Summer 2014

Ultimate Codes: Near-Optimal MDS Array Codes for RAID-6

Zhijie Huang
Huazhong University of Science and Technology, jayzy_huang@hust.edu.cn

Hong Jiang
University of Nebraska-Lincoln, jiang@cse.unl.edu

Chong Wang
Huazhong University of Science and Technology, C_Wang@hust.edu.cn

Ke Zhou
Huazhong University of Science and Technology, k.zhou@hust.edu.cn

Yuhong Zhao
Huazhong University of Science and Technology, yuhongzhao@hust.edu.cn

Ultimate Codes: Near-Optimal MDS Array Codes for RAID-6

Zhijie Huang, Hong Jiang, *Senior Member, IEEE,* Chong Wang,
Ke Zhou, *Member, IEEE,* and Yuhong Zhao

Abstract—As modern storage systems have grown in size and complexity, RAID-6 is poised to replace RAID-5 as the dominant form of RAID architectures due to its ability to protect against double disk failures. Many excellent erasure codes specially designed for RAID-6 have emerged in recent years. However, all of them have limitations. In this paper, we present a class of near perfect erasure codes for RAID-6, called the Ultimate codes. These codes encode, update and decode either optimally or nearly optimally, regardless of what the code length is. This implies that utilizing these codes we can build highly efficient and scalable RAID-6 systems. The performance analysis shows that the Ultimate codes outperform all the existing representative RAID-6 codes in encoding and decoding. Because of these unique advantages of the Ultimate codes, we anticipate them to become a preferred choice of the RAID-6 implementers.

Index Terms—RAID-6, MDS array codes, reliability, storage system

1 Introduction

Ever since it's 1988 introduction, the RAID (Redundant Arrays of Inexpensive Disks) technology [1] has been very widely adopted, almost ubiquitously in the applications needing highly available and reliable storage subsystems. The initial RAID organizations (RAID 1-5) can only protect against a single disk failure, which was shown to be inadequate as the average number of disks in a disk array grows [2]. Hence, the RAID-6 ($P + Q$ Redundancy) organization [3], which can tolerate any two concurrent disk failures and the case of encountering an uncorrectable read error during recovery, was subsequently proposed and added to the basic RAID organizations. Since modern storage systems are growing in size rapidly and use less reliable disks (e.g., ATA and SATA) widely to reduce the total cost in some applications, RAID-6 has been employed increasingly more pervasively to provide the necessary availability.

Unlike other RAID organizations, RAID-6 is merely a specification rather than an exact technique. It is a specification for disk arrays with k data disks plus 2 redundant disks to tolerate the failure of any two disks. The two redundant disks, which are called P and Q respectively, are used to store the coding information calculated from the original data. And the calculations must be made in such a way that if any two of the disks in the array fail, the data on the failed disks can be recovered from the surviving disks. Obviously, implementing a RAID-6 system relies on some erasure coding schemes, and the exact scheme to be adopted is up to the implementers. There are various erasure codes that can be used for implementing RAID-6, such as Reed-Solomon codes [4], EVENODD codes [5], RDP codes [6] and Liberation codes [7]. However, none of these codes has emerged as a clear "all-around" winner — each of them has its own limitations.

In this paper, we present a new class of MDS (Maximum Distance Separable) array codes that we argue are the best alternatives for implementing RAID-6 systems. These codes can provide optimal or near optimal performance in all phases of coding, namely, encoding, decoding and update, regardless of what the code length is. This is quite significant for building efficient and flexible RAID-6 systems. The main idea behind these codes can be conveyed by the three design principles that characterize these codes. First, they are designed so that the parity-check matrices are systematic and the density reaches the corresponding lower bound as presented in [8]. This implies that the update complexity of these codes attains the theoretical lower bound of systematic codes. Second, common subexpressions are identified and utilized to eliminate repetitive calculations in encoding/decoding. Third and finally, shortened codes are constructed such that the number of available common subexpressions is maximized. We provide a detailed description of the new codes, including encoding, update and decoding procedures in this paper. To protect against the data loss caused by silent errors, a simple decoding procedure for a single error is also presented. We will

- *Z. Huang, C. Wang, K. Zhou and Y. Zhao are with the Wuhan National Laboratory for Optoelectronics, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China.*
  E-mail: {jayzy_huang, C_Wang, k.zhou, yuhongzhao}@hust.edu.cn.
- *H. Jiang is with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588-0150.*
  E-mail: jiang@cse.unl.edu.

show that these codes represent the state-of-the-art RAID-6 codes and we conjecture that they may not be further improved. As such, we anticipate that they will likely become a preferred choice of the RAID-6 implementers.

The rest of this paper is organized as follows. In the next section we briefly introduce the RAID-6 specification and review the strengths and weaknesses of the representative existing codes for RAID-6, which serves as the key motivation for this work. Then, in Section 3, we describe the construction of our new codes and prove their MDS property. Section 4 presents the concrete encoding and decoding algorithms. Section 5 describes the construction of the best shortened codes and Section 6 discusses the significance of the lowest density property. The complexity analysis and comparison with the previous works are presented in Section 7. Finally, in Section 8, we summarize the contributions of this work and remark on the directions of our future work.

2 Background and Motivation

In a broader sense, the term "RAID-6" represents any form of RAID that can tolerate any two concurrent disk failures [9]. Nevertheless, the *de facto* standard form of RAID-6 has been the so-called $P + Q$ redundancy, which was first proposed in [3]. The $P + Q$ redundancy RAID-6 is extended from RAID-5 by adding another redundant disk to the original system. Consequently, the data striping and parity placement of RAID-6 are similar to that of RAID-5, as depicted in Fig. 1.

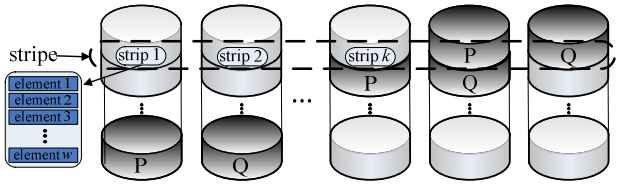

Fig. 1. Data striping and parity placement of RAID-6

Specifically, each disk of a RAID-6 system is partitioned into a number of commensurate segments, called *strips*. Then, a *stripe* is defined as a maximal set of strips that are dependently related by an erasure code. Each disk of the array contributes exactly one strip to a certain stripe, as shown in Fig. 1. In each stripe, the coding strip P is obtained by XORing all the data strips, while the calculation of the coding strip Q depends on the erasure coding scheme used to implement RAID-6. Thus, the key challenge in designing a RAID-6 coding scheme lies in the definition of the Q strip. It is worth mentioning that the terms "strip" and "stripe" correspond to the coding theory terms "symbol" and "codeword" respectively.

There are many erasure codes that can be used to implement RAID-6, and the most popular ones all conform to the $P + Q$ redundancy form. The original implementation of RAID-6 [3] used Reed-Solomon codes that are *general-purpose* erasure codes capable of correcting arbitrary numbers of erasures and errors. However, Reed-Solomon codes require specialized hardware to enable efficient computation of the finite field arithmetic on which the codes are based. Consequently, in order to implement RAID-6 efficiently, a series of *MDS array codes* that involve only XOR (exclusive-OR) computations were proposed in the past few years.

In MDS array codes, each codeword is a $w \times n$ array of binary bits, containing both data bits and parity bits, where w is usually a function of a prime number. Each parity bit is calculated to be the even parity of a certain collection of the data bits, and the calculations must be such that any two erasures can be tolerated without data loss. According to the distribution of the parity information in a codeword, MDS array codes can be categorized into *horizontal codes* such as Cauchy Reed-Solomon Codes [10], EVENODD codes [5], RDP codes [6], and Liberation codes [7], and *vertical codes* such as X-code [11], B-code [12], P-code [13] and Cyclic Lowest Density codes [14]. In horizontal codes, the parity information is placed in dedicated columns, while in vertical codes it is evenly distributed among almost all the columns. Thus, vertical codes are quite inflexible in that they can not change the number of disks in the system dynamically without recoding the entire system. Obviously, only horizontal codes are applicable to the $P + Q$ redundancy form of RAID-6.

There are also non-MDS codes that tolerate two-disk failures, such as WEAVER Codes [15], HoVer codes [16], full-2 codes [17] and generalized X-Code [18]. These codes are less popular due to their suboptimal storage efficiency and practitioners typically do not want to devote extra storage for fault tolerance when they do not have to [19].

When implementing a RAID-6 system with MDS array codes, each strip is further divided into a certain number of *elements*, of which each generally consists of one or more machine words. Now, a stripe appears as a two-dimensional array of elements. Actually, a stripe is exactly a certain number of codewords that are interleaved for efficient computing [20]. For instance, if an element is defined to be a 32-bit machine word, then a stripe consists of 32 interleaved codewords. In this way, the XORs are performed on machine words rather than bits, improving the efficiency. Thus, the element size is restricted to be a multiple of the machine's word size. In the most common case, an element is implemented as a whole sector, which is the smallest unit of disk access.

In general, erasure codes in storage systems can be evaluated in the following metrics:

- *Storage overhead* refers to the redundancy needed to provide certain fault tolerance. Theoretically, in order to protect the system against the loss of

any r disks, we need at least r redundant disks. In coding theory, this is known as the *Singleton bound* [21], and the codes that attain this bound are classified as the *Maximum Distance Separable (MDS)* codes.
- *Encoding complexity* refers to the number of computational operations (usually translated to XORs) needed to calculate the coding information. For a $(k+2)$-system, the theoretical lower bound of the encoding complexity is $k-1$ XOR operations per coding element [6].
- *Decoding complexity* refers to the number of computational operations needed to reconstruct the lost data or coding information. As with encoding, the theoretical lower bound is $k-1$ XOR operations per lost element for a $(k+2)$-system.
- *Update complexity* refers to the average number of coding elements that must be updated when a data element is modified. The theoretical lower bound is 2 for two-erasure-correcting codes.
- *Flexibility* signifies the ease of altering the number of disks of the entire system.

All of the above metrics are important in the practical systems. In particular, storage overhead directly translates into the cost of fault tolerance, thus MDS codes are preferred. Encoding complexity represents the performance of full-stripe writes, while update complexity directly affects the performance of small writes. Moreover, when applied to SSD (Solid State Drive) arrays, update complexity can also affect the SSDs' service lifetime. Decoding complexity determines the performance of recovery and degraded reads. Note that the reconstruction time is inversely proportional to the system's availability. Finally, as the storage systems often need to expand their capacity according to new requirements after initial deployment, it is desirable to have flexible RAID-6 systems that allow easy expansion.

