Rotary-code: Efficient MDS Array Codes for RAID-6 Disk Arrays

YULIN WANG, GUANGJUN LI
School of Computer Science and Engineering
University of Electronic Science and Technology of China
No.4, Section 2, North Jianshe Road, Chengdu
P.R.CHINA
wyl@uestc.edu.cn, gjli@uestc.edu.cn, http://www.uestc.edu.cn

Abstract: - Low encoding/decoding complexity is essential for practical RAID-6 storage systems. In this paper, we describe a new coding scheme, which we call Rotary-code, for RAID-6 disk arrays. We construct Rotary-code based on a bit matrix-vector product similar to the Reed-Solomon coding, and provide the geometry encoding method and detailed non-recursive decoding algorithms. The capability of two-disk fault-tolerance and the property of Maximum Distance Separable (MDS) are proved in Rotary-code. The key novelty in Rotary-code is that the Rotary-code has optimal encoding complexity and optimal decoding complexity comparing with existing RAID-6 codes.

Key-Words: - Array code; MDS; RAID; Two-disk fault-tolerance; Efficient decoding; Decoding complexity

1 Introduction

Disk arrays [11], such as redundant arrays of inexpensive disks (RAID), have been widely used in many companies, universities, and government organizations for a decade. Most of the existing RAID architectures, e.g., RAID-3 and RAID-5, use a simple parity scheme that can recover one disk failure, but now, disk and network storage systems have grown to the point where the fault-tolerance of RAID-5 is no longer enough. RAID-6 is a specification for storage systems composed of multiple storage devices to tolerate the failure of any two devices. In recent years, RAID-6 has become important when a failure of one disk drive occurs in tandem with the latent failure of a block on a second drive [4]. On a standard RAID-5 system, this combination of failures leads to permanent data loss. Hence, storage system designers have started turning to RAID-6.

Numerous erasure coding techniques have been developed that can implement RAID-6; however, each has limitations. The well known Reed-Solomon code [9] can tolerate more than one disk failure. However, the encoding and decoding of Reed-Solomon code involve operations over finite fields and are thus very slow. It would be desirable to have binary linear codes that only involve exclusive-OR (XOR) operations. For tolerating two disk failures, many good codes have been developed (e.g. [1,2,4,6-10,13,14]). Array codes are a class of binary linear codes, where information and parity bits are placed in a two-dimensional (or multidimensional) array instead of a one-dimensional vector. The information and parity bits are defined over an Abelian group $G(q)$ with an addition operation. Usually, $q = 2$. The bits are just binary bits and addition is an XOR operation [3]. The best results are EVENODD codes [1], [3], X-codes [6], and B-codes [7]. The X-Code [6] is an extremely elegant erasure code for two-disk systems that encodes, decodes and updates optimally. However, it is a vertical code that requires each device to hold two coding words for every k data words. It does not fit the RAID-6 specification of having coding devices P and Q, where P is a simple parity device. The B-code [7] is also a vertical code and does not fit the RAID-6 specification.

In this paper, we develop a new class of binary MDS array codes called Rotary-code, which can be efficiently used for RAID-6. The codes are similar to the Reed-Solomon codes and the EVENODD code. The binary MDS array codes are a class of binary linear codes, where information bits form an $m \times n$ array and parity bits form an $m \times 2$ array. In applications of these new codes in RAID, n denotes the number of information disks on which information "data" will be stored, m indicates the number of "data", which can be bytes, computer words, or disk sectors, and are stored on a disk, $(m+1)$ is a very large prime. The code rate is $n/(n+2)$, i.e., it achieves the capacity of erasure channel. Rotary-code is a low-density parity-check code, but has a sparser parity-check matrix than the EVENODD code. This property leads to faster encoding and decoding procedures for the proposed code.

This paper is organized as follows: In Section 2, we introduce the RAID-6 specification and current

2-erasure code schemes. In Section 3, we construct the Rotary-code based on a bit matrix-vector product similar to the Reed-Solomon coding and provide detailed decoding algorithms. We compare the encoding/decoding complexity with existing code scheme in Section 4. The MDS property of Rotary-code is proved in Section 5. We further share our implementation and performance tests of Rotary-code in Section 6 and conclude in Section 7.

2 RAID-6 Specification And 2-erasure Codes

RAID-6 is a specification for storage systems with $m+2$ nodes to tolerate the failure of any two nodes, where k is the amount of data disk nodes. Logically, a typical RAID-6 system appears as depicted in Figure 1. There are $m+2$ storage nodes, each of which holds B bytes, partitioned into m data nodes, $D_0, \ldots D_{m-1}$, and two coding nodes P and Q. The entire system can store mB bytes of data, which are stored in the data nodes. The remaining $2B$ bytes of the system reside in nodes P and Q and are calculated from the data bytes. The calculations are made so that if any two of the $m+2$ nodes fail, the data may be recovered from the surviving nodes.

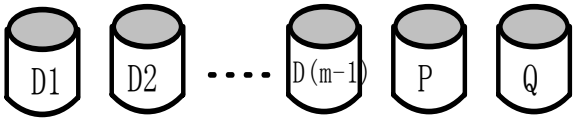


Fig. 1. Logical overview of a RAID-6 system.

Actual implementations optimize this logical configuration by setting B to be smaller than each disk's capacity, and then rotating the identity of the data and coding devices every B bytes. This helps remove hot spots in the system in a manner similar to RAID-5 systems. A pictorial example of this is in Fig. 2. For simplicity, in the remainder of this paper we assume that each storage node contains exactly B bytes as in Fig. 1 since the extrapolation to systems as in Fig. 2 is straightforward.

