Optimal Encoding and Decoding Algorithms for the RAID-6 Liberation Codes

Zhijie Huang*, Hong Jiang*, Zhirong Shen†, Hao Che*, Nong Xiao‡ and Ning Li*
*Dept. of Computer Science and Engineering, University of Texas at Arlington, Arlington, USA
†School of Informatics, Xiamen University, Xiamen, China
‡School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, China
*{zhijie.huang, hong.jiang, hche, ning.li}@uta.edu, †shenzr@xmu.edu.cn, ‡xiaon6@mail.sysu.edu.cn

***Abstract*—RAID-6 is gradually replacing RAID-5 as the dominant form of disk arrays due to its capability of tolerating concurrent failures of any two disks, as well as the case of encountering an uncorrectable read error during recovery. Implementing a RAID-6 system relies on some erasure coding schemes, and so far the most representative solutions are EVENODD codes [1], RDP codes [2] and Liberation codes [3], none of which has emerged as a clear "all-around" winner. In this paper, we are interested in revealing the undiscovered potential of the Liberation codes, since these codes have the following attractive features: (a) they have the best update performance, (b) they have *better scalability*, and (c) they are open-sourced and publicly available, as well as the following drawbacks: fair encoding performance and, more importantly, relatively poor decoding performance. Specifically, we present novel optimal encoding and decoding algorithms for the Liberation codes by introducing an alternative, geometric presentation of these codes. The proposed algorithms completely eliminate redundant computations during the encoding and decoding procedures by extracting and reusing common expressions between the two types of parity constraints, and do not involve any matrix operations on which the original algorithms are based. Our experiment results show that compared with the original solution, the proposed encoding and decoding algorithms reduce the number of XOR's by up to 16 percent and $15 \sim 20$ percent respectively, and the encoding and decoding throughputs are increased by 22.3 percent and at most 155 percent respectively. Moreover, the encoding complexity reaches the theoretical lower bound, while the decoding complexity is also very close to the theoretical lower bound.**

***Index Terms*—MDS array codes, RAID-6, reliability, erasure codes**

I. Introduction

The RAID (Redundant Arrays of Inexpensive Disks) technology [4] has been very widely adopted, almost ubiquitously in modern computing systems as a storage building block, due to its ability to exploit I/O parallelism and provide data protection. While there are several standard RAID organizations [5], only RAID-6 can tolerate any two concurrent disk failures and the case of encountering an uncorrectable read error during recovery, which are common failure patterns in modern storage systems due to the compounding impact of a dramatic increase in single disk capacity, a fairly constant per-bit error rate and a limited transfer rate [6]. Therefore, RAID-6 has become increasingly popular in modern storage applications, especially in large scale storage systems (e.g., cloud storage systems) that consist of a large number of less reliable (but more economical) disks such as SATA (vs. SCSI).

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. As a reference, the Linux RAID-6 implementation employes the well-known Reed-Solomon codes [7].

Although there are many erasure codes that can tolerate double disk failures, e.g., [8] [9] [10] [11] [1] [2] [3] [12], only a small part of them comply with the $P+Q$ specification [5]. Complying with the $P+Q$ specification is important, since it enables upgrading existing RAID-5 systems to RAID-6. Among the erasure codes that are perfectly suitable for RAID-6, the EVENODD codes [1], RDP codes [2] and Liberation codes [3] are the most representative ones. In general, they are all XOR-based, and hence perfom much better than the conventional Reed-Solomon codes which require finite field arithmetic during the encoding and decoding procedures. However, none of these representative codes has emerged as a clear "all-around" winner — each of them has its own limitations. For example, the EVENODD and RDP both have to update 3 (on average) parity blocks whenever a data block is changed, which is 1.5 times the theoretical lower bound, while the decoding complexity of the Liberation codes is $12\% \sim 18\%$ higher than the theoretical lower bound.

In this paper, we are interested in revealing the undiscovered potential of the Liberation codes, since these codes have the following attractive features: (a) changing a data block only results in updating 2 parity blocks, which attains the theoretical lower bound [13], and (b) they are open-sourced and publicly available [14], while EVENODD and RDP are both patented. The Liberation codes have been well recognized and have significant impact in both academia and industry, therefore, we believe it would be more desirable if we can encode and decode these important codes optimally.

