International Journal of Innovative
Computing, Information and Control
Volume **14**, Number **3**, June **2018**

ICIC International ©2018 ISSN 1349-4198
pp. **1079–1094**

A RANDOMLY EXPANDABLE METHOD FOR DATA LAYOUT OF RAID STORAGE SYSTEMS

Dezhai Yuan[1,2,*], Xingyi Peng[1,2], Ting Liu[1,2] and Zhe Cui[1]

[1]Chengdu Institute of Computer Applications
Chinese Academy of Sciences
No. 9, South Renmin Road, Sec. 4, Chengdu 610041, P. R. China

[2]School of Computer and Control Engineering
University of Chinese Academy of Sciences
No. 19(A), Yuquan Road, Shijingshan District, Beijing 100049, P. R. China
*Corresponding author: yuandezhai12@mails.ucas.ac.cn
pengxingyi15@mails.ucas.ac.cn; 15639906259@wo.cn; cuizhe@casit.com.cn

Received October 2017; revised February 2018

Abstract. *With the increase in data volume, velocity, and variety, the need for redundant arrays of inexpensive disks (RAID) storage systems capacity is growing dramatically. However, the probability of disk failures in RAID storage systems is sharply higher with increased program/erase cycles, read cycles, and retention time. RAID storage systems are faced with more challenges in fault tolerance, storage efficiency, computational complexity, and expandability. This article presents a novel data layout scheme for RAID storage systems using random binary extensive code (RBEC), which is designed to ensure random expandability, high reliability, and availability of data in RAID storage systems. RBEC is a systematic code family in which the generator matrix consists of two submatrices with entries over GF(2), an identity matrix on the top, and another submatrix on the bottom. Compared with existing approaches, the attractive advantages of our scheme include the following: (1) it is completely implemented based on simple eXclusive OR (XOR) operations and has systematic code properties, (2) it can provide arbitrary fault tolerance, (3) its storage efficiency is quasi-optimal, and (4) RAID storage system data and parity disks can be randomly expanded according to the requirements of the practical systems. Thus, our scheme is particularly suitable for RAID storage systems that need higher reliability, availability, and expandability.*
Keywords: Random matrix, Data layout, Fault tolerance, RAID

1. **Introduction.** Because the storage availability and reliability of redundant arrays of inexpensive disks (RAID) [1] storage systems are seriously degraded when program/erase cycles, read cycles, and retention time are increased, one of the most urgent challenges is to provide sufficient availability and reliability to prevent data losses and corruption during storage. Consequently, reliable and practical fault-tolerant technology is required to ensure successful data recovery from several varieties of failures occurring in storage systems. This kind of technology for protecting data from disk failures is divided into two groups: N-way mirroring and erasure code. The N-way mirroring is widely used in actual storage systems, such as GFS [2], Hadoop [3] and Dynamo [4], whose significance lies in providing additional redundancy to ensure successful recovery, but the storage efficiency is exceedingly low. Due to the low storage space utilization efficiency of the N-way mirroring technique, erasure code is more suitable for storage systems with low redundancy and high fault tolerance compared with the N-way mirroring technique. Because of its high efficiency and practicality, erasure code has gradually attracted more and more attention

from the industry and academy, and has thus become a hot research topic in the field of data storage in recent years.

All kinds of erasure codes have been proposed for RAID storage systems after years of painstaking research and development, especially the widely used Reed-Solomon (RS) [5] and parity array codes, including, EVENODD [6], WEAVER [7], FENG codes [8,9], and so forth. Each of them has obvious advantages, but none has become the perfect standard. On the one hand, RS codes provide optimal storage efficiency and arbitrarily high fault tolerance, but require special-purpose hardware to enable efficient computation of the Galois field arithmetic on which the codes are based and generally have higher computation costs and complexities. On the other hand, parity array codes are completely based on XOR operations, but have relatively irregular geometric construction and cannot be randomly expanded with increased RAID storage system data and parity disks in accordance with actual requirements. Fortunately, several novel and efficient erasure codes have also been proposed in recent two years to overcome these difficulties, for instance, Short Codes [10], XI-Codes [11], HV Codes [12], and Elastic-RAID [13]. Short Codes are an MDS erasure codes family which can provide satisfied performance on both degraded reads and partial stripe writes. XI-Codes are a new family of the lowest density MDS array codes over $GF(2)$ without extremely strict constraints on the prime number of code length, which makes them pretty practical. HV Codes are all-round MDS codes by taking advantage of horizontal parity and vertical parity, which well balance the load to the disks and offer an optimized partial stripe write experience. Elastic-RAID is a novel RAID architecture which efficiently combines the advantages of the mirroring-based RAID and the parity-based RAID by exploiting and utilizing the available/free space in a parity-based RAID system itself without any additional hardware capacity. In summary, many RS codes, parity array codes, and several novel and efficient erasure codes have also been proposed recently; however, none is first-rank because each of them has inherent disadvantages and a specific application scenario. Although there is extensive research on erasure codes for RAID storage systems that involves balancing fault tolerance, storage efficiency, and computation complexity, very few efforts have been made to improve flexible expandability, which is important performance metric in RAID storage systems.

Motivated by the fact that RAID storage system data and parity disks cannot be randomly expanded according to actual requirements, we present a randomly expandable method for RAID storage system data layout adopting random binary extensive code (RBEC) [14] to encode and decode data. Compared with the existing approaches, the attractive advantages of our scheme include the following: (1) it is completely implemented based on simple XOR operations and has systematic code properties which are more efficient than traditional RS codes in terms of computational complexity, (2) it can provide arbitrary fault tolerance, (3) its storage efficiency is quasi-optimal, and (4) RAID storage system data and parity disks can be randomly expanded according to actual requirements. The advantages of adopting a randomly expandable method are obvious. This is a kind of probabilistic methods over simple XOR operations which ensure the successful recovery of original data at a sky-high probability. The deterministic methods, however, pay such a huge price over complex Galois field arithmetic for successful recovery. Thus, all these advantages make our scheme rather suitable for RAID storage systems that need high reliability, sufficient availability, and flexible expandability.

This article is organized as follows. In the next section, we briefly review previous research work related to erasure codes. In Section 3, we propose the preliminaries used in the construction of our randomly expandable method. Section 4 describes our proposed scheme. Section 5 provides performance analysis, comparisons, and implementation of the proposed scheme. Finally, conclusions are given in Section 6.

2. **Related Work.** In this section, we divide the existing erasure codes into three fundamental categories and cite some instances visually referring to Plank and Huang [15].

Reed-Solomon (RS) Codes: A widely used code developed by Reed and Solomon is based on the Vandermonde matrix. RS codes [5] are a family of MDS codes that provide optimal storage efficiency and arbitrarily high fault tolerance. However, RS codes are over Galois field $GF(2^w)$, so they require special-purpose hardware to enable efficient computation of the Galois field arithmetic and generally have higher computation costs and complexities. Subsequently, Cauchy_RS Codes [16] are represented via the Cauchy matrix using XOR-based operations in the form of a $GF(2^w)$ by $w \times w$ matrix over $GF(2)$ instead of complicated Galois field arithmetic over $GF(2^w)$. Derived from RS Codes, FENG codes [8,9] include Reed-Solomon-Like Code and Rabin-Like Code. To reduce the complexity of the Galois field arithmetic operation, the cyclotomic fast Fourier transform algorithm is also presented in the implementation of RAID based on RS codes, which is much lower than the existing MDS array codes [17].