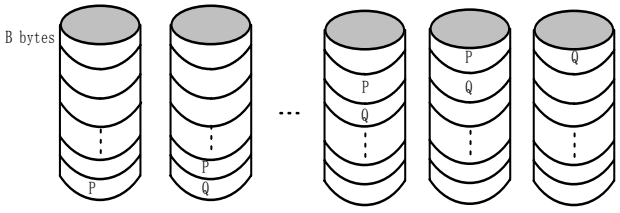


Fig. 2. In actual implementations, the identities of the data and coding nodes rotate every B bytes. This helps to alleviate hot spots on the various drives.

The RAID-6 specification calls for two parity drives, P and Q. The contents of the P drive are calculated as the parity of the data drives, just as in RAID-5. The contents of the Q drive are defined by the particular code, but must be a Maximum Distance Separable (MDS) code. This means that any combination of two-disk failures may be tolerated without data loss. In this way, RAID-6 systems extrapolate naturally from RAID-5 systems by simply adding a Q drive. It also means that the sole challenge in designing a RAID-6 coding methodology lies in the definition of the Q drive. There are several criteria to evaluate an erasure coding technique for a RAID-6 system: encode/decode complexity, update complexity and storage efficiency.

There are a variety of codes that can tolerate two-disk failures (e.g. [1,2,4,6,7,8,9,10,13,14]). We divide the set of the known 2-erasure codes into different categories and give (non-exhaustive) examples in each category. In a category by themselves are the Reed-Solomon codes [9], which are MDS. This means optimal storage efficiency and optimal update penalty. But the computational complexity is a serious problem because Galois Field computation is used though optimized algorithms have been developed [10].Second are non-MDS codes that are XOR-based, such as WEAVER codes [13] and HoVer codes [14]. These have perfect computational complexity, but bad storage efficiency is their inherent drawback. Their property of non-MDS renders them inapplicable to the RAID-6 specification.

Finally, in the last category are XOR-based MDS codes. These come in two types: vertical codes such as the X-code [6], B-code [7] and horizontal codes such as EVENODD [1,3], and Row-Diagonal Parity codes [4]. In a storage system based on horizontal codes, some disks contain nothing but data symbols, and the others contain only parity symbols. The opposite is vertical codes in which the parity symbols and the data symbols are stored together. The X-Code [6] is an extremely elegant erasure code for two-disk systems that encodes, decodes and updates optimally. However, it is a vertical code that requires each device to hold two coding words for every k data words. Vertical codes do not fit the RAID-6 specification of having coding devices P and Q, where P is a simple parity device.

EVENODD [1] is the first MDS array code, perhaps also the most important one - many subsequent array codes are similar to it and its generalization [3], such as RDP [4], STAR-code [5], etc. It is a horizontal code and parity independent (none of the parity symbols participate in other parity groups). EVENODD organizes data as

symbols in an $(m-1)\times(m+2)$ array, referred to as a segment, where the first *m* columns correspond to data disks and columns *m* and $m+1$ are check disks. The first check disk is a horizontal parity disk, and the second is a skew diagonal parity disk. D_i^* denotes a data symbol that participates in P_i and all Q_s. Namely, the sum *S* (over *GF*[2]) of all of these kind of symbols is added into every diagonal parity symbol. Thus the computational performance and the update complexity of EVENODD are non-optimal. RDP Coding is very similar to EVENODD coding, but improves upon it in several ways [4]. RDP calculates the bits of the *Q* device from both the data and parity bits, and in so doing achieves better performance. One important advantage of EVENODD and RDP is that they meet the RAID-6 specification. Moreover their coding schemes are simple and are easy to implement.

The proposed Rotary-codes are horizontal codes as well as EVENODD and share many good properties, such as optimal storage efficiency and simple encoding and decoding schemes. We describe the codes and analyze their performance below.

3 Rotary-code Description

Rotary-code encoding and decoding are based on a bit matrix-vector product very similar to those used in Reed-Solomon coding [9,10]. This product precisely defines how encoding and decoding are performed. More efficient decoding algorithms of Rotary-code are presented based on the observation of geometry property. We first define Rotary-code and then describe decoding, and prove the correctness. We discuss their encoding /decoding performance and compare Rotary-code to the other RAID-6 codes in Section 4.

3.1 Rotary-code definition

Before giving the definition of the proposed code, we first give the notation of some matrices. For a matrix $M=(m_{i,j})_{l\times l}$, we always assume that $0\le i,j\le l\text{-}1$, i.e., the order of rows (columns) is from 0 to $l-1$. Let $p=m+1$ be positive integer (not necessarily a prime). Let I_m be an $m\times m$ identity matrix and O_m be an $m\times m$ zero matrix. Now, we define the elemental right-cyclic matrix E_p as

$$E_p = E_{m+1} = \begin{bmatrix} \overrightarrow{0}^T & I_m \\ 1 & \overrightarrow{0} \end{bmatrix} \quad (1)$$

where $\overrightarrow{0}$ is a $1\times m$ vector of 0s and $\overrightarrow{0}^T$ is an $m\times 1$ vector of 0s. It can be easily checked that $\{I_p, E_p, E_p^2, ..., E_p^m\}$ form a group with matrix multiplication over *GF*(2) and $E_p^p = I_p$.