To the best of our knowledge, EVENODD, RDP and Liberation codes represent the current state-of-the-art RAID-6 codes. However, they still have some limitations that cannot be ignored. In particular, EVENODD and RDP codes all have the update complexity of nearly 3, which is 1.5 times over the theoretical lower bound of 2. Moreover, for the flexibility of altering the number of disks, the parameter w is usually fixed after initial deployment. In this case, EVENODD and RDP codes will suffer from performance loss during encoding and decoding when k shrinks [7]. On the other hand, while Liberation codes attain the lower bound of horizontal codes in update complexity and exhibit near optimal encoding performance, they have notably worse decoding performance than the former two. As a result, it would clearly be desirable to have codes with best trade-offs among all these metrics. To this end, we present a new class of MDS array codes for RAID-6, called the Ultimate codes, with the following properties:

- They conform to the $P+Q$ redundancy form.
- They encode and decode optimally or near optimally for any size of RAID-6 systems.
- Their update complexity is asymptotically optimal as $w \to \infty$. Moreover, it can be shown that it achieves the lower bound for all systematic/horizontal MDS array codes with $w = \text{prime} - 1$.
- For a fixed w, they always exhibit optimal or near optimal performance as k varies from 2 to $w+1$. As we will see in Section 7, this property allows us to build flexible RAID-6 systems almost without performance loss.

We describe the new codes and analyze their complexity next.

3 Construction of Ultimate Codes

Since our codes conform to the $P+Q$ redundancy specification, the sole task left is to define the calculation of the Q strip. As mentioned before, a stripe is essentially a group of codewords that are interleaved for performance purpose. Thus, for simplicity, we will focus on codewords rather than stripes in what follows.

In Ultimate codes, a standard codeword is a $(m-1) \times (m+2)$ bit array with the first m columns containing data bits while the last two columns containing parity bits, where m is an odd prime. This requirement is necessary for the codes to be MDS. However, this does not mean that the number of data disks in a real RAID-6 system must be a prime number too. If we want to build a RAID-6 system with any number k data disks, we can take a prime number m such that $m \geq k$ and assume that there are $m-k$ empty (or imaginary) disks (disks storing only 0s) in the system. In this way, the codes actually can be used to implement a RAID-6 system with an arbitrary size.

Before formally describing the encoding procedure, we first define some notations. We use $d_{i,j}$ to denote the i-th bit in the j-th data column, and p_i, q_i to denote the i-th bit in the parity columns P and Q respectively, where $0 \leq i \leq m-2$ and $0 \leq j \leq m-1$. For ease of describing the code, we assume throughout this paper that there is an imaginary all-zero row after the last row in the codeword. We also let $\langle a \rangle = a \bmod m$, clearly $0 \leq \langle a \rangle \leq m-1$.

3.1 The Encoding Procedure

In any codeword, each parity bit is calculated to be the even parity of a certain subset of the data bits. In order to tolerate two-disk failures, the calculations must be such that the bits contained in any two columns of the codeword can be reconstructed from the other m columns. Using the notations defined above, the

relations between the parity bits and the data bits in a codeword can be described as follows:

$$p_i = \bigoplus_{c=0}^{m-1} d_{i,c} \quad (1)$$

$$q_i = \left(\bigoplus_{c=0}^{m-1} d_{\langle i-c\rangle,c}\right) \oplus d_{m-2-i,i+1} \oplus d_{m-1-\langle 2i+2\rangle,\langle 2i+2\rangle} \quad (2)$$

where $i = 0, 1, \cdots, m-2$.

Obviously, column P is simply the XOR of all the data columns, i.e., the i-th bit of column P is just the even parity of all the data bits in the i-th row. The definition of column Q, however, is somewhat complicated. To make it more intuitive, let us consider the geometric meaning of (2), noting that the $(m-1)$-th (last) row of the codeword is an imaginary all-zero row. First, we define the i-th diagonal as the position set $\{(x,y)|x+y \equiv i(\mathrm{mod}\, m), 0 \le x, y \le m-1\}$, i.e., $\{(i,0), (\langle i-1\rangle, 1), (\langle i-2\rangle, 2), \cdots, (\langle i+1\rangle, m-1)\}$. Clearly, the data part of the codeword can be completely covered by m such diagonals. Then, q_i is calculated to be the XOR of all the data bits along the i-th diagonal and two additional data bits along the $(m-1)$-th diagonal. These two additional data bits are in the $(i+1)$-th column and the $\langle 2i+2\rangle$-th column respectively. We define the i-th diagonal parity group as the set that consists of q_i and the data bits used to calculate q_i. Since every diagonal parity group contains exactly two data bits along the $(m-1)$-th diagonal, we call this diagonal the *shared diagonal*. As an example, Fig. 2 shows the diagonal parity groups and the shared diagonal of the standard Ultimate code with $m = 5$. The i-th diagonal parity group is labeled with the number i.

D0	D1	D2	D3	D4	P	Q
0	1	2	3	**31**		0
1	2	3	**23**	0		1
2	3	**10**	0	1		2
3	**02**	0	1	2		3
	0	1	2	3		

Fig. 2. Diagonal parity groups in a standard Ultimate code with $m = 5$

The notion of the shared diagonal – each diagonal parity group contains exactly two of the data bits lying in the $(m-1)$-th diagonal – is the main novelty of the Ultimate codes. And this is the only difference between the Ultimate codes and the EVENODD codes in which each diagonal parity group contains all the data bits lying in the $(m-1)$-th diagonal. However, as we will see below, it is this difference that enables the former to be optimal or near optimal in all the metrics of erasure codes.

3.2 The MDS Property

The Ultimate codes defined above can recover the information lost in any two columns, provided that m is an odd prime. In other words, the minimum distance of the code is 3, in the sense that any nonzero codeword in the code has at least 3 columns that are nonzero. Next, we prove the MDS property of the Ultimate codes by showing that any nonzero codeword has (column) weight exactly 3.

First we present some lemmas that will be used in the proof of Theorem 1 that claims Ultimate codes' MDS property.

Lemma 1: In the sequence of numbers $\{\langle m-1+n\delta\rangle, n = 0 \ldots m\}$, with m being prime and $0 < \delta < m$, the endpoints are both equal to $m-1$, and all numbers $0, 1, \cdots, m-2$ occur exactly once in the sequence [6].

Lemma 2: For any odd prime m, if $\frac{m+1}{2} \le x \le m-1$, then $\langle 2x\rangle < x$.

Proof: Since $m+1 \le 2x \le 2m-2$, we have $\langle 2x\rangle = 2x - m = x - (m-x) < x$. □

Lemma 3: For $1 \le j < l \le m-1$(m is an odd prime), if $l = 2j$ and $j = \langle 2l\rangle$, then $m = 3$.

Proof: From $l = 2j$ and $j = \langle 2l\rangle$, we have $j \equiv 4j(\mathrm{mod}\, m)$, i.e., $3j \equiv 0(\mathrm{mod}\, m)$. Since m is a prime number and $1 \le j \le m-1$, we have $3 \equiv 0(\mathrm{mod}\, m)$, thus $m = 3$. □

Lemma 4: For any odd prime m, if $1 \le x \le m-1$, then $0 \le \langle\langle (x-2)/2\rangle - x\rangle \le m-2$.

Proof: Since $0 \le \langle\langle (x-2)/2\rangle - x\rangle \le m-1$, we only need to prove that $\langle\langle (x-2)/2\rangle - x\rangle \ne m-1$. Let us prove it by contradiction. Suppose $\langle\langle (x-2)/2\rangle - x\rangle = m-1$, i.e., $(x-2)/2 - x \equiv m-1(\mathrm{mod}\, m)$, then we have $x - 2 - 2x \equiv 2m-2(\mathrm{mod}\, m)$, $-x \equiv 2m(\mathrm{mod}\, m)$, $x \equiv 0(\mathrm{mod}\, m)$. This contradicts the condition $1 \le x \le m-1$. □

Lemma 5: For any odd prime m, if $1 \le x, y \le m-1$ and $x \ne \langle 2y\rangle$, then $0 \le \langle\langle (x-2)/2\rangle - y\rangle \le m-2$.

Proof: Since $0 \le \langle\langle \frac{x-2}{2}\rangle - y\rangle \le m-1$, we only need to prove that $\langle\langle \frac{x-2}{2}\rangle - y\rangle \ne m-1$. Let us prove it by contradiction. Suppose $\langle\langle \frac{x-2}{2}\rangle - y\rangle = m-1$, i.e., $\frac{x-2}{2} - y \equiv m-1(\mathrm{mod}\, m)$, then we have $x - 2 - 2y \equiv 2m-2(\mathrm{mod}\, m)$, $x \equiv 2y + 2m(\mathrm{mod}\, m)$, $x \equiv 2y(\mathrm{mod}\, m)$. Since $1 \le x, y \le m-1$, $x \equiv 2y(\mathrm{mod}\, m)$ is equivalent to $x = \langle 2y\rangle$. This contradicts the condition $x \ne \langle 2y\rangle$. □

Lemma 6: For $1 \le x \le m-1$(m is an odd prime), if $\langle 2x\rangle$ is odd, then $\langle 2x\rangle < x$, otherwise $\langle 2x\rangle > x$.

Proof: Since $2 \le 2x \le 2m-2$, then either $\langle 2x\rangle = 2x$ or $\langle 2x\rangle = 2x - m$. If $\langle 2x\rangle$ is odd, necessarily, $\langle 2x\rangle = 2x - m = x - (m-x) < x$. If $\langle 2x\rangle$ is even, necessarily, $\langle 2x\rangle = 2x > x$. □

Theorem 1: For any odd prime m, the code defined above has column distance of 3, i.e., it is MDS.

Proof: Observe that the code is a linear code, thus its column distance is equal to its *minimum column weight*, i.e., the column weight of the nonzero codeword with the smallest column weight. Therefore, proving that the code is MDS is equivalent to proving

that any valid nonzero codeword of the code has at least three nonzero columns. We will prove it by contradiction.

From the construction of the code, it is very easy to verify that it is impossible for a nonzero codeword to have only one nonzero column.

Now, suppose there is a nonzero codeword that contains two nonzero columns. The positions of the two nonzero columns fall into the following four cases:

Case 1: The two nonzero columns are P and Q. This is impossible according to (1) and (2).

Case 2: The jth data column ($0 \le j \le m-1$) and column Q are nonzero. In this case, column P is equal to the jth data column according to (1), i.e., column P is also nonzero. This contradicts the assumption.