Parity Array Codes: There are at least three types of parity array codes: (1) horizontal codes, such as Row Diagonal Parity [18], EVENODD [6] and generalized X-Code [19]; (2) vertical codes, such as WEAVER [7], CCode [20], X-Code [21] and P-Code [22]; and (3) 2-dimensional (or higher N-dimensional) horizontal and vertical code, such as HoVer codes [23] and GRID codes [24]. A general characteristic of parity array codes is that they are implemented based on simple XOR operations. This is more efficient than traditional RS codes using complicated Galois field operations for encoding and decoding processes in terms of computational complexity. Although we have many parity array codes, none is optimal because each of them has inherent disadvantages and a specific scope of application. Obviously, parity array codes have relatively irregular geometric construction and cannot be randomly expanded with the increased RAID storage system data and parity disks according to actual requirements.

New Codes: In addition, more and more innovative approaches have been presented recently, for example, low density parity codes (LDPC) [25], CRC-Detect-First-LDPC (CDF-LDPC) [26], Regenerating codes [27], Sector Disk (SD) codes [28], STAIR codes [29], HACFS codes [30], and Random RAID codes [31]. LDPC are linear codes that are completely XOR-based defined by bipartite graphs with data elements on the left and parity elements on the right. LDPC include Tornado codes [32], LT codes [33] and their improvement, Raptor codes [34]. The CDF-LDPC algorithm is a new error correction method for Solid-State Drive (SSD) that combines error detection code (EDC, such as cyclic redundancy code, and parity check code) with error correction code (ECC, such as LDPC) to improve the read performance of SSD. Regenerating codes are designed to decrease bandwidth for recovery by increasing more element blocks than before that each storage node holds. SD codes and STAIR codes are invented to tolerate mixed failure models and concurrent sector and disk failures, and are more efficient than the traditional codes that solely tolerate whole disk failures. HACFS is a novel erasure-coded storage system that uses a fast code to optimize recovery performance and a compact code to reduce the storage overhead instead of using two different erasure codes. Random RAID is a new kind of fault-tolerance method that uses a probabilistic approach to achieve high fault-tolerance and flexible scalability.

3. **Preliminaries.** In this section, we will briefly introduce the coding model and terminologies of erasure codes, and define the random matrix used in the RBEC code [14]. Finally, RBEC code will be presented for the construction of our scheme.

3.1. **Coding model and terminologies.** The basic idea of erasure codes is to encode the k original data blocks into n encoded data blocks. When t pieces of these blocks are lost, the original data blocks can be totally reconstructed from the remaining $n-t$ pieces, such an erasure code is called an (n,k) coding model as represented in Figure 1. If $t=n-k$, this can be called a maximum distance separate (MDS) code, which provides optimal storage efficiency [35].

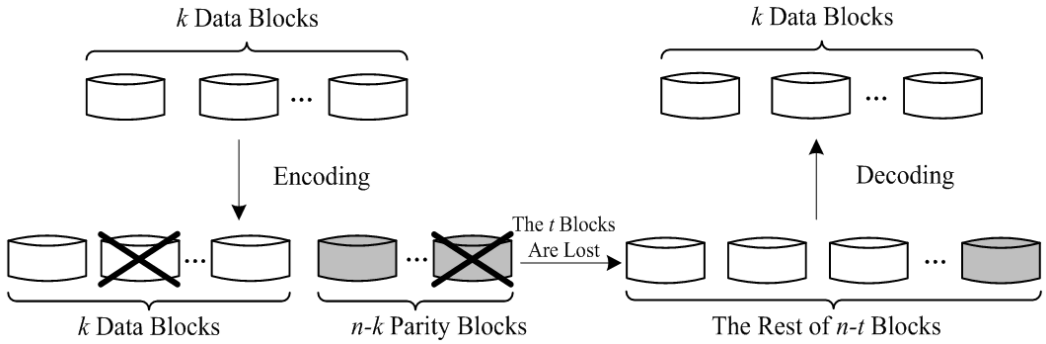


FIGURE 1. The (n,k) coding model

The encoding process can be regarded as a sort of mathematical transform. Generally, it can be realized by a few classic means such as the Vandermonde matrix, which is widely used in RS codes. In terms of coding theory, the encoding and decoding of the (n,k) coding model can be equivalently expressed as two specific matrices, namely, the generator matrix and the parity-check matrix. The former is used to generate the n encoded data blocks, and the latter is for reconstructing the k original data blocks. The successful recovery of original data blocks lies in the orthogonality of the generator matrix and the parity-check matrix.

To avoid possible confusions, some terminologies on erasure codes and RAID storage systems are enumerated [36]. And some of them will be used throughout this paper to describe and evaluate our scheme.

- Parity: bits, bytes or blocks that carry generated redundant parity blocks for recovery.
- Element: the basic building block of erasure codes usually referring to a unit of data or parity. In coding theory, this is a bit within a code symbol.
- Stripe: a connected set of data and parity elements that are dependently related by coding. In coding theory, this is a codeword, and its length is usually defined as the number of disks over which it stretches, i.e., the i-th codeword component is stored on the i-th disk.
- Strip: a stripe unit or a maximal set of continuous elements in a stripe that are stored on the same disk. In coding theory, this is a code symbol.
- Array: a collection of disks on which one or more stripes are implemented.
- Stack: a collection of stripes in an array that are related by a maximal set of permutations of logical mappings of strip number to disk.
- Systematic Code: its codeword is divided into two parts: the data part and parity part. The data part is not modified after encoding and can be directly read if there are no errors.
- Vertical Code: a kind of erasure code in which a strip contains both data elements and parity elements.
- Horizontal Code: a kind of erasure code in which a strip contains either data elements or parity elements, but a stripe contains both data elements and parity elements.

- Storage Efficiency: the proportion of a stripe that contains data elements known as the number of data elements divided by the total number of data and parity elements.
- Fault Tolerance: the maximum number of lost strips that can be accurately reconstructed by erasure codes.
- Complexity: the computational costs of encoding, decoding and updating.

To visually conceptualize the terminologies and structure of RAID storage systems, Figure 2 represents the data layout of elements, strips, stripes, stacks, and arrays in the typical horizontal codes of RAID storage systems [36].

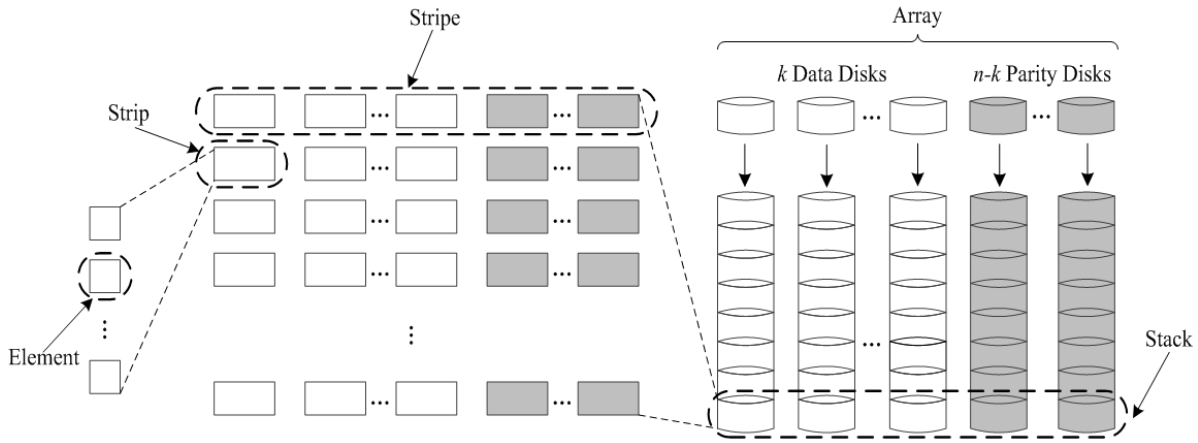


FIGURE 2. The data layout of horizontal codes in RAID storage systems

3.2. **Definition of the random matrix.**

Definition 3.1. *Let $M = (m_{i,j})_{n\times n}$ be a random $n \times n$ matrix over $GF(2)$ whose entries are independently and identically distributed. The random matrix is defined by*

$$\Pr(m_{i,j} = r) = \begin{cases} 1-p, & r = 0 \\ p, & r = 1, \end{cases} \tag{1}$$

where p denotes the probability of an entry being 1.