In the following, if no confusion arises, *I* and *E* are used in place of I_m and E_m respectively. We also define

$\langle a\rangle = a \mod p$

Thus, $0\le\langle a\rangle\le p-1$.

From (1), let $E^\mu=(e_{i,j})_{p\times p}$, we have

$$e_{i,j}=\begin{cases}1 & \text{for } j=\langle i+\mu\rangle\\ 0 & \text{otherwise}\end{cases} \quad (2)$$

Clearly, these matrices form an Abelian group with the traditional multiplication over *GF*(2).The unity element is *I*, i.e., identity matrix. We have

$E^i\times E^j = E^j\times E^i = E^{(i+j)}$ *and* $E^0=I$

It can be easily checked that $(I+E)$ has rank *m*. For any $1\le a\le m$, there is $1\le b\le m$ such that $\langle ab\rangle=1$. Thus, we have

$$(I+E^a)(I+\sum_{j=1}^{b-1}E^{aj}) = I+E^{ab} = I+E$$

Thus, the rank of $(I+E^a)$ is at least *m*. On the other hand, each column and each row has exactly two 1s. Therefore, the rank of $(I+E^a)$ is *m*, i.e., it is a singular $p\times p$ matrix.

Lemma 1. (E^a+E^b) has rank *m* for $a\ne b$.

Proof. There are two cases: 1) one of *a* and *b* is zero. (E^a+E^b) is transformed to $(I+E^\mu)$, the rank of which is *m*. So (E^a+E^b) has rank *m*.

2) Neither of *a* and *b* is zero. From $a\ne b$ and the definition of *E*, we know none of all $e_{i,j}$ is 1 simultaneously in E^a and E^b. Thus, the rank of (E^a+E^b) is greater than or equal to the rank of E^a over *GF*(2). It can be easily checked that E^a has rank *m*. The rank of (E^a+E^b) is at least *m*. On the other hand, each column and each row has exactly two 1s. Therefore, the rank of (E^a+E^b) is *m*.

The proof is completed. □

Now, we give the details of the proposed construction. We first define the following binary matrix:

$$P=\left[\begin{array}{c|c|c|c|c|c} I & I & I & \cdots & I & I \\ \hline I & I+E & I+E^2 & \cdots & I+E^{m-2} & I+E^{m-1} \end{array}\right] \quad (3)$$

This is a $2m \times m^2$ binary matrix. It can be regarded as a $2 \times m$ block matrix, where each block-column contains *m* columns, and each block-row contains *m* rows.

Example 1. Let $m = 4$, i.e. $p = 5$, and, we have

$$\mathrm{I}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} \quad \mathrm{E}=\begin{bmatrix}0&1&0&0\\0&0&1&0\\0&0&0&1\\1&0&0&0\end{bmatrix}$$

$$\mathrm{E}^2=\begin{bmatrix}0&0&1&0\\0&0&0&1\\1&0&0&0\\0&1&0&0\end{bmatrix} \quad \mathrm{E}^3=\begin{bmatrix}0&0&0&1\\1&0&0&0\\0&1&0&0\\0&0&1&0\end{bmatrix}$$

$$\mathrm{I}+\mathrm{E}=\begin{bmatrix}1&1&0&0\\0&1&1&0\\0&0&1&1\\1&0&0&1\end{bmatrix} \quad \mathrm{I}+\mathrm{E}^2=\begin{bmatrix}1&0&1&0\\0&1&0&1\\1&0&1&0\\0&1&0&1\end{bmatrix}$$

$$\mathrm{I}+\mathrm{E}^3=\begin{bmatrix}1&0&0&1\\1&1&0&0\\0&1&1&0\\0&0&1&1\end{bmatrix}$$

$$\mathrm{P}=\left[\begin{array}{cccc|cccc|cccc|cccc} 1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\ 0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0\\ 0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0\\ 0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\ \hline 1&0&0&0&1&1&0&0&1&0&1&0&1&0&0&1\\ 0&1&0&0&0&1&1&0&0&1&0&1&1&1&0&0\\ 0&0&1&0&0&0&1&1&1&0&1&0&0&1&1&0\\ 0&0&0&1&1&0&0&1&0&1&0&1&0&0&1&1 \end{array}\right]$$

We have the following theorem:

Theorem 1. Any 2 block-columns form a full rank submatrix, i.e., the columns of any 2 block-columns are linearly independent.

Proof. Let us consider the submatrix $\overline{P}$ consisting of two block columns x_1, x_2:

$$\overline{P}=\begin{bmatrix} I & I \\ I+E^{x_1} & I+E^{x_2} \end{bmatrix}$$

Transforming $\overline{P}$ according to $r_2+(I+E^{x_1})r_1$, $\overline{P}$ is reduced to the following matrix:

$$\begin{bmatrix} I & I \\ O_m & E^{x_1}+E^{x_2} \end{bmatrix}$$

From Lemma 1, $E^{x_1}+E^{x_2}$ has rank *m* and the submatrices in the diagonal block columns are full rank, i.e., the ranks are all *m*. Thus, the reduced matrix has rank *2m*, i.e., $\overline{P}$ is a full rank matrix. □

Now, we are going to introduce the Rotary-code definition. Let $(G,\oplus)$ be an Abelian group and let *0* be the identity element. Let $b \in \{0,1\}$, $g \in G$. We define

$$b \times g = g \times b = \begin{cases} 0 & b=0 \\ g & b=1 \end{cases} \quad (4)$$