Our basic idea is not to encode and decode with the original bitmatrix presentation, and try to exploit common expressions

1530-2075/20/$31.00 ©2020 IEEE
DOI 10.1109/IPDPS47924.2020.00078

between the two types of parity constraints as much as possible. First, we give an alternative presentation/definition of the Liberation codes, which can reveal the geometrical structure of the encoding rules. Then, common expressions between the two types of parity constraints are extracted and reused to reduce the number of XOR's required in both encoding and decoding procedures. Finally, we develop a recursive-based decoding algorithm to recover the missing blocks with minimized number of XOR's. In addition, in order to protect the system from data loss caused by silent data corruptions, we also provide an efficient algorithm for correcting a single column error. With our new encoding and decoding algorithms, the Liberation codes can encode optimally for any size of the RAID-6, and decode either optimally or near optimally, depending on the positions of the failed disks. We implemented our algorithms in the Jerasure liberary [14], which includes the original implementation of the Liberation codes. The evaluation results show that compared with the original algorithms, the proposed encoding and decoding algorithms reduce 16 percent and $15 \sim 20$ percent of XOR's respectively. Accordingly, the encoding and decoding throughputs are enhanced 22.3 percent and 155 percent at most respectively.

II. RAID-6 and Array Codes

In a broader sense, the term "RAID-6" represents any form of RAID that can tolerate any two concurrent disk failures [15]. Nevertheless, the *de facto* standard form of RAID-6 has been the so-called $P + Q$ redundancy [5], due to its compatibility with existing widely deployed RAID-5. Therefore, in what follows we will focus on the erasure codes that conform to the $P + Q$ redundancy form, which we refer to as "RAID-6 codes".

The first class of RAID-6 codes are the well known, all-purposed Reed-Solomon(RS) codes, which require complicated finite field arithmetic during the encoding and decoding procedures. Other than that, all the existing representative RAID-6 codes belong to a special class of erasure codes, called *array codes* [16], which involve only simple XOR and cyclic shift operations in both the encoding and decoding procedures, thus are *much more efficient* than the RS codes in terms of computational complexity.

A. Array Codes

In array codes, each codeword is a two-dimensional array of binary bits, containing both data bits and parity bits. 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 column erasures can be tolerated without data loss. A column is considered erased/erroneous if at least one bit in the column is erased/erroneous. Logically, each column of the codeword corresponds to a disk of a RAID-6 system, while each bit usually corresponds to a sector of a HDD, or a page of a SSD.

When implementing a RAID-6 system with array codes, every disk is first partitioned into a number of equal-size segments, called *strips*. Then, a *stripe* is defined as a maximal set of strips that are dependently related by an array code. Each disk contributes exactly one strip to a certain stripe. Finally, each strip is 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*. Figure 1 shows the relations among stripes, strips, and elements in a typical RAID-6 system. Actually, a stripe consists of an exact number of codewords that are interleaved for efficient computing [17]. For instance, if an element is defined to be a 64-bit machine word, then a stripe consists of 64 interleaved codewords. In this way, the XORs are performed on *machine words* rather than *bits*, i.e., the 64 codewords are always encoded and decoded in parallel. 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 the smallest unit of disk access, which can be a sector (HDD) or a page (SSD).

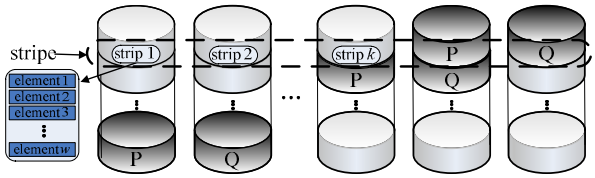


Fig. 1. Stripes, strips, and elements in a RAID-6 system

B. Key Performance Metrics of Coding

According to the operation characteristics of RAID-6 systems, array codes are usually evaluated in the following metrics:

- *Storage overhead* – the redundancy needed to provide a certain level of 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 [18], and the codes that attain this bound are classified as the *Maximum Distance Separable (MDS)* codes.
- *Encoding complexity* – the average number of XOR operations needed to compute each parity bit. For MDS array codes with k data columns and 2 parity columns, optimal encoding needs $k-1$ XOR operations per parity bit [2].
- *Decoding complexity* – the average number of XOR operations needed to reconstruct each missing bit. As with encoding, for MDS array codes with k data columns and 2 parity columns, optimal decoding needs $k-1$ XOR operations per missing bit.
- *Update complexity* – the average number of parity bits that must be updated whenever a data bit is modified. For r-erasure-correcting codes, the update complexity is at least r.