For simplicity, let us suppose that $p = 0.5$ and $\Pr(m_{i,j} = 0) = \Pr(m_{i,j} = 1) = 0.5$, such that all matrix elements are uniformly random. The elaborate generating process is described by Algorithm 1.

3.3. **The properties of the random matrix.**

From previous work [37], we can easily derive the probability of the generated random matrix being nonsingular.

Lemma 3.1.

$$\Pr(Rank(M_{n\times n}) = n) = \prod_{i=1}^{n} \left(1 - \frac{1}{2^i}\right). \tag{2}$$

Proof: Let $M_{n\times n} = (\eta_1, \eta_2, \ldots, \eta_n)$ be a random matrix consisting of n columns and let η_i be the i-th column, where $1 \le i \le n$. $Rank(M_{n\times n}) = n \Leftrightarrow$ Each i-th column cannot be linearly combined with the first $i-1$ columns, denoted by

$$L(i) = 1 - \frac{2^{i-1}}{2^n}, \quad 1 \le i \le n \tag{3}$$

where 2^n is the totality of the i-th column constitution over $GF(2)$, 2^{i-1} is the first $i-1$ columns combination, and $\frac{2^{i-1}}{2^n}$ is the probability of the i-th column being linearly combined by the first $i-1$ columns. So,

$$\begin{aligned}\Pr\left(Rank(M_{n\times n})=n\right)&=\prod_{i=1}^{n}L(i)=\left(1-\frac{2^0}{2^n}\right)\left(1-\frac{2^1}{2^n}\right)\ldots\left(1-\frac{2^{n-1}}{2^n}\right)\\&=\left(1-\frac{1}{2^n}\right)\left(1-\frac{1}{2^{n-1}}\right)\ldots\left(1-\frac{1}{2}\right)\\&=\prod_{i=1}^{n}\left(1-\frac{1}{2^i}\right)=S(n,n).\end{aligned}\tag{4}$$

$\prod_{i=1}^{n}\left(1-\frac{1}{2^i}\right)$ is simply expressed in terms of $S(n,n)$ which denotes the probability of the generated random matrix $M_{n\times n}$ being nonsingular. The explicit value of $S(n,n)$ has not been solved by the scientific community so far, but we find that the function tends to

Algorithm 1. Construction of $\{0,1\}$ random matrix

```
Input: Size of random matrix;
Output: The generated n × n random matrix composed of {0, 1}
repeat
    for i from 1 to n step by 1 do
        for j from 1 to n step by 1 do
            Generate a random floating number between 0 and 1;
                Rnd_{i,j} = Rand()/Double(RAND_MAX);
            if (0 ≤ Rnd_{i,j} ≤ 0.5)
                m_{i,j} = 0;
              else
                m_{i,j} = 1;
            endif
        end
    end
until i = n and j = n;
```

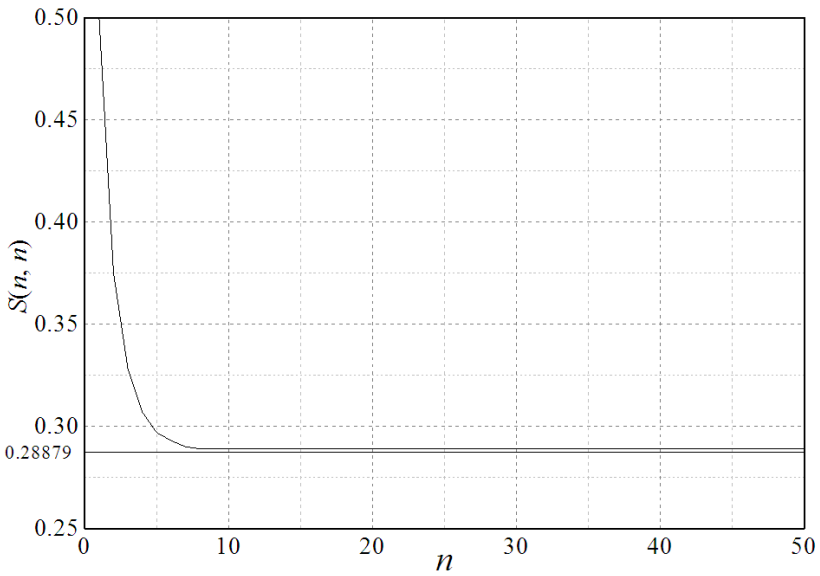


FIGURE 3. The tendency of $S(n,n)$

a constant of 0.28879 when $n \geq 10$ through computational simulation. The tendency of $S(n,n)$ is revealed in Figure 3 in which the x-axis represents n, and the y-axis refers to the approximate tendency of $S(n,n)$.

On the basis of the construction of a random $n \times n$ matrix and its probability of being nonsingular, the random $(n+k) \times n$ matrix, called the high matrix $G_{(n+k)\times n}$, can be easily inferred and the probability of $G_{(n+k)\times n}$ being full column rank is also defined by the following.

Lemma 3.2.

$$\Pr\left(Rank\left(G_{(n+k)\times n}\right) = n\right) = \prod_{i=k+1}^{n+k}\left(1 - \frac{1}{2^i}\right). \tag{5}$$

Proof:

$$\begin{aligned}
&\Pr\left(Rank(G_{(n+k)\times n}) = n\right) \\
&= \prod_{i=0}^{n-1}\left(1 - \frac{2^i}{2^{n+k}}\right) = \left(1 - \frac{2^0}{2^{n+k}}\right)\left(1 - \frac{2^1}{2^{n+k}}\right)\cdots\left(1 - \frac{2^{n-1}}{2^{n+k}}\right) \\
&= \left(1 - \frac{1}{2^{k+1}}\right)\left(1 - \frac{1}{2^{k+2}}\right)\cdots\left(1 - \frac{1}{2^{n+k}}\right) \\
&= \prod_{i=k+1}^{n+k}\left(1 - \frac{1}{2^i}\right) = S(n+k,n).
\end{aligned} \tag{6}$$

From $\prod_{i=k+1}^{n+k}\left(1 - \frac{1}{2^i}\right)$ we can easily conclude that n has little effect on the trend of $S(n+k,n)$ with an increase in k. The value of $S(n+k,n)$ is extremely close to 1 when $k \geq 10$, and the tendency of $S(n+k,n)$ is clear as Figure 4 illustrates.