Let $\vec{v}=(v_0,v_1,\cdots,v_{n-1})$ be a vector over *G* and $\vec{b}=(b_0,b_1,\cdots,b_{n-1})$ be a vector over *GF*(2), we define

$$\vec{b}\times\vec{v}=\vec{v}\times\vec{b}=(b_0\times v_0)\oplus(b_1\times v_1)\oplus\cdots\oplus(b_{n-1}\times v_{n-1}) \quad (5)$$

Definition 1. Let ***C*** be the following linear code over an Abelian group defined by

$$C=\{c=(c_0,c_1,\cdots,c_{m-1},c_m,c_{m+1}) \mid Hc^T=\vec{0}^T\},$$

where $\vec{c_i}=(c_{i1},c_{i2},\cdots,c_{im})$, $c_{ij} \in G$, and

$$H=\left[\begin{array}{ccccccc} I & I & I & \cdots & I & I & O_m \\ I & I+E & I+E^2 & \cdots & I+E^{m-2} & I+E^{m-1} & I \end{array}\right]$$

$$\equiv \left[P \mid I^*\right] \quad (6)$$

For these codes, the last 2 vectors, c_m and c_{m+1}, are parity-check vectors, and the other *m* vectors, i.e., c_j for $0 \le j \le m-1$, are information vectors.

In disk array applications, the Abelian group *G* can be computer words with bit-XOR operations.

The encoding process is to find $\vec{c_m}$, $\vec{c_{m+1}}$, given $\vec{c_0}$, $\vec{c_1}$, …, $\vec{c_{m-1}}$, by

$$\begin{bmatrix} \vec{c_m} \\ \vec{c_{m+1}} \end{bmatrix} = P \begin{bmatrix} \vec{c_0} \\ \vec{c_1} \\ \vdots \\ \vec{c_{m-1}} \end{bmatrix} \quad (7)$$

In the sequel, $p = m+1$ denotes a prime. Each codeword of the code *C* is represented by an array

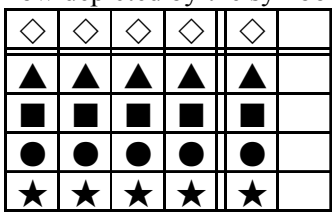
as in the case of the EVENODD code. Let $(a_{i,j})_{\substack{0\le i\le p-1\\ 0\le j\le p}}$ be a $p\times(p+1)$ array of the code *C*, where column *j*, $0\le j\le p-2$, represents an information symbol and columns $p-1$ and p represent parity symbols, and we assume that there is an imaginary 0-row, i.e., $a_{0,j}=0$, $0\le j\le p$. Although the parity-check matrix given by (6) is not systematic, the encoding procedure is very simple. First, we compute column $p-1$ by using the information bits in column *j*, $0\le j\le p-2$:

$$a_{i,p-1}=\bigoplus_{k=0}^{p-2} a_{i,k},\quad 1\le i\le p-1 \qquad (8)$$

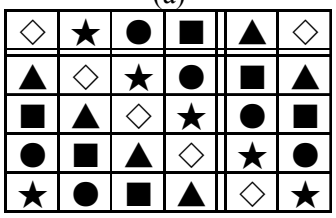
and next compute column p by using data in information columns and the parity bits in column $p-1$:

$$a_{i,p}=\bigoplus_{k=0}^{p-1} a_{<i+k>,k},\ 1\le i\le p-1 \qquad (9)$$

Equations (8) and (9) cannot be computed in parallel, but in software implementation this is not a problem. The proposed code, *C*, has a geometric description similar to that of the EVENODD code. Consider the code with $p=5$, for example. Fig. 3a and 3b show the horizontal parities given by (8) and the diagonal parities given by (9), respectively. Note that we do not compute the parity check of the imaginary 0-row depicted by the symbol ◇.



(a)



(b)

Fig. 3. (a) Horizontal and (b) diagonal parities of Rotary-code.

3.2 Erasure Decoding

According to Theorem 1, Rotary-code can correct up to any two erased columns. In other words, the minimum distance of the code is 3. Using the geometric description of the proposed code, we can give an efficient erasure decoding algorithm for the proposed code. It will be executed when a disk fails, or when two disks fail simultaneously.

Consider the $p\times(p+1)$ array of symbols $a_{i,j}$, such that the last two columns are redundant according to Equations (8) and (9). If one column (disk) has failed, say column (disk) *j*, $j\ne p$, then it can be retrieved using Equations (8). If column p fails, then the symbols can be retrieved using Equations (9).

Next, assume that columns (disks) j_1 and j_2 have failed, where $0\le j_1<j_2\le p$. We have two cases:

Case 1: $j_2=p$. Namely, the diagonal redundant disk and one other data disk have failed. We can reconstruct disk j_1 using Equation (8) and subsequently disk j_2 using Equation (9).