All of the above performance metrics are important in the practical RAID-6 systems, and hence there are research works

trying to improve any of these metrics. In particular, storage overhead directly translates into the cost for fault tolerance, thus MDS codes are preferred. Encoding and decoding complexities indicate how much computing resource is required during *full-stripe writes* and *reconstruction* (including *degraded reads*) respectively [19] [20]. And update complexity directly affects the performance of *small writes* [21] [22] [23], which are the dominant write operations in database systems and many big-data and data-intensive storage systems. Moreover, when applied to SSD arrays, update complexity can also affect the SSDs' service lifetime, since it dictates the *actual amount* of data being written into the SSDs whenever a small write occurs.

C. Motivation of this Work

According to the above performance metrics, the most representative RAID-6 codes should be EVENODD codes [1], RDP codes [2] and Liberation codes [3], all of which are MDS array codes. Let p denote a prime number, w denote the number of bits in a column, and k denote the number of data columns in a codeword, then the main characteristics of these codes can be summarized in the following table. Note that RDP codes achieve optimal performance in both encoding and decoding only when $k = p - 1$ or $k = p - 2$.

From the above, we can find that all of the existing codes have their own advantages and disadvantages, and none of them can truly represent the *de facto* standard of RAID-6 codes. In this paper, we are interested in revealing the undiscovered potential of the Liberation codes, since they have the following attractive features:

- the update complexity attains the theoretical lower bound of systematic RAID-6 codes [13];
- they are open-sourced and publicly available [14], while EVENODD and RDP are both patented;
- they have *better scalability* in that for any given p, their encoding/decoding performance will not shrink as $p - k$ increases [3].

On the other hand, the drawbacks of the Liberation codes are also quite notable: the *fair* encoding performance, and more importantly, the *relatively poor* decoding performance. Therefore, it is clearly desirable to have Liberation codes with optimal (or at least near optimal) encoding and decoding performances. To this end, we present the following improved encoding and decoding algorithms for the Liberation codes.

III. Optimal Encoding and Decoding Algorithms for the Liberation Codes

The Liberation codes are originally presented with the bitmatrix method, which is a generic method to describe any kind of array codes, but may generate suboptimal encoding and decoding algorithms. Therefore, to facilitate the following discussion, we first give an alternative presentation of the Liberation codes, which can explicitly reveal the geometric structure of the encoding rules.

A. An Alternative Presentation of Liberation Codes

In Liberation codes, a codeword is a $p \times (p + 2)$ array of binary bits, with the first p columns containing data bits and the last two columns containing parity bits, where p is an odd prime. For a RAID-6 system with $n = k+2$ disks, we need to choose a prime p such that $p \geq k$, then assume that there are $p - k$ "phantom" disks that hold nothing but zeros. In other words, the Liberation codes always work on the $p \times (p + 2)$ arrays. To formally describe the encoding rules, we use $b_{i,j}$ to denote the ith bit in the jth column, and let $\langle x \rangle = x \bmod p$. Then, the Liberation codes can also be defined as follows:

$$b_{i,p} = \bigoplus_{t=0}^{p-1} b_{i,t} \tag{1}$$

$$b_{i,p+1} = \left(\bigoplus_{t=0}^{p-1} b_{\langle i+t \rangle, t} \right) \oplus a_i \tag{2}$$

where $i = 0, 1, 2, \cdots, p - 1$, and

$$a_i = \begin{cases} b_{\langle -i-1 \rangle, \langle -2i \rangle}, \text{if } i \neq 0 \\ 0, \text{if } i = 0 \end{cases}.$$

Geometrically speaking, there are two types of parity bits: *row* parity bits and *anti-diagonal* (diagonal of slope -1) parity bits. Clearly, each row parity bit is obtained by xoring all the data bits in the same row. Define the i-th diagonal of slope s as the position set $\{(x, y)|x + sy \equiv i (\bmod p), 0 \leq x, y \leq p - 1\}$, then the ith anti-diagonal parity bit is calculated to be the even parity of the data bits along the anti-diagonal $\{(x, y)|x - y \equiv i (\bmod p), 0 \leq x, y \leq p-1\}$ and the bit lying in the intersection point of the $(i - 1)$th anti-diagonal and the $(p - 1)$th diagonal of slope $\frac{p-1}{2}$. Note that the 0th anti-diagonal parity bit is an exception in that it does not contain the above mentioned extra bit. As an example, Figure 2 shows the encoding rules of the Liberation code with $p = 5$, where row parity constraints are labeled with numbers and anti-diagonal parity constraints are labeled with captical letters.