Case 3: The jth data column ($0 \le j \le m-1$) and column P are nonzero. For conciseness, we let $u = \langle \frac{j-2}{2} \rangle$ throughout this paper. According to (2), we have $d_{\langle i-j\rangle,j} \oplus d_{m-2-i,i+1} \oplus d_{m-1-\langle 2i+2\rangle,\langle 2i+2\rangle} = 0$ for $i = 0, 1, \cdots, m-2$. Since $0 \le i \le m-2$, it can be easily deduced that $1 \le i+1, \langle 2i+2\rangle \le m-1$. Then, if $j = 0$, we have $d_{m-2-i,i+1} = d_{m-1-\langle 2i+2\rangle,\langle 2i+2\rangle} = 0$, thus $d_{i,0} = 0$, which contradicts the assumption. If $1 \le j \le m-1$, then we get

$$\begin{cases} d_{m-1,j} \oplus d_{m-1-j,j} = 0 \\ d_{\langle u-j\rangle,j} \oplus d_{m-1-j,j} = 0 \\ d_{\langle i-j\rangle,j} = 0, \quad i \ne j-1, u \end{cases} \tag{3}$$

From this equation we can deduce that every data bit in the jth data column is 0, which contradicts the assumption again.

Case 4: The jth data column and the lth data column ($0 \le j < l \le m-1$) are nonzero. This is the main case. For conciseness, we let $v = \langle \frac{l-2}{2} \rangle$ throughout this paper. Moreover, we use S_i^P (S_i^Q) to denote the XOR of all the bits in the i-th row (diagonal) parity group except the ones that are in the j-th and l-th data columns. First, according to (1), we have

$$d_{i,j} \oplus d_{i,l} = S_i^P = 0, \quad 0 \le i \le m-2. \tag{4}$$

Then, if $j = 0$ and $1 \le l \le m-1$, according to (2), we have

$$\begin{cases} d_{l-1,0} \oplus d_{m-1-l,l} = S_{l-1}^Q = 0 \\ d_{v,0} \oplus d_{\langle v-l\rangle,l} \oplus d_{m-1-l,l} = S_v^Q = 0 \\ d_{i,0} \oplus d_{\langle i-l\rangle,l} = S_i^Q = 0 (i \ne l-1, v) \end{cases}. \tag{5}$$

According to Lemma 1, v exists in the sequence of numbers $m-1$,$\langle m-1+l\rangle$,$\langle m-1+2l\rangle$,$\cdots$,$m-1-l$,$m-1$. Thus, there must be two numbers n_1 and n_2 such that $\langle v - n_1 l\rangle = \langle m-1+l\rangle = l-1$ and $\langle v + n_2 l\rangle = m-1-l$. From (4) and (5), we have the following set of equations:

$$\begin{aligned} d_{v,0} \oplus d_{v,l} &= 0 \\ d_{v,l} \oplus d_{\langle v+l\rangle,0} &= 0 \\ d_{\langle v+l\rangle,0} \oplus d_{\langle v+l\rangle,l} &= 0 \\ d_{\langle v+l\rangle,l} \oplus d_{\langle v+2l\rangle,0} &= 0 \\ &\vdots \\ d_{m-1-l,0} \oplus d_{m-1-l,l} &= 0 \end{aligned}$$

Adding these equations results in a new equation $d_{v,0} \oplus d_{m-1-l,l} = 0$. From this equation and (6) we get $d_{\langle v-l\rangle,l} = 0$. Once $d_{\langle v-l\rangle,l}$ is determined, according to (4) and (5), we have $d_{i,0} = d_{i,l} = 0 (i = \langle v-l\rangle \dots l-1)$. From $d_{l-1,0} = 0$ and Equation (5) we get $d_{m-1-l,l} = 0$. Similarly, once $d_{m-1-l,l}$ is determined, we can deduce that $d_{i,0} = d_{i,l} = 0 (i = v \dots m-1-l)$. Notice that $\langle v-l\rangle$ and v are adjacent points in the sequence mentioned above, thus all the data bits in the 0th data column and the lth data column are zeros. This contradicts the assumption.

If $1 \le j < l \le m-1$, then according to the relationship between j and l, this case can be further divided into four subcases:

(a) $l = \langle 2j\rangle$ and $j = \langle 2l\rangle$. When $\langle 2j\rangle = l$, since $j < l$, according to Lemma 2, we have $1 \le j \le \frac{m-1}{2}$, thus $l = \langle 2j\rangle = 2j$. According to Lemma 3, this subcase occurs only when $m = 3$, thus $j = 1$ and $l = 2$. Then, from (2) we have $d_{1,2} \oplus d_{1,1} \oplus d_{0,2} = 0$ and $d_{0,1} \oplus d_{0,2} \oplus d_{1,1} = 0$. From these two equations and (4), it can be easily deduced that $d_{0,2} = d_{0,1} = d_{1,1} = d_{1,2} = 0$, which contradicts the assumption.

(b) $l = \langle 2j\rangle$ and $j \ne \langle 2l\rangle$. In this subcase there is no i such that $i+1 = l$ and $\langle 2i+2\rangle = j$. Moreover, when $i+1 = j$, we have $\langle 2i+2\rangle = \langle 2j\rangle = l$ and $\langle i-l\rangle = \langle j-1-l\rangle = \langle j-1-2j\rangle = \langle -1-j\rangle = m-1-j$. Therefore, from (2) we have

$$\begin{cases} d_{m-1-j,l} \oplus d_{m-1-j,j} \oplus d_{m-1-l,l} = S_{j-1}^Q = 0 \\ d_{\langle u-j\rangle,j} \oplus d_{\langle u-l\rangle,l} \oplus d_{m-1-j,j} = S_u^Q = 0 \\ d_{\langle l-1-j\rangle,j} \oplus d_{m-1-l,l} = S_{l-1}^Q = 0 \\ d_{\langle i-j\rangle,j} \oplus d_{\langle i-l\rangle,l} = S_i^Q = 0, \\ i \ne j-1, u, l-1 \end{cases}. \tag{6}$$

From (4) and (6), we can first get $d_{m-1-l,l} = 0$ and $d_{\langle l-1-j\rangle,j} = 0$. According to Lemma 1, all numbers $0 \dots m-2$ occur exactly once in the sequence of numbers $m-1, \langle m-1+(l-j)\rangle, \langle m-1+2(l-j)\rangle, \cdots, m-1-(l-j), m-1$. Moreover, according to Lemma 4 and Lemma 5, we have $0 \le \langle u-j\rangle, \langle u-l\rangle \le m-2$, i.e., they are in the sequence. Then, from $d_{m-1-l,l} = 0$ we can deduce that $d_{i,j} = d_{i,l} = 0 (i = m-1-l, m-1-l-(l-j) \dots \langle u-j\rangle)$ by using Equations (4) and (6) alternately. Similarly, from $d_{\langle l-1-j\rangle,j} = 0$ we can deduce that $d_{i,j} = d_{i,l} = 0 (i = \langle m-1+(l-j)\rangle, \langle m-1+2(l-j)\rangle \dots \langle u-l\rangle)$. Note that $m-1-l = m-1-2j = m-1-2(l-j)$ and $\langle l-1-j\rangle = \langle m-1+(l-j)\rangle$. Once $d_{\langle u-j\rangle,j}$ and $d_{\langle u-l\rangle,l}$ are determined, according to Equation (6)

we get $d_{m-1-j,j}=0$, thus $d_{m-1-j,l}=0$. Notice that $m-1-j=m-1-(l-j)$, and that $\langle u-l\rangle$ and $\langle u-j\rangle$ are adjacent points in the sequence, thus we have $d_{i,j}=d_{i,l}=0(0\le i\le m-2)$. This contradicts the assumption.

(c) $l\neq\langle 2j\rangle$ and $j=\langle 2l\rangle$. This subcase is similar to Subcase (b).

(d) $l\neq\langle 2j\rangle$ and $j\neq\langle 2l\rangle$. This is the main subcase. In this subcase, from (2) we have

$$\begin{cases} d_{\langle j-1-l\rangle,l}\oplus d_{m-1-j,j}=S_{j-1}^{Q}=0\\ d_{\langle u-j\rangle,j}\oplus d_{\langle u-l\rangle,l}\oplus d_{m-1-j,j}=S_{u}^{Q}=0\\ d_{\langle l-1-j\rangle,j}\oplus d_{m-1-l,l}=S_{l-1}^{Q}=0\\ d_{\langle v-j\rangle,j}\oplus d_{\langle v-l\rangle,l}\oplus d_{m-1-l,l}=S_{v}^{Q}=0\\ d_{\langle i-j\rangle,j}\oplus d_{\langle i-l\rangle,l}=S_{i}^{Q}=0,\\ i\neq j-1,u,l-1,v \end{cases}. \quad (7)$$

According to Lemma 4 and Lemma 5, we have $0\le\langle u-j\rangle,\langle u-l\rangle,\langle v-j\rangle,\langle v-l\rangle\le m-2$, thus they are in the sequence $\{\langle m-1+n(l-j)\rangle,n=0\dots m\}$. Therefore, there must be $1\le x,y,z\le m-2$ such that $\langle v-l\rangle=\langle m-1+x(l-j)\rangle$, $\langle u-l\rangle=\langle m-1+y(l-j)\rangle$ and $m-1-l=\langle m-1+z(l-j)\rangle$. From $\langle v-l\rangle=\langle m-1+x(l-j)\rangle$, we have $-l\equiv 2x(l-j)(\mathrm{mod}\,m)$. Moreover, from $m-1-l=\langle m-1+z(l-j)\rangle$, we have $-l\equiv z(l-j)(\mathrm{mod}\,m)$, thus $z(l-j)\equiv 2x(l-j)(\mathrm{mod}\,m)$, i.e., $(z-2x)(l-j)\equiv 0(\mathrm{mod}\,m)$. Since m is a prime number and $1\le l-j\le m-2$, we have $(z-2x)\equiv 0(\mathrm{mod}\,m)$, i.e., $z=\langle 2x\rangle$. Similarly, from $\langle u-l\rangle=\langle m-1+y(l-j)\rangle$, we have $\frac{j-2}{2}-l\equiv -1+y(l-j)(\mathrm{mod}\,m)$, thus $-j\equiv 2(y+1)(l-j)(\mathrm{mod}\,m)$. In addition, since $-l\equiv z(l-j)(\mathrm{mod}\,m)$ is equivalent to $-j\equiv(z+1)(l-j)(\mathrm{mod}\,m)$, we have $(z+1)(l-j)\equiv 2(y+1)(l-j)(\mathrm{mod}\,m)$, i.e., $z+1=\langle 2(y+1)\rangle$. According to Lemma 6, if z is odd, we have $z<x$ and $z+1>y+1$, i.e., $x\ge z+1$ and $y+1\le z$; otherwise, we have $z>x$ and $z+1<y+1$, i.e., $x+1\le z$ and $z+1\le y$.