Case 2: $j_2<p$. This is the main case. We cannot retrieve them using the parities separately, as in the previous case. We analyze this case in detail. We retrieve the symbols in columns j_1 and j_2 as follows:

1. Set $s\leftarrow j_2-j_1$, $t\leftarrow p-j_2$, $h\leftarrow 0$ and $r\leftarrow 0$.
2. Calculate *w*, *w* satisfies $<w\times s>=j_1$ and $w<p$.
3. For($l=0;\ l<w;\ l++$){
   (a) $h\leftarrow<h+s>$;
   (b) $a_{h,j_2}\leftarrow a_{<h+t>,p}\oplus\sum_{k=0,k\ne j_2}^{p-1} a_{<h+t+k>,k}$;
   (c) $a_{h,j_1}\leftarrow\sum_{k=0,k\ne j_1}^{p-1} a_{h,k}$.}
4. For($l=0;\ l<p-1-w;\ l++$){
   (a) $r\leftarrow<r-s>$;
   (b) $a_{r,j1}\leftarrow a_{<r+s+t>,p}\oplus\sum_{k=0,k\ne j_1}^{p-1} a_{<r+s+t+k>,k}$;
   (c) $a_{r,j_2}\leftarrow\sum_{k=0,k\ne j_2}^{p-1} a_{r,k}$.}

Step 3 and step 4 can be computed in parallel to reduce the reconstruction time. The algorithm is not recursive and very simple to implement in software or in hardware.

Next we illustrate the decoding algorithm with an example.

Example 2. Let us now consider that the codeword of the array code with $p=5$ shown in Fig.4a.

Assume that columns (disks) 1 and 3 have been erased (lost). See Fig.4b.

0	1	0	1	0	0
1	1	1	0	1	0
0	0	1	1	0	0
1	0	1	0	0	1

(a)

0	?	0	?	0	0
1	?	1	?	1	0
0	?	1	?	0	0
1	?	1	?	0	1

(b)

Fig.4 (a) The codeword of the proposed code (b) Columns 1 and 3 have been erased.

We apply Case 2 of the erasure decoding algorithm. In this case, $s=2$, $t=2$ and $w=3$. The decoding process in the first loop is shown in Table 1.

Table 1. The decoding process in the first loop.

l	h	Calculations	Results
0	2	$a_{2,3}=a_{4,5}\oplus a_{4,0}\oplus a_{1,2}\oplus a_{3,4}$	0
		$a_{2,1}=a_{2,4}\oplus a_{2,0}\oplus a_{2,2}\oplus a_{2,3}$	1
1	4	$a_{4,3}=a_{1,5}\oplus a_{1,0}\oplus a_{2,1}\oplus a_{3,2}$	0
		$a_{4,1}=a_{4,4}\oplus a_{4,0}\oplus a_{4,2}\oplus a_{4,3}$	0
2	1	$a_{1,3}=a_{3,5}\oplus a_{3,0}\oplus a_{4,1}\oplus a_{2,4}$	1
		$a_{1,1}=a_{1,4}\oplus a_{1,0}\oplus a_{1,2}\oplus a_{1,3}$	1

$a_{0,j}$ ($0\le j\le p$) doesn't participate in calculations in the above table, the same hereinafter, because it is part of the imaginary 0-row, which has no effect on the results. After the first loop, we obtain $a_{2,3}=0$, $a_{2,1}=1$, $a_{4,3}=0$, $a_{4,1}=0$, $a_{1,3}=1$ and $a_{1,1}=1$. Next, we proceed to the second loop. The decoding process in the second loop is shown in Table 2.

Table 2. The decoding process in the second loop.

l	r	Calculations	Results
0	3	$a_{3,1}=a_{2,5}\oplus a_{2,0}\oplus a_{4,2}\oplus a_{1,4}$	0
		$a_{3,3}=a_{3,4}\oplus a_{3,0}\oplus a_{3,1}\oplus a_{3,2}$	1

After the second loop, we obtain $a_{3,1}=0$ and $a_{3,3}=1$. The decoding algorithm ends and the two erased columns (disks) are reconstructed successfully.

4 Complexity Comparison with Existing Schemes

There are mainly two existing schemes applicable to the RAID-6 specification. One is the Reed-Solomon (RS) codes [9], the other is EVENODD [1]. The results [1] show that EVENODD outperforms the RS codes because EVENODD does not require finite field computations. Hence, in this section, we compare the complexity of Rotary-code only with EVENODD from encoding, and decoding.

Both Rotary-code and EVENODD only need exclusive-OR (XOR) operations in encoding, decoding. The complexity can be measured by the numbers of XOR operations. In order to perform fair comparison, we assume that Rotary-code and EVENODD have the same array size for a given parameter p (p is a prime number greater than 2) by assuming the array for EVENODD has one imaginary data 0-column. The assumption is reasonable because the imaginary 0-column does not change the numbers of XOR operations in encoding, decoding for EVENODD.

4.1 Encoding complexity

At the encoding of Rotary-code, one parity symbol in the first parity column needs (p-2) XOR operations and the total of the first parity column is $(p-2)\times(p-1)$ using Equation (8). One parity symbol in the second parity column also needs (p-2) XOR operations because the imaginary 0-row symbol does not need XOR operations using Equation (9). The total of the second parity column is also $(p-2)\times(p-1)$. The total of XOR operations in the encoding of Rotary-code is

$$(p-2)\times(p-1)+(p-2)\times(p-1)=2p^2-6p-4$$

Theorem 2. For a 2-columns-erased tolerant array with n data columns and 2 parity columns, the lower bound of XOR operations per data symbol is $2-2/n$.