TABLE I
A Summary of Representative RAID-6 Codes

	EVENODD	RDP	Liberation	Lower bound
w	$p - 1$	$p - 1$	p	—
Restriction on k	$\leq p$	$\leq p - 1$	$\leq p$	—
Storage Overhead	2	2	2	2
Encoding Complexity	$\approx k - 1/2$	$k - 1$	$k - 1 + \frac{k-1}{2p}$	$k - 1$
Decoding Complexity	$\approx k$	$k - 1$	$\approx 1.15(k - 1)$	$k - 1$
Update Complexity	≈ 3	≈ 3	≈ 2	2

	0	1	2	3	4	5	6
0	1A	1E	1DE	1C	1B	1	A
1	2B	2A	2E	2D	2CD	2	B
2	3C	3BC	3A	3E	3D	3	C
3	4D	4C	4B	4AB	4E	4	D
4	5E	5D	5C	5B	5A	5	E

Fig. 2. Encoding rules of the Liberation code with $p = 5$.

B. Optimal Encoding Algorithm

Although the parity bits can be computed by using (1) and (2) directly, we do not encode in this way since it contains redundant computations. Instead, we will exploit common expressions between (1) and (2) to minimize the encoding complexity. Consider the example shown in Figure 3, we can observe that $b_{2,0}$ and $b_{2,1}$ are both involved in the row parity constraint "3" and the anti-diagonal parity constraint "C", i.e., $b_{2,0} \oplus b_{2,1}$ is a common expression between parity constraints "3" and "C". Similarly, $b_{0,1} \oplus b_{0,2}$, $b_{3,2} \oplus b_{3,3}$, and $b_{1,3} \oplus b_{1,4}$ are common expressions between parity constraints "1" and "E", "4" and "B", "2" and "D", respectively.

	0	1	2	3	4	5	6
0	1A	1E	1DE	1C	1B	1	A
1	2B	2A	2E	2D	2CD	2	B
2	3C	3BC	3A	3E	3D	3	C
3	4D	4C	4B	4AB	4E	4	D
4	5E	5D	5C	5B	5A	5	E

Fig. 3. Common expressions in the Liberation code with $p = 5$.

Therefore, we can evaluate these common expressions first, and reuse their values in the subsequent encoding procedure. For instance, the parity bits in Figure 3 can be optimally calculated as follows:

1) $b_{0,5} \leftarrow b_{0,1} \oplus b_{0,2}$, $b_{4,6} \leftarrow b_{0,5}$
2) $b_{1,5} \leftarrow b_{1,3} \oplus b_{1,4}$, $b_{3,6} \leftarrow b_{1,5}$
3) $b_{2,5} \leftarrow b_{2,0} \oplus b_{2,1}$, $b_{2,6} \leftarrow b_{2,5}$
4) $b_{3,5} \leftarrow b_{3,2} \oplus b_{3,3}$, $b_{1,6} \leftarrow b_{3,5}$
5) $b_{0,5} \leftarrow b_{0,5} \oplus b_{0,0} \oplus b_{0,3} \oplus b_{0,4}$
6) $b_{1,5} \leftarrow b_{1,5} \oplus b_{1,0} \oplus b_{1,1} \oplus b_{1,2}$
7) $b_{2,5} \leftarrow b_{2,5} \oplus b_{2,2} \oplus b_{2,3} \oplus b_{2,4}$
8) $b_{3,5} \leftarrow b_{3,5} \oplus b_{3,0} \oplus b_{3,1} \oplus b_{3,2}$
9) $b_{4,5} \leftarrow b_{4,0} \oplus b_{4,1} \oplus b_{4,2} \oplus b_{4,3} \oplus b_{4,4}$
10) $b_{0,6} \leftarrow b_{0,0} \oplus b_{1,1} \oplus b_{2,2} \oplus b_{3,3} \oplus b_{4,4}$
11) $b_{1,6} \leftarrow b_{1,6} \oplus b_{0,4} \oplus b_{1,0} \oplus b_{2,1} \oplus b_{4,3}$
12) $b_{2,6} \leftarrow b_{2,6} \oplus b_{3,1} \oplus b_{4,2} \oplus b_{0,3} \oplus b_{1,4}$
13) $b_{3,6} \leftarrow b_{3,6} \oplus b_{3,0} \oplus b_{4,1} \oplus b_{0,2} \oplus b_{2,4}$
14) $b_{4,6} \leftarrow b_{4,6} \oplus b_{4,0} \oplus b_{1,2} \oplus b_{2,3} \oplus b_{3,4}$

The above procedure uses 40 XOR's, i.e., 4 XOR's per parity bit, attaining the theoretical lower bound.