Then, if z is odd, adding equations $d_{i,j}\oplus d_{i,l}=0(i=m-1-j,\langle m-1+(z+2)(l-j)\rangle\dots\langle m-1-(l-j)\rangle)$, $d_{\langle i-j\rangle,j}\oplus d_{\langle i-l\rangle,l}=0(i=l-j-1,\langle m-1+2(l-j)\rangle\dots\langle j-1-(l-j)\rangle\,\mathrm{and}\,i\neq v)$ and $d_{\langle j-1-l\rangle,l}\oplus d_{m-1-j,j}=0$ results in a new equation $d_{\langle v-j\rangle,j}\oplus d_{\langle v-l\rangle,l}=0$. From this equation and (7), we get $d_{m-1-l,l}=0$. Once $d_{m-1-l,l}$ is determined, we can easily deduce that $d_{i,j}=d_{i,l}=0(i=\langle m-1+(l-j)\rangle,\langle m-1+2(l-j)\rangle\dots m-1-l)$ by using (4) and (7) alternately. Note that $y+1\le z$, thus $d_{\langle u-j\rangle,j}=d_{\langle u-l\rangle,l}=0$. From this equation and Equation (7), we get $d_{m-1-j,j}=0$. Similarly, once $d_{m-1-j,j}$ is determined, we can easily deduce that $d_{i,j}=d_{i,l}=0(i=m-1-j\dots\langle m-1-(l-j)\rangle)$ by using (4) and (7) alternately. From the above we have $d_{i,j}=d_{i,l}=0(0\le i\le m-2)$, which contradicts the assumption.

If z is even, let us first consider the data bits $d_{i,j}$ and $d_{i,l}$, where i is in the subsequence $\{\langle m-1+n(l-j)\rangle,n=1\dots m-1-l\}$. If $x+1=z$, then $\langle v-j\rangle=m-1-l$, thus, from (4) and (7), we get $d_{\langle v-l\rangle,l}=0$. Otherwise, adding equations $d_{i,j}\oplus d_{i,l}=0(i=\langle m-1+(x+1)(l-j)\rangle\dots m-1-l)$ and $d_{\langle i-j\rangle,j}\oplus d_{\langle i-l\rangle,l}=0(i=\langle v+l-j\rangle\dots m-1-(l-j))$ results in a new equation $d_{\langle v-j\rangle,j}\oplus d_{m-1-l,l}=0$. From this equation and (7), we get $d_{\langle v-l\rangle,l}=0$. Once $d_{\langle v-l\rangle,l}$ is determined, we can easily deduce that $d_{i,j}=d_{i,l}=0(i=\langle m-1+(l-j)\rangle,\langle m-1+2(l-j)\rangle\dots m-1-l)$ by using (4) and (7) alternately. Similarly, from $z+1\le y$, we can deduce that $d_{i,j}=d_{i,l}=0(i=m-1-j,\langle m-1+(z+2)(l-j)\rangle\dots\langle m-1-(l-j)\rangle)$. Therefore, $d_{i,j}=d_{i,l}=0(0\le i\le m-2)$, which contradicts the assumption.

From the above, we can conclude that any valid nonzero codeword of the code has at least three nonzero columns, but it is easy to see that there is a codeword of column weight 3, thus the code has column distance of 3. □

3.3 Optimal Encoding Algorithm

According to (1) and (2), calculating each parity bit in column P needs $m-1$ XOR operations and calculating each parity bit in column Q needs m XOR operations. Thus, if the parity bits in columns P and Q are calculated independently, it will need on average $m-\frac{1}{2}$ XOR operations per parity bit for encoding. This is suboptimal, since the optimal encoding needs only $m-1$ XOR operations per parity bit. However, we can achieve the optimal encoding by coordinating the calculations of columns P and Q, specifically, by eliminating the repetitive calculations for certain *common subexpressions* of (1) and (2).

To understand how the scheme works, let us look closely at (2) again. Since the i-th diagonal traverses every row of the codeword, both the $(m-2-i)$-th row and the $(m-1-\langle 2i+2\rangle)$-th row have two data bits in the i-th diagonal parity group. In particular, the intersection of the $(m-2-i)$-th row and the i-th diagonal parity group is $\{d_{m-2-i,i+1},d_{m-2-i,\langle 2i+2\rangle}\}$, and the intersection of the $(m-1-\langle 2i+2\rangle)$-th row and the i-th diagonal parity group is $\{d_{m-1-\langle 2i+2\rangle,\langle 2i+2\rangle},d_{m-1-\langle 2i+2\rangle,\langle 3i+3\rangle}\}$. Therefore, $d_{m-2-i,i+1}\oplus d_{m-2-i,\langle 2i+2\rangle}$ is the common subexpression of the two expressions used to calculate p_{m-2-i} and q_i. Similarly, $d_{m-1-\langle 2i+2\rangle,\langle 2i+2\rangle}\oplus d_{m-1-\langle 2i+2\rangle,\langle 3i+3\rangle}$ is the common subexpression of the two expressions used to calculate $p_{m-1-\langle 2i+2\rangle}$ and q_i.

Then, we divide all the parity bits into $m-1$ pairs $\{(p_{m-2-i},q_i)|i=0,1,\cdots,m-2\}$. For each pair, we evaluate the common subexpression $d_{m-2-i,i+1}\oplus d_{m-2-i,\langle 2i+2\rangle}$ in advance, and reuse the result in both the calculations of p_{m-2-i} and q_i. In this way, we will eliminate a total of $m-1$ XOR operations during encoding. Now we need only $\frac{(m-1/2)\times 2(m-1)-(m-1)}{2(m-1)}=m-1$ XOR operations per parity bit for encoding, which is optimal. It is worth mentioning that utilizing the other type of common subexpressions can also achieve the same effect.

According to the optimizing scheme discussed above, a formal encoding algorithm can be described as follows.

Algorithm 1 (Optimal Encoding Algorithm): For each $i \in \{0, 1, \cdots, m-2\}$, the algorithm performs the following operations:

1) $r \leftarrow m-2-i$, $c_1 \leftarrow i+1$, $c_2 \leftarrow \langle 2i+2 \rangle$.
2) $p_r \leftarrow d_{r,c_1} \oplus d_{r,c_2}, q_i \leftarrow p_r$
3) $p_r \leftarrow p_r \oplus \left(\bigoplus_{\substack{c=0 \\ c \neq c_1, c_2}}^{m-1} d_{r,c} \right)$
4) $q_i \leftarrow q_i \oplus \left(\bigoplus_{\substack{c=0 \\ c \neq c_2}}^{m-1} d_{\langle i-c \rangle, c} \right) \oplus d_{m-1-c_2, c_2}$

4 Optimized Decoding Algorithms

Since our codes have the column distance of 3, they are able to correct two erasures and one error. We first present a decoding algorithm for two erasures, which can be used to reconstruct the lost data when no more than two disks fail. In addition to disk failures, data loss can also be caused by silent data corruptions, where transient data-corrupting errors occur in some data blocks but the system has no idea about which disk is affected and which blocks are corrupted. To this end, we also present a decoding algorithm that detects/locates and corrects one error caused by such silent data corruptions.

First, we present the decoding algorithm for correcting two erasures. Note that the proof of Theorem 1 has essentially provided a decoding algorithm for double erasures, thus we will mainly focus on optimizing the algorithm.

Algorithm 2 (Two-Erasure Decoding Algorithm): Like the encoding procedure, in order to eliminate the repetitive calculations in decoding, we need to evaluate the common sub-expressions first. Most of the common sub-expressions can be evaluated directly from the surviving bits, while a few special common sub-expressions need to be evaluated during decoding. Such special common sub-expressions that cannot be evaluated directly are considered *unknown common sub-expressions*.

If only one column (disk) is erased (failed), then the erased column can be easily reconstructed using either (1) or (2). Next, if two columns (disks) have been erased (failed), then there are four cases:

Case 1: Columns P and Q are erased. In this case, the reconstruction process is equivalent to the encoding process.

Case 2: Column Q and data column $j(0 \leq j \leq m-1)$ are erased. We can first reconstruct data column j using (1), then reconstruct column Q using (2) once data column j is reconstructed. Utilizing the common sub-expressions, we can achieve optimal decoding in this case.

Case 3: Column P and data column $j(0 \leq j \leq m-1)$ are erased. We can first reconstruct data column j using (2), then reconstruct column P using (1) once data column j is reconstructed. Again, optimal decoding can be achieved by reusing the values of the common sub-expressions.

Case 4: Data columns j and $l(0 \leq j < l \leq m-1)$ are erased. This is the main case. According to the proof of Theorem 1, S_i^P (S_i^Q) in this case denotes the XOR of the surviving bits in the i-th row (diagonal) parity group. For ease of notation, we let $E_i = d_{m-2-i,i+1} \oplus d_{m-2-i,\langle 2i+2 \rangle}$ and $\delta = l - j$ in what follows. Moreover, if E_i is an unknown common sub-expression (i.e., either $d_{m-2-i,i+1}$ or $d_{m-2-i,\langle 2i+2 \rangle}$ is erased), then we let $S_{m-2-i}^{P'}$ ($S_i^{Q'}$) denote the XOR of the surviving bits in the $(m-2-i)$-th (i-th) row (diagonal) parity group except the one that belongs to E_i.

When $j = 0$ and $1 \leq l \leq m-1$, we have two unknown common sub-expressions: $d_{m-1-l,l} \oplus d_{m-1-l,\langle 2l \rangle}$ and $d_{\langle v-l \rangle, \left\langle \frac{l}{2} \right\rangle} \oplus d_{\langle v-l \rangle, l}$. Thus, according to (4) and (5) we have

$$\begin{cases} d_{m-1-l,0} \oplus E_{l-1} = S_{m-1-l}^{P'} \\ d_{\langle v-l \rangle,0} \oplus E_v = S_{\langle v-l \rangle}^{P'} \\ d_{i,0} \oplus d_{i,l} = S_i^P \end{cases}, \tag{8}$$

where $i \notin \{m-1-l, \langle v-l \rangle\}$, and

$$\begin{cases} d_{l-1,0} \oplus E_{l-1} = S_{l-1}^{Q'} \\ d_{v,0} \oplus E_v \oplus d_{m-1-l,l} = S_v^{Q'} \\ d_{i,0} \oplus d_{\langle i-l \rangle, l} = S_i^Q \end{cases} \tag{9}$$

where $i \notin \{l-1, v\}$.