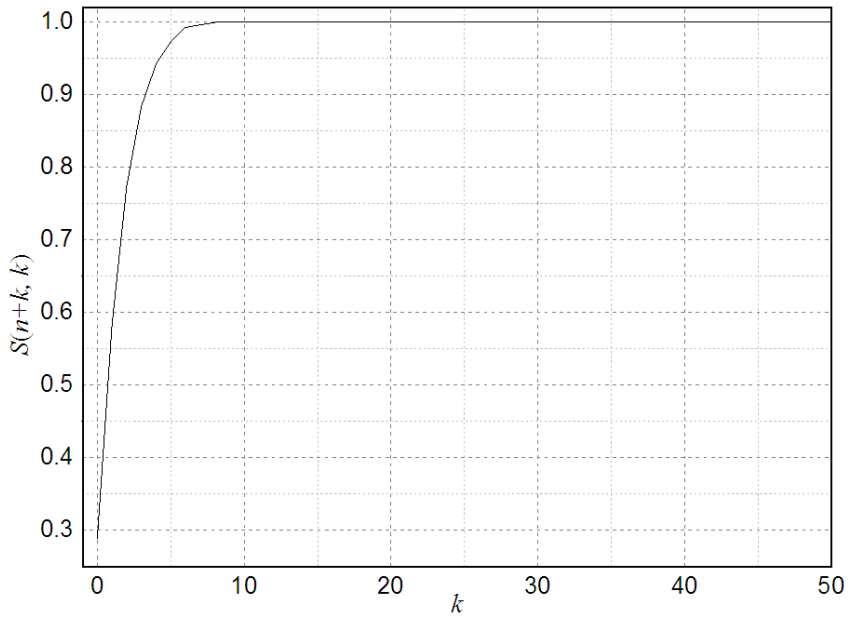


FIGURE 4. The tendency of $S(n+k,n)$

3.4. **RBEC.** RBEC is a systematic code family that can be represented by a generator matrix consisting of two submatrices with entries over $GF(2)$, an identity matrix on the top, and another random submatrix on the bottom. The successful decoding of RBEC code primarily lies in the high probability of being full column rank of the random matrix.

RBEC Encoding: The purpose of RBEC encoding is to generate a codeword combined with original data blocks and encoded parity blocks. RBEC encoding is actually a more efficient process which is a product of original data blocks and the RBEC generator matrix as shown in Equation (8). For example, $G \cdot D = C$, where D is original data with k blocks, $G_{n\times k}$ is the RBEC generator matrix, and C is the codeword with n blocks. Let $I_{k\times k}$ be a $k \times k$ identity matrix and $R_{(n-k)\times k}$ be an $(n-k) \times k$ random matrix. Now, we define $G_{n\times k}$ as follows:

$$G_{n\times k} = \begin{bmatrix} I_{k\times k} \\ R_{(n-k)\times k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ r_{1,1} & r_{1,2} & \cdots & r_{1,k} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n-k,1} & r_{n-k,2} & \cdots & r_{n-k,k} \end{bmatrix}, \tag{7}$$

where $r_{ij} \in GF(2)$ for $1 \le i \le k$ and $1 \le j \le n$.

RBEC Decoding: The RBEC decoding process can be briefly summarized as the process of reconstructing the original data D by the parity-check matrix $H_{k\times(n-k)}$ which can be derived from $G_{n\times k}$ and defined as:

$$H_{n\times(n-k)} = \begin{bmatrix} R^T_{k\times(n-k)} \\ I_{(n-k)\times(n-k)} \end{bmatrix} = \begin{bmatrix} r_{1,1} & r_{2,1} & \cdots & r_{n-k,1} \\ r_{1,2} & r_{2,2} & \cdots & r_{n-k,2} \\ \vdots & \vdots & \ddots & \vdots \\ r_{1,k} & r_{2,k} & \cdots & r_{n-k,k} \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}, \tag{8}$$

where $R^T_{k\times(n-k)}$ is the transpose of $R_{(n-k)\times k}$. In addition, it can be easily checked that $G^T_{k\times n} \times H_{n\times(n-k)} = 0_{k\times(n-k)}$, where $0_{k\times(n-k)}$ is a $k \times (n-k)$ all-zero matrix. Thus, we have $H^T_{(n-k)\times n} \times C_{n\times 1} = 0_{(n-k)\times 1}$, and the decoding process can be reduced to a solving system of equations. For more details, please see [14].

4. **Our Proposed Scheme.**

In this section, we present a novel data layout scheme for RAID storage systems that can randomly expand RAID data and parity disks according to actual requirements. Before presenting our basic data layout scheme, we firstly introduce some corresponding justifications in this subsection to make the setting of the data layout feel more reasonable. On the one hand, the codeword length can be dynamically adjusted by folding a one-dimensional codeword into a two-dimensional array placed in RAID storage system data and parity disks according to actual requirements. On the other hand, the granularity of error correction (the size of one strip which is protected together by an erasure code) can be dramatically decreased from one entire disk to some sectors or blocks of the one disk.

4.1. **A basic data layout scheme.**

The above observations motivate us to propose a basic data layout scheme, which utilizes the RBEC codes to encode and decode data in RAID storage systems. It provides higher reliability, availability, and expandability by introducing a new data layout scheme. Suppose that there exist such a RAID

storage system containing five data disks numbered 1 through 5 and three parity disks numbered 1, 2, and 3. We first need to initialize a 40×25 generator matrix $G_{40\times25}$ with a 25×25 identity matrix on the top and another 15×25 random submatrix on the bottom based on RBEC code. Then, we regard the original data with 25 blocks $(D_1, D_2, D_3, \ldots, D_{24}, D_{25})$ as the message D that needs to be encoded. We will use message D to generate codeword W with the generator matrix $G_{40\times25}$, defined by $W = G_{40\times25} \cdot D$. Because the upper part of $G_{40\times25}$ is a 25×25 identity matrix, the former components of the codeword W are identical to message D, so the codeword W can be denoted as $W = (D_1, D_2, D_3, \ldots, D_{24}, D_{25}, P_1, P_2, P_3, \ldots, P_{14}, P_{15})$. Finally, for array reasons, the generated one-dimensional codeword W itself will be arranged into a 5×8 two-dimensional array placed in RAID storage system data and parity disks, as Figure 5 illustrates.

