with optimal encoding. IEEE Trans Inform Theory, 1999, 45: 272–276.
8. Huang C, Xu L. STAR: an efficient coding scheme for correcting triple storage node failures. IEEE Trans Comput, 2008, 57: 889–901.
9. Meng Q C. Research of erasure codes applied in distributed storage system. Dissertation for Ph.D. Degree. Beijing: Graduate University of the Chinese Academy of Sciences, 2007.
10. Hafner J L. Weaver codes: highly fault tolerant erasure codes for storage systems. In: Proceedings of Usenix Conference on File and Storage Technologies, San Francisco, 2005. 16–16.
11. Hafner J L. Hover erasure codes for disk arrays. In: Proceedings of International Conference on Dependable Systems and Networks, Philadelphia, 2006. 217–226.
12. Li M Q, Shu J W, Zheng W M. Grid codes: strip-based erasure codes with high fault tolerance for storage systems. Acm Trans Stor, 2009, 4: 15–22.
13. Chen Z. A class of array erasure codes and their applications. Dissertation for Ph.D. Degree. Beijing: Graduate University of the Chinese Academy of Sciences, 2009.