From the proof of Theorem 1 we have $\langle v + n_2 l \rangle = m-1-l$. First, we can get E_v by adding equations $d_{i,0} \oplus d_{i,l} = S_i^P (i = \langle v + \mathrm{t}l \rangle, 0 \leq t \leq n_2 - 1)$, $d_{i,0} \oplus E_{l-1} = S_i^{P'} (i = m-1-l)$, $d_{i,0} \oplus d_{\langle i-l \rangle, l} = S_i^Q (i = \langle v + \mathrm{t}l \rangle, 1 \leq t \leq n_2)$ and $d_{v,0} \oplus E_v \oplus d_{m-1-l,l} = S_v^{Q'}$, i.e., $E_v = S_v^{Q'} \oplus \left(\bigoplus_{t=0}^{n_2-1} S_{\langle v+\mathrm{t}l \rangle}^P \right) \oplus \left(\bigoplus_{t=1}^{n_2} S_{\langle v+\mathrm{t}l \rangle}^Q \right) \oplus S_{m-1-l}^{p'} \oplus d_{m-1-l,\langle 2l \rangle}$. Note that the v-th diagonal intersects the $\langle v + \mathrm{t}l \rangle$-th row in $(\langle v + \mathrm{t}l \rangle, \langle -\mathrm{t}l \rangle)$, thus $d_{\langle v+\mathrm{t}l \rangle, \langle -\mathrm{t}l \rangle} (1 \leq t \leq n_2 - 1)$ is involved in calculating both $S_v^{Q'}$ and $S_{\langle v+\mathrm{t}l \rangle}^P$, and $d_{m-1-l,\langle 3v+3 \rangle}$ is involved in calculating both $S_v^{Q'}$ and $S_{m-1-l}^{P'}$. Therefore, if we let $S_{\langle v+\mathrm{t}l \rangle}^{P'} = p_{\langle v+\mathrm{t}l \rangle} \oplus \left(\bigoplus_{\substack{c=1 \\ c \neq l, \langle -\mathrm{t}l \rangle}}^{m-1} d_{\langle v+\mathrm{t}l \rangle, c} \right) (1 \leq t \leq n_2 - 1)$, $S_{m-1-l}^{P''} = p_{m-1-l} \oplus \left(\bigoplus_{\substack{c=1 \\ c \neq l, \langle 2l \rangle, \langle 3v+3 \rangle}}^{m-1} d_{m-1-l,c} \right)$ and $S_v^{Q''} = q_v \oplus \left(\bigoplus_{\substack{c=1 \\ c \neq l, \langle -\mathrm{t}l \rangle}}^{m-1} d_{\langle v-c \rangle, c} \right) (1 \leq t \leq n_2)$, we have $S_v^{Q''} \oplus S_{m-1-l}^{P''} \oplus \left(\bigoplus_{t=1}^{n_2-1} S_{\langle v+\mathrm{t}l \rangle}^{P'} \right) = S_v^{Q'} \oplus S_{m-1-l}^{P'} \oplus \left(\bigoplus_{t=1}^{n_2-1} S_{\langle v+\mathrm{t}l \rangle}^P \right)$, thus $E_v = S_v^{Q''} \oplus S_v^P \oplus \left(\bigoplus_{t=1}^{n_2-1} S_{\langle v+\mathrm{t}l \rangle}^{P'} \right) \oplus S_{m-1-l}^{P''} \oplus \left(\bigoplus_{t=1}^{n_2} S_{\langle v+\mathrm{t}l \rangle}^Q \right) \oplus d_{m-1-l,\langle 2l \rangle}$. In this way, the calculation of E_v is optimized. Then, E_v can be used to calculate both $d_{\langle v-l \rangle, l}$ and $d_{\langle v-l \rangle, 0}$. Once we get $d_{\langle v-l \rangle, l}$, we can recover the other missing bits using (8) and (9) alternately.

If $1 \leq j < l \leq m-1$, according to the relationship between j and l, this case can be further divided into four subcases:

1) $l = \langle 2j \rangle$ and $j = \langle 2l \rangle$. From the proof of Theorem 1, we know that this subcase occurs only when $m = 3$, thus $j = 1$ and $l = 2$. According to (1) and (2) we have

$$\begin{cases} d_{0,1} \oplus d_{0,2} = d_{0,0} \oplus p_0 \\ d_{1,1} \oplus d_{1,2} = d_{1,0} \oplus p_1 \\ d_{1,1} \oplus d_{1,2} \oplus d_{0,2} = d_{0,0} \oplus q_0 \\ d_{0,1} \oplus d_{0,2} \oplus d_{1,1} = d_{1,0} \oplus q_1 \end{cases} \tag{10}$$

Thus, we can recover the erased bits through the following operations: (1)$S_0^P \leftarrow d_{0,0} \oplus p_0$, (2)$S_1^P \leftarrow d_{1,0} \oplus p_1$, (3)$S_0^Q \leftarrow d_{0,0} \oplus q_0$, (4)$S_1^Q \leftarrow d_{1,0} \oplus q_1$, (5)$d_{0,2} \leftarrow S_1^P \oplus S_0^Q$, (6)$d_{0,1} \leftarrow d_{0,2} \oplus S_0^P$, (7)$d_{1,1} \leftarrow S_0^P \oplus S_1^Q$, (8)$d_{1,2} \leftarrow d_{1,1} \oplus S_1^P$.

2) $l = \langle 2j \rangle$ and $j \neq \langle 2l \rangle$. In this subcase there are two unknown common sub-expressions: $d_{m-1-l,l} \oplus d_{m-1-l,\langle 2l \rangle}$ and $d_{\langle u-j \rangle, \left\langle \frac{j}{2} \right\rangle} \oplus d_{\langle u-j \rangle, j}$. Thus, according to (4) and (6) we have:

$$\begin{cases} d_{m-1-l,j} \oplus E_{l-1} = S_{m-1-l}^{P'} \\ E_u \oplus d_{\langle u-j \rangle, l} = S_{\langle u-j \rangle}^{P'} \\ d_{i,j} \oplus d_{i,l} = S_i^P \end{cases} \tag{11}$$

where $i \notin \{m-1-l, \langle u-j \rangle\}$, and

$$\begin{cases} E_{j-1} \oplus d_{m-1-l,l} = S_{j-1}^{Q'} \\ E_u \oplus d_{\langle u-l \rangle, l} \oplus d_{m-1-j,j} = S_u^{Q'} \\ d_{\langle l-1-j \rangle, j} \oplus E_{l-1} = S_{l-1}^{Q'} \\ d_{\langle i-j \rangle, j} \oplus d_{\langle i-l \rangle, l} = S_i^Q \end{cases} \tag{12}$$

where $i \notin \{j-1, u, l-1\}$.

First, we can directly calculate E_{j-1} according to (11), i.e., $E_{j-1} = d_{m-1-j,j} \oplus d_{m-1-j,l} = S_{m-1-j}^P$. Then, from Equation (12) we get $d_{m-1-l,l} = S_{j-1}^Q \oplus S_{m-1-j}^P$. Once we get $d_{m-1-l,l}$, we can determine E_{l-1}, and using E_{l-1} we can further determine the values of $d_{m-1-l,j}$ and $d_{\langle l-1-j \rangle, j}$ according to (11) and (12) respectively. Similarly, we can recover the other missing bits using (11) and (12) alternately. Note that when we get $d_{\langle u-j \rangle, l}$, we will first calculate E_u according to (11), so that its value can be used in the calculations of both $d_{\langle u-j \rangle, j}$ and $d_{m-1-j,j}$.

3) $l \neq \langle 2j \rangle$ and $j = \langle 2l \rangle$. This subcase is equivalent to the last one if we exchange the values of j and l.

4) $l \neq \langle 2j \rangle$ and $j \neq \langle 2l \rangle$. This is the main subcase. In this subcase there are four unknown common sub-expressions: $d_{m-1-j,j} \oplus d_{m-1-j,\langle 2j \rangle}$, $d_{\langle u-j \rangle, \left\langle \frac{j}{2} \right\rangle} \oplus d_{\langle u-j \rangle, j}$, $d_{m-1-l,l} \oplus d_{m-1-l,\langle 2l \rangle}$ and $d_{\langle v-l \rangle, \left\langle \frac{l}{2} \right\rangle} \oplus d_{\langle v-l \rangle, l}$. Thus, according to (4) and (7) we have:

$$\begin{cases} d_{m-1-j,l} \oplus E_{j-1} = S_{m-1-j}^{P'} \\ d_{m-1-l,j} \oplus E_{l-1} = S_{m-1-l}^{P'} \\ E_u \oplus d_{\langle u-j \rangle, l} = S_{\langle u-j \rangle}^{P'} \\ d_{\langle v-l \rangle, j} \oplus E_v = S_{\langle v-l \rangle}^{P'} \\ d_{i,j} \oplus d_{i,l} = S_i^P \end{cases} \tag{13}$$

where $i \notin \{m-1-j, m-1-l, \langle u-j \rangle, \langle v-l \rangle\}$, and

$$\begin{cases} d_{\langle j-1-l \rangle, l} \oplus E_{j-1} = S_{j-1}^{Q'} \\ E_u \oplus d_{\langle u-l \rangle, l} \oplus d_{m-1-j,j} = S_u^{Q'} \\ d_{\langle l-1-j \rangle, j} \oplus E_{l-1} = S_{l-1}^{Q'} \\ d_{\langle v-j \rangle, j} \oplus E_v \oplus d_{m-1-l,l} = S_v^{Q'} \\ d_{\langle i-j \rangle, j} \oplus d_{\langle i-l \rangle, l} = S_i^Q \end{cases} \tag{14}$$

where $i \notin \{j-1, u, l-1, v\}$.

Let $z = \left\langle \frac{-l}{\delta} \right\rangle$, according to the proof of Theorem 1, $m-1-l$ and $m-1-j$ divide the sequence $\{\langle m-1+n\delta \rangle, n = 0 \dots m\}$ into two sub-sequences: $\{\langle m-1+n\delta \rangle, n = 0 \dots z\}$ and $\{\langle m-1+n\delta \rangle, n = z+1 \dots m\}$. Then, the missing bits $d_{i,j}$ and $d_{i,l} (0 \leq i \leq m-2)$ can be divided into two parts according to the value of i, since i is either in the sub-sequence $\{\langle m-1+n\delta \rangle, n = 0 \dots z\}$ or in the sub-sequence $\{\langle m-1+n\delta \rangle, n = z+1 \dots m\}$. From Equation (14) we can find that only the u-th and v-th diagonal parity groups may contain both parts of the missing bits.