Proof: Assume that the coding array contains m rows. We have a minimum of two parity symbols per row of n data symbols, from the Singleton bound. The array size is $m\times(n+2)$. Each data symbol must contribute to at least two different parity symbols, one on each parity column, to ensure that we can recover if the data symbol and one parity symbol is lost. Therefore, we need to construct 2m parity symbols from equations that in the minimal formulation contain no common pairs of data symbols. The minimum number of

separately XORed input terms required to construct the 2*m* parity symbols is *2mn*. Accordingly, the minimum number of XOR operations is $2m(n-1)$. Therefore, the lower bound of XOR operations per data symbol to ensure 2-columns-erased tolerance is:

$$\frac{2m(n-1)}{mn} = \frac{2(n-1)}{n} = 2 - \frac{2}{n} \qquad \square$$

Rotary-code protects $(p-1)^2$ data symbols using $2p^2 - 6p + 4$ XOR operations. Let $n = p-1$, the count of XOR operations per each symbol is $\frac{2n^2 - 2n}{n^2} = 2 - \frac{2}{n}$, which meets the lower bound.

Now, we count the XOR operations in the encoding of EVENODD. The equations for computing the parity symbols are given below [1]:

$$a_{i,p} = \bigoplus_{k=0}^{p-1} a_{i,k}, \quad 0 \le i \le p-2 \tag{10}$$

$$a_{i,p+1} = \mathrm{S} \oplus \left(\bigoplus_{k=0}^{p-1} a_{<i-k>,k} \right), \ 0 \le i \le p-2 \tag{11}$$

$$S = \bigoplus_{k=1}^{p-1} a_{p-1-k,k} \tag{12}$$

Equation (10) equals to Equation (8) considering that the array for EVENODD has one imaginary data 0-column. The XOR operation count of the first parity column is also $(p-2)\times(p-1)$ using Equation (10). The adaptor *S* is computed before computing the parity symbols in the second parity column. $(p-3)$ XOR operations are required to obtain the adaptor *S* using Equation (12) and the assumption. In computing the second parity column using Equation (11) and the assumption, $(p-2)\times(p-1)$ XOR operations are needed because each parity symbol needs $p-2$ valid data symbols and the adaptor *S*. Therefore, the total of XOR operations in the encoding of EVENODD is

$$(p-2)\times(p-1)+(p-3)+(p-2)\times(p-1)=2p^2-5p-7 \tag{13}$$

As we can see, Rotary-code has faster encoding procedure than EVENODD by less than $p-3$ XOR operations for a given parameter *p* and the same array size. Table 3 compares Rotary-code to EVENODD for different values of *p*, assuming that *p* is prime (This is not a hard constraint, since the codes can be shortened to cover cases in which *p* is not a prime). The last column of Table 1 contains the reduced number of XOR operations between the number in column 2 (i.e., the number of operations needed in Rotary-code) and the number in column 3 (i.e., the number of operations needed in EVENODD). For instance, we can see that for *p* = 43 (last row), Rotary-code requires less 40 operations than EVENODD at the encoding. We can see from Table 1 that the number of XOR operations needed for encoding Rotary-code decreases accordingly with respect to EVENODD when the parameter *p* increases.

Table 3. Number of XOR operations needed to encode with the parameter *p*.

p	Rotary-code	EVENODD	Improvement
5	18	20	2
7	56	60	4
11	180	188	8
13	266	276	10
17	486	500	14
19	620	636	16
23	936	956	20
29	1530	1556	26
31	1760	1788	28
37	2546	2580	34
41	3150	3188	38
43	3476	2516	40

4.2 **Decoding complexity**

Rotary-code has optimal decoding complexity after any one column failure. Reconstruction from any one column failure requires exactly $(p-1)(p-2) = p^2 - 3p + 2$ XOR operations, since each last symbol is reconstructed by XORing the surviving (*p*−1) symbols in the first parity set or in the second parity set and the last column has (*p*-1) symbols. Let *n*=*p*−1, we construct *n* symbols with $(n^2 - n)$ XOR operations, which is (*n*−1) XOR operations per parity symbol. It is already shown that it has the minimum number of XOR operations for construction of an array that double protects parity, and since all parity sets are the same size, the cost to repair any one lost column is the same and is also a lower bound.

Reconstructing from any double failure that includes the second parity column is exactly the same cost as parity construction. Reconstructing any of the data columns from the first parity column has the same cost as constructing the first parity column. The cost of reconstructing any combination of two columns can also be determined by $2p^2 - 6p + 4 = 2n^2 - 2n$ XOR operations required to reconstruct $2(p-1)$ data symbols. Therefore, Rotary-code has optimal decoding complexity after any two column failure.

Comparing again to EVENODD, using the data reconstruction algorithm described in the

EVENODD paper, we see that different failure mode has different reconstruction cost. To compare the complexity fairly, we classify the columns into three types: data columns denoted by *D*, the first parity column denoted by *P*, the second parity column denoted by *Q*. There are three types: *D*, *P* and *Q* for any one column failure. And (*D, D*), (*D, P*), (*D, Q*) and (*P, Q*) are all the possible types for any two column failures. Assume that ach column has the same failure probability. It is calculated from the array with one imaginary 0-column that the average of XOR operations is $\frac{p^3-2p^2-1}{p+1}$ for one column failure and $\frac{2p^4-2p^3-9p^2+13p-24}{p(p+1)}$ for two column failures.