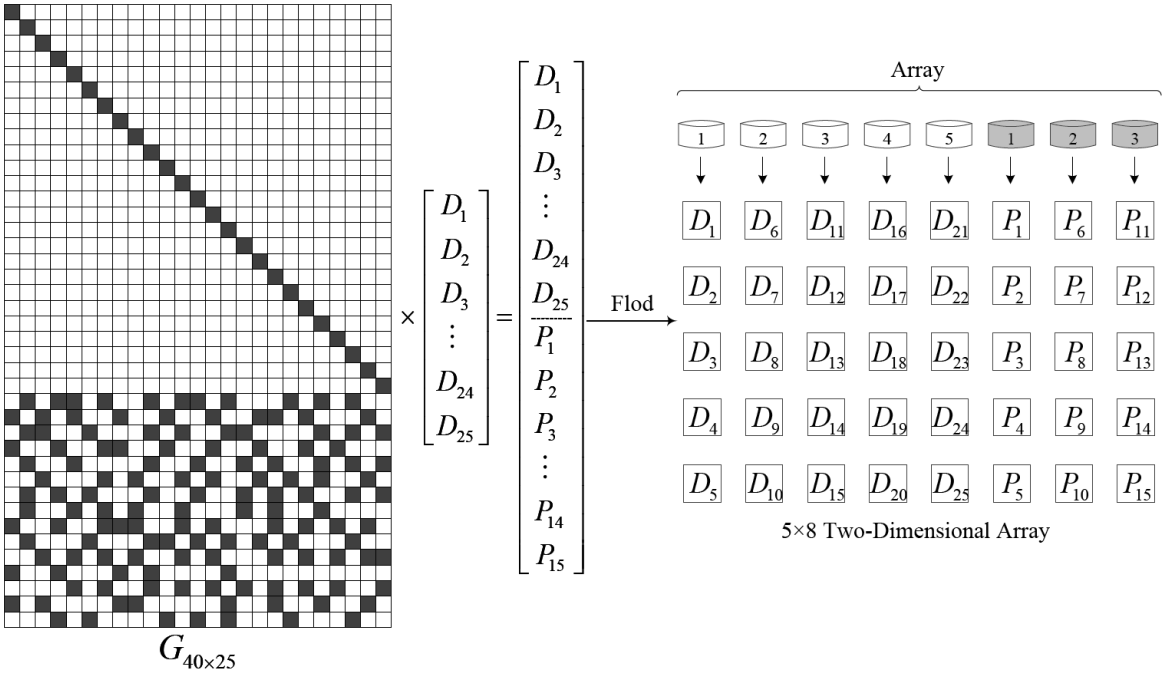


FIGURE 5. The two-dimensional data array layout

In practical applications, the RAID storage system data and parity disks can be randomly expanded according to actual requirements. RAID storage systems can be expanded by adding disks, and disks can be removed when they fail.

4.2. **The random expansion of data disks.** In this case, the number of data disks is dynamically adjusted to meet actual requirements with the growth of data volume. We assume that only one data disk is added or removed. The diagram in Figure 6 shows the general data/parity disks layout. We have two cases.

Removing: When one data disk is removed, the corresponding generator matrix is dynamically adjusted, and the data and parity disks need to be updated accordingly. Assume that the data disk numbered 2 is removed. Then, the 5 rows numbered 6 through 10, and the 5 columns numbered 6 through 10 in generator matrix $G_{40\times25}$ will be removed. The original generator matrix $G_{40\times25}$ will be converted into a newly generated matrix $\bar{G}_{35\times20}$ with a 20×20 identity matrix on the top and another 15×20 random submatrix on the bottom. The original data with 25 blocks $(D_1, D_2, D_3, \ldots, D_{24}, D_{25})$ as the message D needs to be cut into $\bar{D}$ with 20 blocks again. Meanwhile, message $\bar{D}$ needs to be stored

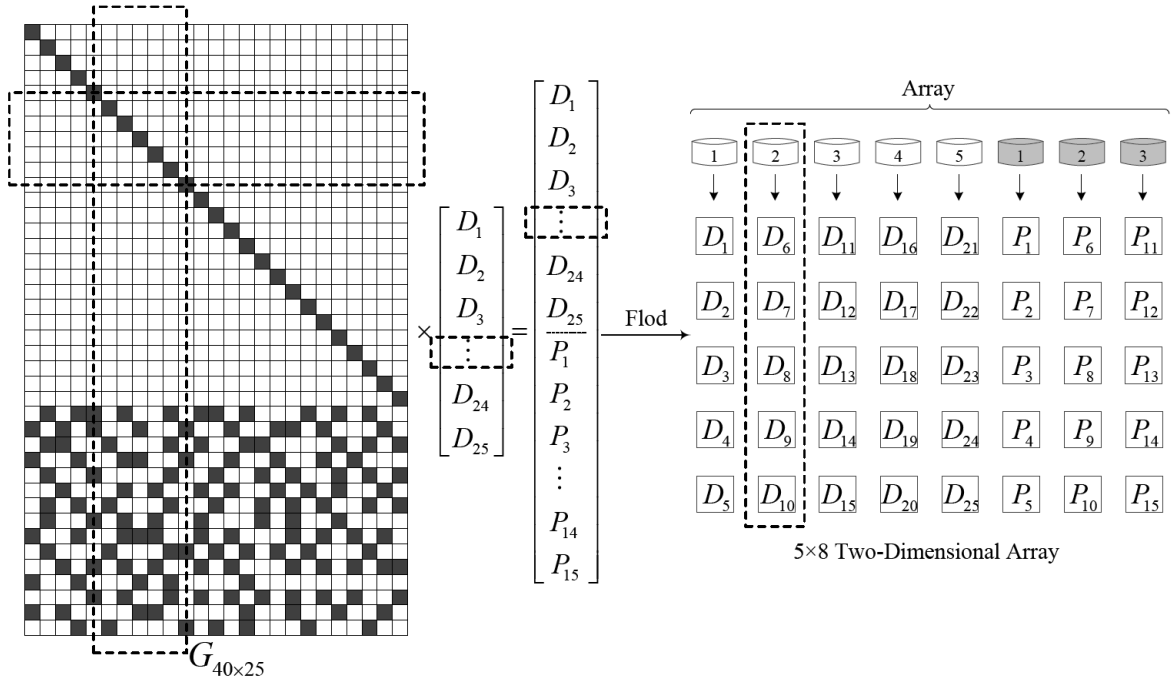


FIGURE 6. The expansion of data disks

in the remaining 4 data disks numbered 1, 3, 4, and 5. Finally, the parity disks numbered 1, 2, and 3 need to be updated accordingly to $\bar{D}$ and $\bar{G}_{35\times20}$ defined by $\bar{W} = G_{35\times20} \cdot \bar{D}$.

Adding: In contrast, when one data disk is added, the corresponding generator matrix is dynamically adjusted, and the data and parity disks need to be updated accordingly. Assume that the data disk numbered 2 is added. Then, the 5 rows numbered 6 through 10, and the 5 columns numbered 6 through 10 in generator matrix $\bar{G}_{35\times20}$ will be added. The original generator matrix $\bar{G}_{35\times20}$ will be converted into a newly generated matrix $G_{40\times25}$ with a 25×25 identity matrix on the top and another 15×25 random submatrix on the bottom. The original data with 20 blocks $(D_1, D_2, D_3, \ldots, D_{19}, D_{20})$ as the message $\bar{D}$ needs to be cut into D with 25 blocks again. Meanwhile, message D needs to be stored in the 5 data disks numbered 1 through 5. Finally, the parity disks numbered 1, 2, and 3 need to be updated accordingly to D and $G_{40\times25}$ defined by $W = G_{40\times25} \cdot D$.

4.3. **The random expansion of parity disks.** This case is similar to the random expansion of data disks. The number of parity disks is dynamically adjusted to provide reliability of customer data in RAID storage systems. We assume that only one parity disk is added or removed. The diagram in Figure 7 shows the general data/parity disk layout. We also have two cases:

Removing: Once one parity disk is removed, the corresponding generator matrix will be dynamically adjusted, but the data and parity disks do not need to be updated. Assume that in this example the parity disk numbered 3 is removed. Then, the 5 rows numbered 36 through 40 in generator matrix $G_{40\times25}$ will be removed. The original generator matrix $G_{40\times25}$ will be converted into a newly generated matrix $\bar{G}_{35\times25}$ with a 25×25 identity matrix on the top and another 10×25 random submatrix on the bottom.