If z is odd, then $d_{m-1-l,l}$ is in the first part, while both $d_{\langle v-j \rangle, j}$ and $d_{\langle v-l \rangle, l}$ are in the second part. Moreover, $d_{m-1-j,j}$ is in the second part, while both $d_{\langle u-j \rangle, j}$ and $d_{\langle u-l \rangle, l}$ are in the first part. Thus, according to (13) and (14) we can get $d_{m-1-l,l}$ by adding equations that contain the second part of the missing bits, except for the equation $E_u \oplus d_{\langle u-l \rangle, l} \oplus d_{m-1-j,j} = S_u^{Q'}$. Once $d_{m-1-l,l}$ is determined, we can recover the other missing bits using (13) and (14) alternately. Similarly, we can also first determine $d_{m-1-j,j}$, then recover the other missing bits. If $z > \frac{m-1}{2}$, then we will calculate $d_{m-1-l,l}$ first, since calculating $d_{m-1-l,l}$ requires fewer XORs than calculating $d_{m-1-j,j}$. Otherwise, we will calculate $d_{m-1-j,j}$ first. In the case $z > \frac{m-1}{2}$, we have

$$d_{m-1-l,l} = S_{m-1-j}^{P'} \oplus S_{\langle v-l \rangle}^{P'} \oplus \left(\bigoplus_{\substack{t=z+2 \\ t \neq x}}^{m-1} S_{\langle m-1+t\delta \rangle}^P \right) \oplus S_{j-1}^{Q'} \oplus S_v^{Q'} \oplus \left(\bigoplus_{\substack{t=1 \\ t \neq x-z}}^{m-2-z} S_{\langle m-1+t\delta \rangle}^Q \right).$$

Observe that the v-th diagonal intersects the $\langle m-1+t\delta \rangle$-th row in $(\langle m-1+t\delta \rangle, \langle v+1-t\delta \rangle)$, thus we can eliminate the unnecessary XORs in the calculation of $d_{m-1-l,l}$. Specifically, if we let

$$S_v^{Q''} = q_v \oplus \left(\bigoplus_{t=1}^{z} d_{\langle t\delta-1 \rangle, \langle v+1-t\delta \rangle} \right)$$

$$S_{m-1-j}^{P''} = p_{m-1-j} \oplus \left(\bigoplus_{\substack{c=0 \\ c \neq j,l,\langle 2j \rangle, \\ \langle v+1+j \rangle}}^{m-1} d_{m-1-j,c} \right)$$

$$S^{P'}_{\langle t\delta-1\rangle} = p_{\langle t\delta-1\rangle} \oplus \left(\bigoplus_{\substack{c=0 \\ c\neq j,l,\langle v+1-t\delta\rangle}}^{m-1} d_{\langle t\delta-1\rangle,c} \right)$$
$$(z+2 \le t \le m-1 \text{ and } t \neq x, x+1)$$

, then we have

$$S^{Q''}_v \oplus S^{P''}_{m-1-j} \oplus \left(\bigoplus_{\substack{t=z+2 \\ t\neq x,x+1}}^{m-1} S^{P'}_{\langle t\delta-1\rangle} \right)$$
$$= S^{Q'}_v \oplus S^{P'}_{m-1-j} \oplus \left(\bigoplus_{\substack{t=z+2 \\ t\neq x,x+1}}^{m-1} S^{P}_{\langle t\delta-1\rangle} \right)$$

, thus

$$d_{m-1-l,l} = S^{Q''}_v \oplus S^{P''}_{m-1-j} \oplus \left(\bigoplus_{\substack{t=z+2 \\ t\neq x,x+1}}^{m-1} S^{P'}_{\langle t\delta-1\rangle} \right) \oplus S^{P}_{\langle v-j\rangle} \oplus S^{P'}_{\langle v-l\rangle} \oplus S^{Q'}_{j-1} \oplus \left(\bigoplus_{\substack{t=1 \\ t\neq x-z}}^{m-2-z} S^{Q}_{\langle m-1+t\delta\rangle} \right)$$

. In this way, the calculation of $d_{m-1-l,l}$ is optimized. If $z \le \frac{m-1}{2}$, the calculation of $d_{m-1-j,j}$ can be optimized with a similar method.

If z is even, then the two parts of the missing bits are independent of each other. Each part can be recovered by employing the method used in the case $j = 0$ and $1 \le l \le m-1$, thus we do not repeat the detailed decoding procedure here.

To make the above algorithm more intuitive, we give an example that illustrates the decoding procedure.

Example 1: Consider the standard Ultimate code with $m = 7$, whose diagonal parity groups are shown in Fig. 3.

D0	D1	D2	D3	D4	D5	D6	P	Q
0	1	2	3	4	5	**52**		0
1	2	3	4	5	**45**	0		1
2	3	4	5	**31**	0	1		2
3	4	5	**24**	0	1	2		3
4	5	**10**	0	1	2	3		4
5	**03**	0	1	2	3	4		5
	0	1	2	3	4	5		

Fig. 3. Diagonal parity groups in a standard Ultimate code with $m = 7$

Assume that data columns D1 and D3 are erased (lost), then we have $u = \langle \frac{1-2}{2} \rangle = 3$, $v = \langle \frac{3-2}{2} \rangle = 4$ and $z = \langle \frac{-3}{3-1} \rangle = 2$. Since z is even, the missing bits can be divided into two independent parts. The decoding procedures for the two parts are as follows:

Part 1

1) $S^{P'}_3 \leftarrow p_3 \oplus d_{3,0} \oplus d_{3,2} \oplus d_{3,4} \oplus d_{3,5}$
2) $S^{Q'}_4 \leftarrow q_4 \oplus d_{4,0} \oplus d_{2,2} \oplus d_{0,4} \oplus d_{5,6}$
3) $E_4 \leftarrow d_{3,6} \oplus S^{P'}_3 \oplus S^{Q'}_4$
4) $d_{1,3} \leftarrow E_4 \oplus d_{1,5}$
5) $d_{1,1} \leftarrow E_4 \oplus d_{1,0} \oplus d_{1,2} \oplus d_{1,4} \oplus d_{1,6} \oplus p_1$
6) $E_2 \leftarrow d_{2,0} \oplus d_{1,1} \oplus d_{0,2} \oplus d_{5,4} \oplus d_{4,5} \oplus d_{0,6} \oplus q_2$
7) $d_{3,3} \leftarrow d_{3,6} \oplus E_2$
8) $d_{3,1} \leftarrow S^{P'}_3 \oplus E_2$

Part 2

1) $S^{P'}_5 \leftarrow d_{5,0} \oplus d_{5,4} \oplus d_{5,6} \oplus p_5$
2) $E_5 \leftarrow d_{0,5} \oplus d_{0,6}$
3) $S^{P}_0 \leftarrow d_{0,0} \oplus d_{0,2} \oplus d_{0,4} \oplus E_5 \oplus p_0$
4) $E_1 \leftarrow d_{4,2} \oplus d_{4,4}$
5) $S^{Q}_1 \leftarrow d_{1,0} \oplus d_{2,4} \oplus d_{2,6} \oplus d_{3,5} \oplus E_1 \oplus q_1$
6) $S^{Q'}_3 \leftarrow d_{3,0} \oplus d_{1,2} \oplus d_{4,6} \oplus q_3$
7) $E_3 \leftarrow S^{Q'}_3 \oplus S^{P'}_5 \oplus S^{Q}_1 \oplus S^{P}_0 \oplus d_{5,2}$
8) $d_{2,1} \leftarrow E_3 \oplus d_{2,4}$
9) $d_{2,3} \leftarrow d_{2,0} \oplus E_3 \oplus d_{2,2} \oplus d_{2,5} \oplus d_{2,6} \oplus p_2$
10) $d_{4,1} \leftarrow d_{5,0} \oplus d_{3,2} \oplus d_{2,3} \oplus d_{1,4} \oplus d_{1,5} \oplus E_5 \oplus q_5$
11) $d_{4,3} \leftarrow d_{4,0} \oplus d_{4,1} \oplus E_1 \oplus d_{4,5} \oplus d_{4,6} \oplus p_4$
12) $E_0 \leftarrow d_{0,0} \oplus d_{4,2} \oplus d_{4,3} \oplus d_{3,4} \oplus d_{2,5} \oplus d_{1,6} \oplus q_0$
13) $d_{5,1} \leftarrow E_0 \oplus d_{5,2}$
14) $d_{5,3} \leftarrow S^{P'}_5 \oplus d_{5,5} \oplus E_0$
15) $d_{0,1} \leftarrow S^{Q}_1 \oplus d_{5,3}$
16) $d_{0,3} \leftarrow S^{P}_0 \oplus d_{0,1}$

Note that we have two zigzag recursions, which can be executed in parallel in a practical system. It is easy to check that the decoding algorithm needs 73 XORs in this case, which is exactly one more than the theoretical lower bound.

Next, we present the decoding algorithm in the case where at most one column is in error, that is, it detects/locates and then corrects the single error.

Algorithm 3 (Single-Error Decoding Algorithm): Given a possibly corrupted codeword, if at most one column is in error, this algorithm is able to locate and correct it. The decoding procedure works as follows.

First, let $S^P = (S^P_0, S^P_1, \ldots, S^P_{m-2}, 0)^T$ denote the horizontal syndrome vector and $S^Q = (S^Q_0, S^Q_1, \ldots, S^Q_{m-2}, 0)^T$ denote the diagonal syndrome vector. We compute them as

$$S^P_i = p_i \oplus \left(\bigoplus_{c=0}^{m-1} d_{i,c} \right) \quad (15)$$

$$S^Q_i = q_i \oplus d_{m-2-i,i+1} \oplus d_{m-1-\langle 2i+2\rangle,\langle 2i+2\rangle} \oplus \left(\bigoplus_{c=0}^{m-1} d_{\langle i-c\rangle,c} \right) \quad (16)$$

for $i = 0, 1, \cdots, m-2$.

Let $\vec{0}$ be the m-dimensional zero vector, then we distinguish between the following four cases:

1). $S^P = \vec{0}, S^Q = \vec{0}$. In this case the algorithm concludes that no errors have occurred and no further action is needed.

2). $S^P \neq \vec{0}, S^Q = \vec{0}$. In this case the error is in column P. We can correct this error by adding modulo-2 the first $m-1$ bits of the syndrome S^P to column P.

3). $S^P = \vec{0}, S^Q \neq \vec{0}$. In this case the error is in column Q. We can reconstruct this column by using (2).