Table 4. Number of XOR operations needed to decode from two failures with the parameter *p*.

p	Rotary-code	EVENODD	Improvement
5	18	27.2	9.2
7	56	66.8	10.8
11	180	194.3	14.3
13	266	282.2	16.2
17	486	505.9	19.9
19	620	641.8	21.8
23	936	961.7	25.7
29	1530	1561.6	31.6
31	1760	1793.5	33.5
37	2546	2585.5	39.5
41	3150	3193.4	43.4
43	3476	2521.4	45.4

It is obvious that Rotary-code needs less XOR operations to reconstruct from one column failure than EVENODD. Table 4 compares the decoding complexity of Rotary-code from two column failure to EVENODD for different values of *p*. We can see from Table 4 that the number of XOR operations needed for decoding Rotary-code from two failures decreases accordingly with respect to EVENODD when the parameter *p* increases.

5 The MDS Property

In this section, we state and prove the MDS property of Rotary-code.

Theorem 3—MDS Property: Rotary-code has column distance of 3, i.e., it is MDS, if and only if *p* is a prime number.

Proof: Let us start with the sufficient condition, namely, to prove that for any prime number *p*, Rotary-code is MDS.

First observe that Rotary-code is a linear code, thus proving that the code has distance of 3 is equivalent to proving that the code has minimum column weight w_{min} of 3, i.e., a valid codeword of Rotary-code has at least three nonzero columns. (A column is called a nonzero column if at least one symbol in the column is nonzero.) We will prove it by contradiction.

From the definition of Rotary-code, it is impossible to have only one nonzero column, thus $w_{min} > 1$.

Now suppose w_{min}=2, then without loss of generality, we can assume that v_i and v_j, $0 \le i < j \le p$, are nonzero vectors and the others are the all-zero vectors in a codeword $v = (v_0, v_1, \cdots, v_p)$ of weight 2. If $j = p$, then we have $v_i = 0$ since $vH^T = 0$. Thus, we get a contradiction. Next, we assume that $j<p$. In this case, we have $v_i = v_j$ and $v_i E^{iT} = v_j E^{jT}$. From these equations, we have

$$(v_i, 0)E_p^{iT} = (v_i, 0)E_p^{jT} + (0, a) \qquad (14)$$

where "+" denotes component wise modulo 2 addition and $a \in GF(2)$. Let *S* be the sum of all the components in v_i. From (14), we have $S = S + a$ since E_p^i and E_p^j are permutation matrices. So, we get $a = 0$. Hence, (14) becomes $(v_i,0)E_p^{(j-i)T} = (v_i,0)$. We have $(v_i,0)E_p^{l(j-i)T} = (v_i,0)$ for $1 \le l \le p-1$. Since $0 < j - i < p$ and p is a prime, we have $(v_i,0)E_p^{kT} = (v_i,0)$ for $1 < k < p$-1. Especially, $(v_i,0)E_p^T = (v_i,0)$. It is known that, for *w*, a binary vector of length *p*, $wE_p^T = w$, if and only if *w* is the all-zero or all-one vector. Since $(v_i,0)$ is a nonzero vector whose last component is zero, we get a contradiction. Thus $w_{\min} \ge 3$, but it is easy to see there is a codeword of column weight 3, so w_{min} = 3. This concludes the proof for the sufficient condition.

On the other hand, if *p* were not a prime number, then there exists a positive integer *d*, $1 < d < p$, such that *d* divides *p*. We define a binary vector $u = (u_0, u_1, \cdots, u_{p-2})$ as follows:

$$u_i = \begin{cases} 1 & \text{if } i \text{ is multiple of } d \\ 0 & \text{otherwise} \end{cases} \qquad (15)$$

Then, $v = (v_0, v_1, \cdots, v_p)$, where $v_i = v_p = u$ and v_j, $1 \le v \le p$, $j \ne d$, is the all-zero vector, is a

codeword of C with weight 2, or the distance of the code is no greater than 2, which contradicts with the fact that the code is of distance 3. So p being a prime number is a necessary condition to the MDS property of Rotary-code. □

6 Implementation and Performance

The implementation of the Rotary-code encoding is straightforward and simply follows the procedure described in Section 3.1. Thus, in this part, our main focus is on the erasure decoding procedure. As stated in Section 4.2, the decoding complexity is also optimal. The decoding algorithm in Section 3.2 is not recursive and very simple to implement in software or in hardware. To achieve the maximum performance, we apply the parallel technique in the decoding procedure as described in the earlier section.

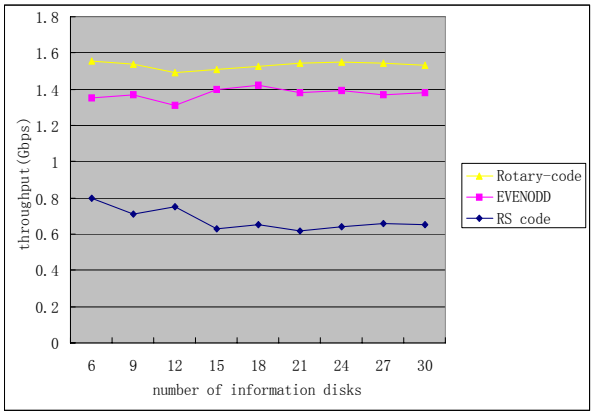


Fig. 5. Throughput performance (2 erasures are randomly generated among information disks).