Adding: In contrast, when one parity disk is added, the corresponding generator matrix will be dynamically adjusted, and only the parity disks need to be updated. Assume that in this example the parity disk numbered 3 is added. Then, the 5 rows numbered 36 through 40 in generator matrix $G_{35\times25}$ will be added. The original generator matrix $G_{35\times25}$ will be converted into a newly generated matrix $\bar{G}_{40\times25}$ with a 25×25 identity

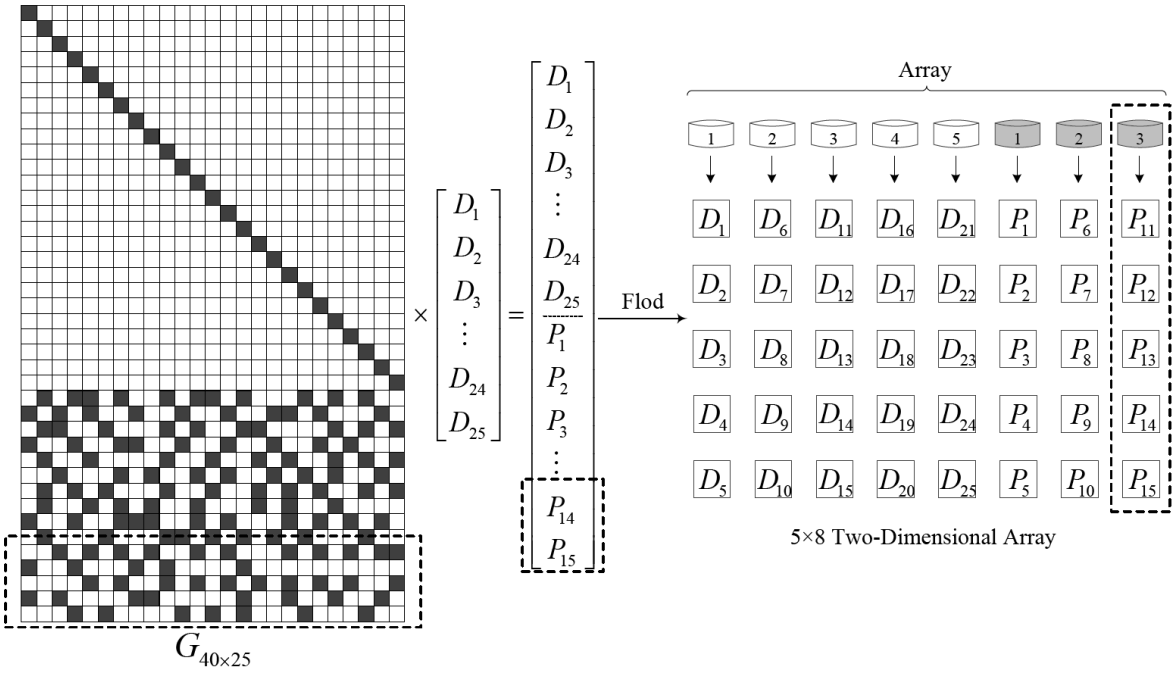


FIGURE 7. The expansion of parity disks

matrix on the top and another 15×25 random submatrix on the bottom. Meanwhile, the data disks numbered 1 through 5 do not need to be updated. The parity disks are computed independently but only parity disks numbered 3 need to be updated accordingly to D and $\bar{G}_{40\times25}$ defined by $\bar{W} = \bar{G}_{40\times25} \cdot D$.

5. **Performance and Implementation.** In this section, we summarize some primary features and the performances of our scheme, and then compare them with some other existing codes in terms of fault tolerance, storage efficiency computational complexity and expandability. Besides, the implementation of our scheme will also be included.

5.1. **Fault tolerance.** From Section 4.1 and previous works [14,30], we can easily find that the fault tolerance t of our scheme is approximately close to $n - k - 10$, where n is the total number of data and parity disks, and k is the number of data disks. This further shows that its fault tolerance can be dynamically adjusted according to actual requirements at the cost of only 10 redundant parity disks. Table 1 gives the fault tolerance of our scheme compared with other schemes. From this table, we can see that it can provide

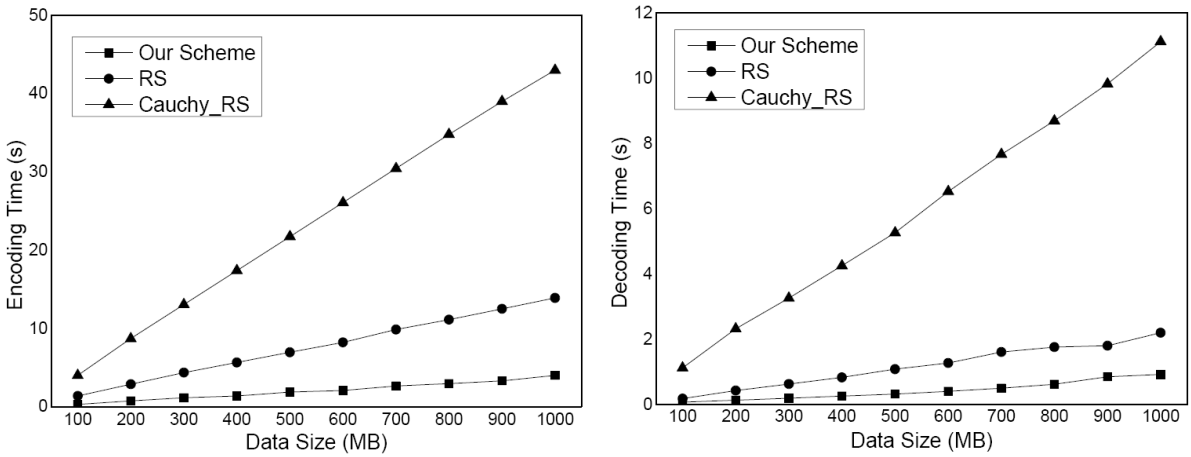
TABLE 1. Comparison of features with other schemes

Schemes	Fault Tolerance	Storage Efficiency	Computational Complexity	Expandability
RS	Arbitrary	Optimal	Galois field	Yes
EVENODD	2	Optimal	XOR	No
X-Code	2	Optimal	XOR	No
HoVer Code	4	Quasi-optimal	XOR	No
GRID	Up to 15 or even higher	Non-optimal	XOR	No
LDPC	Arbitrary	Quasi-optimal	XOR	No
Our Scheme	Arbitrary	Quasi-optimal	XOR	Yes

up to arbitrary fault tolerance. Especially when applied to the storage of big data which has dramatic increase in data volume, velocity, and variety, our scheme is particularly suitable for large-scale RAID storage systems in which the possibility of concurrent disk failures and multiple unrecoverable sector errors is noteworthy. In addition, our scheme is completely XOR-based code, and it is more efficient and economical to deploy our scheme to distribute data over multiple nodes in distributed storage systems than naive replications.

5.2. **Storage efficiency.** From the preceding discussions, we can see that our scheme can provide quasi-optimal storage efficiency, and that storage efficiency can reach up to $e = \frac{k}{n} = \frac{k}{k+t+10}$, where k is the number of data disks and t is the fault tolerance. It should be noted that the storage efficiency increases with the growth of data disks number k and can increase to a very high level; however, only 10 more redundant parity disks are additionally needed to be provided. It is clear that our scheme with higher fault tolerance always has lower storage efficiency than that with lower fault tolerance. This shows a trade-off between fault tolerance and storage efficiency. From what we have discussed in this subsection, we can see that our scheme can provide high storage efficiency, and in some scenarios their storage efficiency can reach up to quasi-MDS with the increase of data disks number k. This advantage can become very remarkable when they are applied to large-scale distributed storage systems.