4). $S^P \neq \vec{0}, S^Q \neq \vec{0}$. This is the main case, where the column in error must be one of the data columns. Note that the first $m-1$ bits of the syndrome S^P exactly represent the error itself, thus the main task is to locate the data column in error. If $S^P = S^Q$, the column in error must be the 0-th data column. Otherwise, suppose that the j-th data column $(1 \leq j \leq m-1)$ is in error, let us look into the relationship between S^P and S^Q. First, from (1) and (2) we know that $d_{m-1-j,j}$ is contained in both the $(j-1)$-th diagonal parity group and the u-th diagonal parity group, and $d_{\langle i-j\rangle,j}$ is contained in the i-th diagonal parity group. Then, we have $S^Q_{j-1} = S^P_{m-1-j}$, $S^Q_u = S^P_{m-1-j} \oplus S^P_{\langle u-j\rangle}$ and $S^Q_i = S^P_{\langle i-j\rangle} (0 \leq i \leq m-2, i \neq j-1, u)$. For any m-dimensional column vector $\vec{v} = (v_0, v_1, \cdots, v_{m-2}, 0)^T$, let $f(\vec{v}, j)$ denote the vector that is derived from $\vec{v}$ through the following operations: (1) $v_{m-1} \leftarrow v_{m-1-j}$, (2) $v_{\langle u-j\rangle} \leftarrow v_{\langle u-j\rangle} \oplus v_{m-1-j}$, (3) $v_{m-1-j} \leftarrow 0$ and (4) cyclically down-shift j positions. Then from the above, the relationship between S^P and S^Q can be denoted by $f(S^P, j) = S^Q$. Therefore, to locate the data column in error, we try to find the first index j with $1 \leq j \leq m-1$ such that $f(S^P, j) = S^Q$. If we can find such j, then the column in error must be the j-th data column. If there is no such j, the algorithm declares an uncorrectable error pattern. Finally, the algorithm corrects the error by adding modulo-2 the first $m-1$ bits of the syndrome S^P to the data column in error.

5 Constructing The Best Shortened Ultimate Codes

As mentioned before, when the number of data disks in the RAID-6 system is not a prime number, which is the common case, we need to employ the shortened codes. Specifically, if there are k data disks in the system, we will take a code with $m \geq k$ and assume that there are $m-k$ data disks that hold nothing but zeros (i.e., imaginary disks). As the code is asymmetric in that the selection of which $m-k$ data disks (columns) to be zero disks may affect the performance of the shortened code. For RDP codes, the best performance is achieved when the first $m-k$ data disks are zero disks, while EVENODD and Liberation codes perform best when the last $m-k$ data disks are zero disks.

To see how to construct the best shortened Ultimate codes, let us review the Optimal Encoding Algorithm. We find that each common subexpression can be used to eliminate one repetitive XOR operation. Thus, in order to reduce the encoding/decoding complexity to the greatest extent, we should make the shortened codes contain as many common subexpressions as possible. Observe that the first type of common subexpressions are $d_{m-1-j,j} \oplus d_{m-1-j,\langle 2j\rangle} (1 \leq j \leq m-1)$, thus we shorten the Ultimate codes as follows:

1) Set $A \leftarrow \{0, 1\}$, $B \leftarrow \{2, \cdots, m-1\}$, $i \leftarrow k-2$ and $j \leftarrow 1$

2) If $i = 0$ then stop, else set $j \leftarrow \langle 2j\rangle$.

3) If $j \in A$ then set j to be the maximum element of B.

4) Move j from B to A.

5) Set $i \leftarrow i-1$ and go to step 2.

Finally we will get two sets A and B. Set A contains the indices of the columns that correspond to the data disks, while Set B contains the indices of the columns that correspond to the (imaginary) zero disks. In this way, we have at least $k-2$ common subexpressions available with the shortened Ultimate codes. As we will see later, this algorithm constructs the lowest density shortened codes. Obviously, we can also obtain other shortened codes by changing the initial values of A and B. In particular, if we initialize the two sets with $A \leftarrow \{1, 2\}$ and $B \leftarrow \{0, \cdots, m-1\} - A$, then we can get shortened codes whose decoding performance are better than the lowest density ones.

6 Lowest Density Property

The Ultimate codes can also be defined by parity-check matrices in GF(2). Consider the parity check matrix for the shortened code with k data columns and $m-k$ zero columns. Let $w = m-1$, then the matrix is a $2w \times (k+2)w$ bit matrix, in which each row corresponds to a certain parity group and each column corresponds to a certain data/parity bit in the array. Each bit of the matrix indicates whether a certain data/parity bit is contained in a certain parity group. Then, according to the definition of the Ultimate codes, we can calculate the number of 1's in the matrix as $3(k-1) + 2(k \cdot w - (k-1)) + 2w = 2w(k+1) + k - 1$. Thus, the average number of 1's per row in the parity-check matrix is $(\frac{2w(k+1)+k-1}{2w}) = k + 1 + \frac{k-1}{2w}$, which is the lower bound proposed in [8].

Obviously, the lowest density parity-check matrix implies the lowest update penalty. Specifically, if the modified data bit $d_{i,j}$ is along the shared diagonal (i.e., $i + j = m - 1$), then we need to update three corresponding parity bits p_i, q_{j-1} and q_u, otherwise we only need to update two corresponding parity bits p_i and $q_{\langle i+j\rangle}$. Therefore, the average update penalty is $\frac{3(k-1)+2(k \cdot w-(k-1))}{k \cdot w} = 2 + \frac{k-1}{k \cdot w}$, which is asymptotically optimal as $w \to \infty$.

7 Evaluating Ultimate Codes: A Detailed Comparative Study on Complexity

In this section, we evaluate the Ultimate codes by comparing them with other representative RAID-6 codes in terms of encoding complexity, update complexity and decoding complexity. In all the tables and figures of this section, the numeric results were generated either by analytical models based on analysis presented in this paper or by running the corresponding implementations of the codes and counting the XOR operations.

7.1 Encoding Complexity

As mentioned before, the encoding complexity is measured as the average number of XOR operations required per coding (parity) bit. In the following, we distinguish between the two cases of (a) m varying with k and (b) m being fixed.

Case (a) happens if the RAID-6 system does not intend to alter the number of disks after initial deployment, then m is usually set to be the first prime number that is greater than k (or $k+1$ for RDP). In this way, the word size $w = m-1$ is minimized, which will result in the minimum memory consumption during encoding and decoding. For this case, it is a known fact that RDP codes outperform all the other alternatives in both encoding and decoding [7], thus we only compare our Ultimate codes with RDP codes here. Table 1 presents the encoding complexities of RDP codes and Ultimate codes for $2 \leq k \leq 33$, excluding the entries in which the two schemes perform identically.

TABLE 1
Encoding Complexities of RDP and Ultimate Codes as m varies with k

k	RDP	Ultimate	Ultimates Improvement over RDP
7	6.1	6	1.67%
9	8	8.05	−0.62%
13	12.06	12	0.5%
15	14	14.03	−0.21%
19	18.05	18	0.28%
21	20	20.02	−0.1%
23	22.07	22	0.32%
24	23.05	23.02	0.13%
25	24.04	24.02	0.08%
27	26	26.02	−0.08%
31	30.06	30	0.2%
32	31.04	31.01	0.1%
33	32.03	32.01	0.06%

From Table 1 we can see that by and large Ultimate codes slightly outperform RDP codes — there are only four entries in which RDP codes outperform Ultimate codes slightly. Nevertheless, the differences are quite small.

On the other hand, Case (b) arises if scalability and dynamic change of the RAID-6 system size are necessary, which is the common case. In this case, we usually set m to be a fixed, sufficiently large prime number. The m value must be large enough to accommodate all the possible numbers of data disks in the RAID-6 system anticipated by the system administrator. In this way, we can add (remove) disks to (from) the RAID-6 system "on the fly". Given the practical usage of RAID-6 codes in the production environment, we measure their shortened codes that are derived from the standard codes with $m = 17$ and $m = 31$. Since optimal encoding needs $k-1$ XORs per coding bit, we can normalize the encoding complexity by dividing the number of XORs per coding bit by $k-1$. The normalized results are presented in Fig. 4.

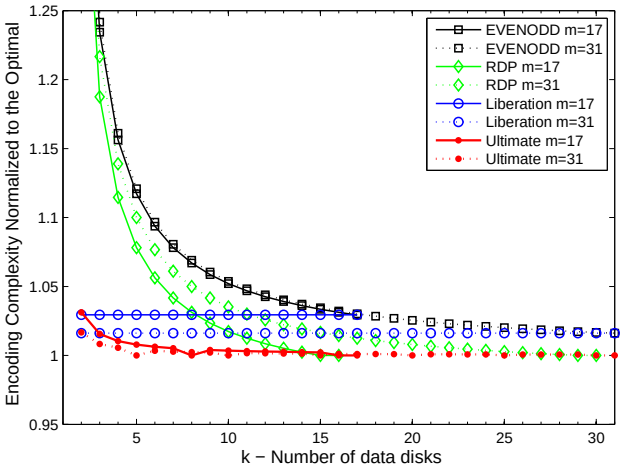

Fig. 4. Encoding complexities of various RAID-6 codes as a function of k with fixed m of 17 and 31 respectively.

We can see that the Ultimate codes outperform all the other RAID-6 codes for almost all k values. This is due to the shortening algorithm, which makes the shortened codes contain as many common subexpressions as possible. From Section 5 we can deduce that the encoding complexity of the shortened Ultimate codes is either $k-1$ or $k-1+\frac{1}{2(m-1)}$, and therefore their normalized complexity is either 1 or $1+\frac{1}{2(m-1)(k-1)}$. The normalized encoding complexity of the other shortened RAID-6 codes can also be calculated according to their encoding rules.

The most attractive feature of the Ultimate codes is that if m is chosen to be sufficiently large, then for every $k \leq m$ the encoding complexity of the shortened codes is either optimal or very close to being optimal. This is critically significant, since we are likely to employ the shortened codes in most instances. From Fig. 4 we can see that this property is only possessed by the Ultimate codes. In contrast, the curves of the Liberation codes are flat but always above the optimal

one, and the encoding complexities of the other two increase substantially as k shrinks.

7.2 Update Complexity

We measure the update complexity as the average number of parity bits that must be updated when a data bit is modified. Fig. 5 shows the results of various shortened codes. Like Liberation codes, the update complexity of the Ultimate codes is very close to the optimal value of two, and is asymptotically optimal as m grows. In contrast, the update complexity of the RDP codes is always roughly three, and that of the EVENODD codes increases as k grows, approaching an upper bound of three.