We have implemented the Rotary-code in C/C++ and apply it to a reliable storage platform [17]. The throughput performance is measured and compared to the publicly available implementation of the XOR-based RS code [16], the EVENODD code [15]. The results are shown in Fig. 5, where the size of a single data block from each disk is 2,880 bytes and the number of information disks varies from 6 to 31. Note that our focus is on decoding erasures that all occur at information columns since, otherwise, Rotary-code just reduces to RAID-5 (when there is the second parity column erasure), so we only simulate random information column erasures in Fig. 5. Recall that a single data block from each disk corresponds to a single column in Rotary-code and is divided into $p-1$ symbols, so the block size needs to be a multiple of $p-1$. For comparison purposes, we use 2,880 here since it is a common multiple of $p-1$ for most p values in the range. In real applications, we are free to choose the block size to be any multiple of $p-1$ once p, as a system parameter, is determined. These results are obtained from experiments on a Pentium 4 1.6 GHz Linux machine with 512 Mbytes of memory running Redhat 9.0. It is clear that Rotary-code achieves throughput that is about twice that of the RS code and is more than that of the EVENODD code. Note that there are jigsaw effects in the throughputs of both EVENODD and Rotary-code. This happens mainly due to the shortening technique. When the number of storage nodes is not prime, the codes are constructed using the closest larger prime number. A larger prime number means that each column (data block here) is divided into more pieces, which in turn incurs additional control overhead. As the number of information disks increases, the overhead is then amortized, reflected by the performance ramping up after each dip. (Similarly, the performance of the RS code shows jigsaw effects too, which happens at the change of L due to the increment of total disks n.)

7 Conclusion

In this paper, we have presented a class of MDS array codes, called Rotary-code, which are based on a low-density parity-check matrix. The codes have a sparser parity-check matrix than the EVENODD code and have two major advantages over the other 2-erasure codes. One is that Rotary-code has optimal storage efficiency and applicable to the RAID-6 specification. The other is that Rotary-code uses only simple XOR and cyclic shift operations and has optimal encoding and decoding complexity.

From the perspective of error-correcting codes, we have constructed a new code that is capable for correcting two erasures. The natural generalization works of Rotary-code is to construct the codes for the case of 3 erasures as well as for 4 erasures. In a sequel paper, we will present an extension of Rotary-code to a more than two erasures case.

References:

[1] M. Blaum, J. Brady, J. Bruck, and J. Menon, EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures, *IEEE Trans. Computers*, vol. 44, no. 2, pp. 192-202, Feb. 1995.

[2] J. S. Plank. The RAID-6 Liberation codes. *In FAST-2008: 6th Usenix Conference on File and Storage Technologies*, pp. 97–110, San Jose, February 2008.

[3] M. Blaum, J. Bradt, J. Bruck, J. Menon, and A. Vardy, The EVENODD Code and Its

Generalization: An Efficient Scheme for Tolerating Multiple Disk Failures in RAID Architectures, *High Performance Mass Storage and Parallel I/O*, chapter 14, 2002.

[4] CORBETT, P., ENGLISH, B., GOEL, A., GRCANAC, T., KLEIMAN, S., LEONG, J., AND SANKAR, S. Row diagonal parity for double disk failure correction. *In 4th Usenix Conference on File and Storage Technologies* , San Francisco, CA,March 2004.

[5] Cheng Huang, Lihao Xu, STAR: An Efficient Coding Scheme for Correcting Triple Storage Node Failures. *IEEE Trans. Computers*, vol. 57, no. 7, pp. 889-901, July 2008.

[6] L. Xu and J. Bruck, X-Code: MDS Array Codes with Optimal Encoding, *IEEE Trans. Information Theory*, pp. 272-276, Jan. 1999.

[7] L. Xu, V. Bohossian, J. Bruck, and D. Wagner, Low Density MDS Codes and Factors of Complete Graphs, *IEEE Trans. Information Theory*, vol. 45, no. 1, pp. 1817-1826, Nov. 1999.

[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, August 1995.

[9] J. S. Plank. A tutorial on Reed-Solomon coding for faulttolerance in RAID-like systems. *Software – Practice & Experience*, vol. 27, no. 9, pp. 995–1012, September 1997.

[10] J. S. Plank and Y. Ding. Note: Correction to the 1997 tutorial on Reed-Solomon coding. *Software – Practice & Experience*, vol. 35, no. 2, pp.189–194, February 2005.

[11] D. Patterson, G. Gibson, and R. Katz, A Case for Redundant Arrays of Inexpensive Disks (RAID), *Proc. ACM SIGMOD '88*, pp. 109-116, June 1988.

[12] G.-L. Feng, R.H. Deng, F. Bao, and J.-C. Shen, New Efficient MDS Array Codes for RAID Part I: Reed-Solomon-Like Codes for Tolerating Three Disk Failures, *IEEE Trans. Computers*, vol. 54, no. 9, pp. 1071-1080, Sept. 2005.

[13] J. L. Hafner. WEAVER Codes: Highly fault tolerant erasure codes for storage systems. *In FAST-2005: 4th Usenix Conference on File and Storage Technologies*, pp. 211–224, San Francisco, December 2005.

[14] J. L. Hafner. HoVer erasure codes for disk arrays. *In DSN-2006: The International Conference on Dependable Systemsand Networks*, Philadelphia, June 2006.

[15] PLANK, J. S. Jerasure: A library in C/C++ facilitating erasure coding for storage applications. *Tech. Rep. CS-07-603*, University of Tennessee, September 2007.

[16] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, http://www.icsi.berkeley.edu/~luby/cauchy.tar.uu, 2007.

[17] L. Xu, Hydra: A Platform for Survivable and Secure Data Storage Systems, *Proc. Int'l Workshop Storage Security and Survivability*, Nov. 2005.