5.3. **Computational complexity.** Our scheme is completely based on XOR operations over Galois field $GF(2)$ and does not need special-purpose hardware to enable efficient computation of encoding and decoding over the complex Galois field $GF(2^w)$. Table 1 gives the computational complexity of our scheme compared with other schemes. From this table, we can see that it is XOR-based scheme when compared with expandable method. The computational complexity of encoding and decoding directly depends on the number of 1s in generator matrix $G_{n\times k}$ and parity-check matrix $H_{k\times(n-k)}$. Thus, we can easily deduce that our scheme has an encoding complexity of $O(nk)$ and a decoding complexity of $O(k^3)$. From Figure 8, it can be found that our scheme has a significant advantage over the two kinds of widely used expandable methods, RS code [5] and Cauchy_RS code [15], in terms of computation time for encoding and decoding.



(a) (b)

FIGURE 8. The computation time for encoding and decoding of some existing schemes: (a) encoding time, (b) decoding time

5.4. **Expandability.** The main difference between our scheme and other existing approaches is that RAID storage system data and parity disks can be randomly expanded according to actual requirements. It is generally known that RS code and its generations are the unique expandable codes which also can provide optimal storage efficiency; however, they need special-purpose hardware to enable efficient computation of encoding and decoding over the complex Galois field $GF(2^w)$. On the contrary, our scheme is also expandable code and also can provide quasi-optimal storage efficiency without special-purpose hardware to accelerate computation. Compared with parity array codes based on XOR operations, our scheme has relatively regular geometric construction and can be randomly expanded with increased RAID storage systems data and parity disks in accordance with actual requirements. Furthermore, the original data and parity disks do not need to be completely updated when adding or removing some parity disks. With this capability, we can insert a new disk by hot plugging into an available slot while the RAID storage system is still running.

5.5. **Comparisons.** In this subsection, we will compare our scheme with other existing codes. Some of these codes are widely used in storage systems and communication fields.

RS: Both RS codes and our scheme can provide arbitrarily high fault tolerance and can be randomly expanded according to actual requirements. However, RS codes and their generations are based on Galois field arithmetic over $GF(2^w)$, which requires special-purpose hardware to enable efficient computation of the Galois field arithmetic on which the codes are based and generally has higher computation costs and complexities. And they are obviously not suitable for the large-scale RAID storage systems and distributed storage systems which have huge demands in data volume, velocity, and variety. Furthermore, our scheme is completely implemented based on simpler XOR operations instead of complicated Galois field arithmetic. Therefore, our scheme can have much better performance and easier implementation than traditional RS codes and their generations.

Parity Array Codes: Both parity array codes and our scheme are completely based on XOR operations. However, parity array codes have relatively irregular geometric construction and cannot be randomly expanded with the increased RAID storage system data and parity disks according to actual requirements. In our scheme, however, RAID storage system data and parity disks can be randomly expanded. Compared with EVENODD [6], X-Code [21], and generalized X-Code [19], which are MDS codes and are completely based on simple XOR operations, our scheme can provide arbitrarily higher fault tolerance. Compared with WEAVER [7] and GRID codes [24], which are completely based on simple XOR operations and can provide high fault tolerance, our scheme can provide expandability and also arbitrarily higher fault tolerance. Compared with FENG codes [8,9], which are MDS codes as the generalizations of RS codes [5] and can provide high fault tolerance, our scheme requires simpler operations and thus can have better performance. Thus, all these advantages make our scheme rather suitable for RAID storage systems that need flexible expandability.

New Codes: Newly invented codes are designed for some special cases in storage systems, which have no universality. The structures are too irregular to implement efficiently, and they are not well suited to RAID storage systems that need flexible expandability. However, our scheme has very regular structures and thus can be more easily implemented in RAID storage systems.

Table 1 compares our scheme with some other schemes in terms of fault tolerance, storage efficiency, computational complexity, and expandability. From this table, we can see that no matter which scheme we pick for the evaluation, our scheme has attractive advantages over it from some of the above-mentioned perspectives. Thus, it is worth

noting that our scheme is relatively suitable for RAID storage systems that need flexible expandability, arbitrary fault tolerance, and efficient computation costs. Furthermore, there will always be ample scope for our scheme's abilities in the field of large-scale distributed storage systems that greatly need flexible expandability.

5.6. **Implementation.** The implementation of our scheme is straightforward, and simply follows the procedure described in Section 4. Experiments are conducted to compare the encoding and decoding complexity of our proposed scheme with the only two existing kinds of expandable schemes: RS code [5] and Cauchy_RS code [15]. Both of them are widely used codes that can provide arbitrarily high fault tolerance without any restriction on the parameter settings. To make the comparison as fair as possible, we employ the Jerasure 2.0 package [38], an open source library that is a widely adopted and highly optimized software-based erasure coding implementation. The experiments are run on a single Intel Core i3 2.10-GHz machine with a 4GB memory running Linux Ubuntu 16.04.

For the RS code [5], Cauchy_RS code [15], and our scheme, we test all values of data block size from 100 MB to 1000 MB at intervals of 100 MB. The results are presented in Figure 8 which shows the computation time of data encoding and decoding using different schemes under different data block sizes. We plot the speed of encoding and decoding, which are measured as the amount of time per data block whose sizes are roughly from 100 MB to 1000 MB at intervals of 100 MB. For example, when the data block whose size is 500 MB, it takes an average of 1.85 seconds and 0.32 seconds to encode and decode this data block. Comparatively speaking, it takes about 6.94 seconds and 1.08 seconds for RS code, and 21.74 seconds and 5.26 seconds for Cauchy_RS code, respectively. It is clear that the proposed scheme still outperforms the two widely used expandable schemes in terms of computation time of data encoding and decoding by a significant margin.

6. **Conclusions.** In this article, we presented a novel data layout scheme using RBEC, which is designed to ensure random expandability, high reliability and availability of data in RAID storage systems. Compared with the existing approaches, our scheme has these attractive advantages: (1) it is completely implemented based on simple XOR operations and has systematic code properties ensuring easy implementations; (2) it can provide arbitrary fault tolerance only if providing 10 more redundant parity disks; (3) its storage efficiency is quasi-optimal with the growth of RAID storage system data disks, (4) RAID storage system data and parity disks can be randomly expanded according to the requirements of the practical systems. In addition, there is no any restriction on the parameters of our scheme. Our scheme provides RAID storage systems designers with good tradeoffs between fault tolerance and storage efficiency with continuous increase of RAID storage systems disks. All these advantages make our scheme particularly suitable for large-scale RAID storage systems that need higher reliability, availability, and expandability.

As detailed in article, at present we have only been able to implement our scheme. In the future, we will move to optimize the decoding algorithm by replacing the previous Gaussian elimination technique for better computational efficiency of speed. However, most of erasure codes are merely designed to tolerate the failures of entire disk at first. Thus, such a kind of erasure codes that can efficiently tolerate both disk and sector failures should be taken into consideration. It is also to be noted that besides the applications of our scheme in RAID storage systems, the distributed storage systems will extensively deploy erasure codes today for lower storage overhead instead of naive replication. Some attractive research directions of erasure codes in distributed storage systems will also be promoted, for example, generating code, data failure prediction, and updating efficiency.

Acknowledgments. The authors gratefully acknowledge the editor and anonymous reviewers for their insightful comments and suggestions on a previous version of this paper. This work was supported by the National Natural Science Foundation of China (No. 61501064) and the Sichuan Technology Support Program (No. 2015GZ0088).