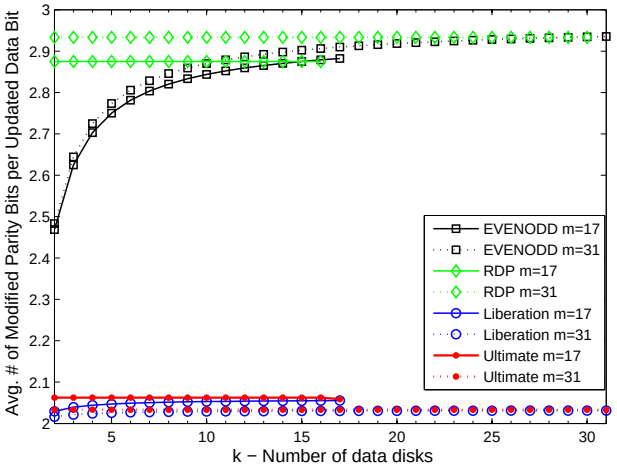

Fig. 5. Update complexities of various RAID-6 codes as a function of k with fixed m of 17 and 31 respectively.

7.3 Decoding Complexity

As with encoding, the decoding complexity can be measured by the average number of XOR operations required per recovered bit. Since the decoding complexity depends on which two columns are erased for most of the codes, we considered all the $\begin{pmatrix} k+2 \\ 2 \end{pmatrix}$ possible combinations of erasures and computed the average complexity. Again, we distinguish between the two cases of (a) m varying with k and (b) m being fixed.

For Case (a), as with encoding, we only compare Ultimate codes with RDP codes here. The normalized results are depicted in Fig. 6. It is clear that Ultimate codes perform slightly worse than RDP codes. Nevertheless, there is only a marginal difference between the two curves.

Now let us consider Case (b), namely, m is fixed. In this case, we measured the decoding complexities of all the representative RAID-6 codes. For Liberation codes, the number of XOR operations required by each failure pattern can be obtained by running the

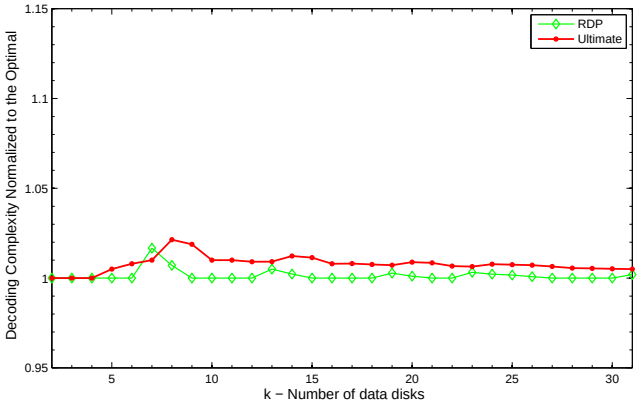

Fig. 6. Decoding Complexities of RDP and Ultimate Codes as m varies with k.

implementation in the Jerasure library [22]. For the other three, the numbers can be exactly calculated by analyzing their decoding algorithms. The results are plotted in Fig. 7, again normalized to the optimal.

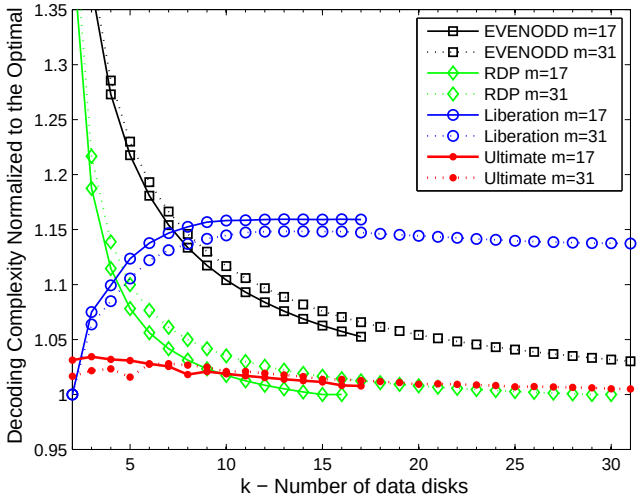

Fig. 7. Decoding complexities of various RAID-6 codes as a function of k with fixed m of 17 and 31 respectively.

Obviously, the Ultimate codes and the RDP codes exhibit the best decoding performance. For $m = 17$, the shortened Ultimate codes notably outperform all the other codes when $3 \leq k \leq 9$, and they perform slightly worse than the RDP codes when $10 \leq k \leq 16$. Similarly, for $m = 31$, the shortened Ultimate codes notably outperform all the other codes when $3 \leq k \leq 16$, and they perform slightly worse (within one percent) than the RDP codes when $17 \leq k \leq 30$. Furthermore, from Fig. 7 we can see that even in the worst case, i.e., $m = 17$ and $k = 3$, the decoding complexity of the shortened Ultimate codes is not more than four percent higher than the optimal. In other words, for every $k \leq m$ the decoding complexities of the shortened Ultimate codes are always very close to the optimal. In contrast, the decoding complexities of the Liberation codes increase as k grows, while

the decoding complexities of the other two increase noticeably as k shrinks.

8 Conclusions

We have presented a new class of MDS array codes, called Ultimate Codes, for building highly efficient and scalable RAID-6 systems. They are parity array codes with systematic, lowest density parity check matrices. Like other representative RAID-6 codes (e.g., EVENODD, RDP and Liberation codes), we can build scalable RAID-6 systems that allow adding or removing disks without re-encoding by choosing a sufficiently large prime number m and employing the corresponding shortened codes when $k < m$. Compared with the existing codes, the most attractive advantage of the Ultimate codes is the fact that if m is chosen to be sufficiently large, then for any $k \leq m$, the encoding, update and decoding complexities of the corresponding shortened code are either optimal or very close to being optimal. This allows us to achieve scalability with almost no performance loss. Considering all the metrics of erasure codes, we conclude that the Ultimate codes overall outperform other representative RAID-6 codes.

Our immediate future work is proceeding along two lines. First, although the Ultimate Codes are either optimal or very close to being optimal in terms of encoding, update and decoding complexities, the practical performance of RAID-6 systems also depends on the concrete implementation of the codes. The design and implementation of a coding scheme are two relatively independent issues, and different implementations of the same coding scheme may show different practical performances. Thus, we are seeking to work out an efficient implementation that can fully translate the complexity advantages of the Ultimate Codes into practical performance enhancement.

Another problem that we are interested is whether the Ultimate Codes can be generalized to tolerate three or more concurrent failures, while retaining all their attractive properties.

Acknowledgments

This work is supported in part by the National Basic Research Program (973 Program) of China under Grant No. 2011CB302305, the National Natural Science Foundation of China under Grant No. 61232004, and the US NSF under Grants NSF-IIS-0916859, NSF-CCF-0937993, NSF-CNS-1016609 and NSF-CNS-1116606.

References

[1] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (raid)," in *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '88. New York, NY, USA: ACM, 1988, pp. 109–116. [Online]. Available: http://doi.acm.org/10.1145/50202.50214

[2] W. Burkhard and J. Menon, "Disk array storage system reliability," in *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*, 1993, pp. 432–441.

[3] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "Raid: High-performance, reliable secondary storage," *ACM Computing Surveys*, vol. 26, no. 2, pp. 145–185, june 1994. [Online]. Available: http://doi.acm.org/10.1145/176979.176981

[4] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial & Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960. [Online]. Available: http://epubs.siam.org/doi/pdf/10.1137/0108018

[5] M. Blaum, J. Brady, J. Bruck, and J. Menon, "Evenodd: An efficient scheme for tolerating double disk failures in raid architectures," *Computers, IEEE Transactions on*, vol. 44, no. 2, pp. 192–202, 1995. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=364531

[6] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, 2004, pp. 1–14. [Online]. Available: https://www.usenix.org/publications/library/proceedings/fast04/tech/corbett/corbett_html/

[7] J. S. Plank, "The raid-6 liberation codes," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*. USENIX Association, 2008, p. 7.

[8] M. Blaum and R. M. Roth, "On lowest density mds codes," *Information Theory, IEEE Transactions on*, vol. 45, no. 1, pp. 46–59, 1999. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=746771

[9] S. N. I. Association, "The 2013 snia dictionary," https://www.snia.org/education/dictionary/r, 2013.

[10] J. Bloemer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An xor-based erasure-resilient coding scheme," ICSI, Berkeley, California, Tech. Rep. TR-95-048, August 1995.

[11] L. Xu and J. Bruck, "X-code: Mds array codes with optimal encoding," *Information Theory, IEEE Transactions on*, vol. 45, no. 1, pp. 272–276, 1999. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=746809

[12] L. Xu, V. Bohossian, J. Bruck, and D. G. Wagner, "Low-density mds codes and factors of complete graphs," *Information Theory, IEEE Transactions on*, vol. 45, no. 6, pp. 1817–1826, 1999. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=782102

[13] C. Jin, H. Jiang, D. Feng, and L. Tian, "P-code: A new raid-6 code with optimal properties," in *Proceedings of the 23rd international conference on Supercomputing*. ACM, 2009, pp. 360–369. [Online]. Available: http://dl.acm.org/citation.cfm?id=1542326

[14] Y. Cassuto and J. Bruck, "Cyclic lowest density mds array codes," *Information Theory, IEEE Transactions on*, vol. 55, no. 4, pp. 1721–1729, 2009. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4802333

[15] J. L. Hafner, "Weaver codes: highly fault tolerant erasure codes for storage systems," in *Proceedings of the 4th USENIX Conference on File and Storage Technologies*. USENIX Association, 2005, pp. 211–224.

[16] ——, "Hover erasure codes for disk arrays," in *International Conference on Dependable Systems and Networks*. IEEE, 2006, pp. 217–226. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1633511

[17] L. Hellerstein, G. A. Gibson, R. M. Karp, R. H. Katz, and D. A. Patterson, "Coding techniques for handling failures in large disk arrays," *Algorithmica*, vol. 12, no. 2-3, pp. 182–208, 1994.

[18] X. Luo and J. Shu, "Generalized x-code: An efficient raid-6 code for arbitrary size of disk array," *Trans. Storage*, vol. 8, no. 3, pp. 10:1–10:16, Sep. 2012. [Online]. Available: http://doi.acm.org/10.1145/2339118.2339121

[19] J. S. Plank, A. L. Buchsbaum, and B. T. Vander Zanden, "Minimum density raid-6 codes," *ACM Transactions on Storage (TOS)*, vol. 6, no. 4, p. 16, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1970340

[20] J. S. Plank and C. Huang, "Tutorial: Erasure coding for storage applications," Slides presented at FAST-2013: 11th Usenix Con-

ference on File and Storage Technologies, San Jose, February 2013.

[21] N. Sloane and F. J. MacWilliams, "The theory of error correcting codes," *North-Holland Math. Library*, vol. 16, 1977.

[22] J. S. Plank, S. Simmerman, and C. D. Schuman, "Jerasure: A library in c/c++ facilitating erasure coding for storage applications-version 1.2," *University of Tennessee, Tech. Rep. CS-08-627*, vol. 23, 2008.