REFERENCES

[1] D. A. Patterson, G. Gibson and R. H. Katz, A case for redundant arrays of inexpensive disks (RAID), *Proc. of the 1988 ACM SIGMOD International Conference on Management of Data*, pp.109-116, 1988.

[2] S. Ghemawat, H. Gobioff and S. T. Leung, The Google file system, *Proc. of the 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, USA, 2003.

[3] K. Shvachko, H. Kuang, S. Radia et al., The hadoop distributed file system, *Proc. of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies*, Incline Village, NV, USA, pp.1-10, 2010.

[4] G. DeCandia, D. Hastorun, M. Jampani et al., Dynamo: Amazon's highly available key-value store, *Proc. of the 21st ACM SIGOPS Symposium on Operating Systems Principles*, Stevenson, Washington, USA, vol.41, no.6, pp.205-220, 2007.

[5] I. S. Reed and G. Solomon, Polynomial codes over certain finite fields, *Journal of the Society for Industrial and Applied Mathematics*, vol.8, no.2, pp.300-304, 1960.

[6] M. Blaum, J. Brady, J. Bruck et al., EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures, *IEEE Trans. Computers*, vol.44, no.2, pp.192-202, 1995.

[7] J. L. Hafner, WEAVER codes: Highly fault tolerant erasure codes for storage systems, *Proc. of the 4th USENIX Conference on File and Storage Technologies*, pp.211-224, 2005.

[8] G. L. Feng, R. H. Deng, F. Bao et al., 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, 2005.

[9] G. L. Feng, R. H. Deng, F. Bao et al., New efficient MDS array codes for RAID part II: Rabin-like codes for tolerating multiple (≥ 4) disk failures, *IEEE Trans. Computers*, vol.54, no.12, pp.1473-1483, 2005.

[10] Y. X. Fu, J. W. Shu, X. H. Luo et al., Short code: An efficient RAID-6 MDS code for optimizing degraded reads and partial stripe writes, *IEEE Trans. Computers*, vol.66, no.1, pp.127-137, 2017.

[11] Z. J. Huang, H. Jiang, K. Zhou et al., XI-code: A family of practical lowest density MDS array codes of distance 4, *IEEE Trans. Communications*, vol.64, no.7, pp.2707-2718, 2016.

[12] Z. R. Shen, J. W. Shu and Y. X. Fu, HV code: An all-around MDS code for RAID-6 storage systems, *IEEE Trans. Parallel and Distributed Systems*, vol.27, no.6, pp.1674-1686, 2016.

[13] J. Yao, H. Jiang, Q. Cao et al., Elastic-RAID: A new architecture for improved availability of parity-based RAIDs by elastic mirroring, *IEEE Trans. Parallel and Distributed Systems*, vol.27, no.4, pp.1044-1056, 2016.

[14] L. Chen, J. Z. Zhang, P. G. Teng et al., Random binary extensive code (RBEC): An efficient code for distributed storage system, *Chinese Journal of Computers*, vol.40, no.9, pp.1980-1995, 2017.

[15] J. S. Plank and C. Huang, Tutorial: Erasure coding for storage systems, *Slides of the 11th USENIX Conference on File and Storage Technologies*, San Jose, CA, USA, 2013.

[16] R. M. Roth and A. Lempel, On MDS codes via cauchy matrices, *IEEE Trans. Information Theory*, vol.35, no.6, pp.1314-1319, 1989.

[17] P. Trifonov, Low-complexity implementation of RAID based on Reed-Solomon codes, *ACM Trans. Storage*, vol.11, no.1, 2015.

[18] P. Corbett, B. English, A. Goel et al., Row-diagonal parity for double disk failure correction, *Proc. of the 3rd USENIX Conference on File and Storage Technologies*, San Francisco, CA, USA, pp.1-14, 2004.

[19] X. H. Luo and J. W. Shu, Generalized X-code: An efficient RAID-6 code for arbitrary size of disk array, *ACM Trans. Storage*, vol.8, no.3, 2012.

[20] M. Q. Li and J. W. Shu, C-codes: Cyclic lowest-density MDS array codes constructed using starters for RAID 6, *Technical Report*, http://arxiv.org/abs/1104.2547, 2012.

[21] L. Xu and J. Bruck, X-code: MDS array codes with optimal encoding, *IEEE Trans. Information Theory*, vol.45, no.1, pp.272-276, 1999.

[22] C. Jin, H. Jiang, D. Feng et al., P-code: A new RAID-6 code with optimal properties, *Proc. of the 23rd International Conference on Supercomputing*, Yorktown Heights, NY, USA, pp.360-369, 2009.

[23] J. L. Hafner, Hover erasure codes for disk arrays, *Proc. of the International Conference on Dependable Systems and Networks*, Philadelphia, PA, USA, pp.217-226, 2006.
[24] M. Q. Li, J. W. Shu and W. M. Zheng, GRID codes: Strip-based erasure codes with high fault tolerance for storage systems, *ACM Trans. Storage*, vol.4, no.4, 2009.
[25] R. Gallager, Low-density parity-check codes, *IRE Trans. Information Theory*, vol.8, no.1, pp.21-28, 1962.
[26] S. Qi, D. Feng, N. Su et al., CDF-LDPC: A new error correction method for SSD to improve the read performance, *ACM Trans. Storage*, vol.13, no.1, 2017.
[27] A. G. Dimakis, P. B. Godfrey, Y. Wu et al., Network coding for distributed storage systems, *IEEE Trans. Information Theory*, vol.56, no.9, pp.4539-4551, 2010.
[28] J. S. Plank and M. Blaum, Sector-disk (SD) erasure codes for mixed failure modes in RAID systems, *ACM Trans. Storage*, vol.10, no.1, 2014.
[29] M. Q. Li and P. P. C. Lee, STAIR codes: A general family of erasure codes for tolerating device and sector failures, *ACM Trans. Storage*, vol.10, no.4, 2014.
[30] M. Y. Xia, M. Saxena, M. Blaum et al., A tale of two erasure codes in HDFS, *Proc. of the 13th USENIX Conference on File and Storage Technologies*, Santa Clara, CA, USA, pp.213-226, 2015.
[31] P. G. Teng, J. Z. Zhang, L. Chen et al., Random RAID: A RAID storage scheme with high fault-tolerance and flexibility, *Advanced Engineering Sciences*, vol.49, no.3, pp.110-116, 2017.
[32] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi et al., Practical loss-resilient codes, *Proc. of the 29th Annual ACM Symposium on Theory of Computing*, El Paso, TX, USA, pp.150-159, 1997.
[33] M. Luby, LT codes, *Proc. of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, Vancouver, BC, Canada, pp.271-280, 2002.
[34] A. Shokrollahi, Raptor codes, *IEEE Trans. Information Theory*, vol.52, no.6, pp.2551-2567, 2006.
[35] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, Elsevier, 1977.
[36] J. L. Hafner, V. Deenadhayalan, T. Kanungo et al., Performance metrics for erasure codes in storage systems, *IBM Research Report RJ 10321 (A0408-003)*, 2004.
[37] C. Cooper, On the rank of random matrices, *Random Structures & Algorithms*, vol.16, no.2, pp.209-232, 2000.
[38] J. S. Plank and K. M. Greenan, Jerasure 2.0: A library in C facilitating erasure coding for storage applications, *Technical Report*, UT-EECS-14-721, 2014.