When the number of data columns (disks) k is smaller than p, we can assume that the *last* $p - k$ columns are constant zeros. In this way, the encoding complexity is always $k - 1$ (XOR's per parity bit) for $2 \leq k \leq p$. Adopting the idea illustrated above, a formal encoding algorithm can be described as follows:

Algorithm 1: Optimal Encoding

Input: An uncoded stripe, p, k
Output: An encoded stripe

1 **for** $j \in \{1, 2, \cdots, k-1\}$ **do**
2 $\quad$ row $\leftarrow \left\langle \frac{p+1}{2} j - 1 \right\rangle$;
3 $\quad$ $b_{\text{row},k} \leftarrow b_{\text{row},j-1} \oplus b_{\text{row},j}$;
4 $\quad$ $b_{p-1-\text{row},k+1} \leftarrow b_{\text{row},k}$;
5 **end**
6 **for** $j \in \{0, 1, \cdots, k-1\}$ **do**
7 $\quad$ **for** $i \in \{0, 1, \cdots, p-1\}$ **do**
8 $\quad\quad$ **if** $\left\langle i + \frac{p-1}{2} j \right\rangle = \frac{p-1}{2}$ *and* $i \neq p-1$ **then**
9 $\quad\quad\quad$ continue;
10 $\quad\quad$ **end**
11 $\quad\quad$ **if** $b_{\langle i-j \rangle, k+1}$ *has not been accessed,* **then**
12 $\quad\quad\quad$ $b_{\langle i-j \rangle, k+1} \leftarrow b_{i,j}$;
13 $\quad\quad$ **else**
14 $\quad\quad\quad$ $b_{\langle i-j \rangle, k+1} \leftarrow b_{\langle i-j \rangle, k+1} \oplus b_{i,j}$;
15 $\quad\quad$ **end**
16 $\quad\quad$ **if** $\left\langle i + \frac{p-1}{2} j \right\rangle = p-1$ *and* $i \neq p-1$ **then**
17 $\quad\quad\quad$ continue;
18 $\quad\quad$ **end**
19 $\quad\quad$ **if** $b_{i,k}$ *has not been accessed* **then**
20 $\quad\quad\quad$ $b_{i,k} \leftarrow b_{i,j}$;
21 $\quad\quad$ **else**
22 $\quad\quad\quad$ $b_{i,k} \leftarrow b_{i,k} \oplus b_{i,j}$;
23 $\quad\quad$ **end**
24 $\quad$ **end**
25 **end**

Note that steps 7 and 11 are used to determine whether $b_{i,j}$ belongs to any common expression.

C. Optimal Decoding Algorithm

Since the Liberation codes are originally presented using bitmatrix method, the original decoding algorithm is accordingly based on a technique called *bitmatrix scheduling*, which cannot perfectly eliminate the redundant computations during the decoding procedure, resulting in a *relatively high* decoding complexity. However, from the alternative presentation we provided above, it is clear that the Liberation codes essentially have a similar geometric structure with EVENODD and RDP. Therefore, it should be possible to find an iterative approach to decode the Liberation codes. In this subsection we will provide such an approach.

If only one column is erased, or one of the two erased columns is a parity column, then it is quite easy to find an optimal decoding algorithm based on Algorithm 1. Therefore, *in what follows we only focus on the case that two data columns are erased*. Again, let us start with a small example, i.e., the Liberation code with $p = 5$. Suppose that columns 1 and 3 are both erased, as depicted in Figure 4.

	0	1	2	3	4	5	6
0	1A	1E	1DE	1C	1B	1	A
1	2B	2A	2E	2D	2CD	2	B
2	3C	3BC	3A	3E	3D	3	C
3	4D	4C	4B	4AB	4E	4	D
4	5E	5D	5C	5B	5A	5	E

Fig. 4. The Liberation code with $p = 5$, where columns 1 and 3 are both erased.

Let E_i denote the value of the common expression in the ith row, i.e., $E_0 = b_{0,1} \oplus b_{0,2}$, $E_1 = b_{1,3} \oplus b_{1,4}$, $E_2 = b_{2,0} \oplus b_{2,1}$, $E_3 = b_{3,2} \oplus b_{3,3}$. We refer to the common expressions containing at least one missing bit as *unknown common expressions*. Let S_i^P and S_i^Q denote the ith row parity syndrome and anti-diagonal parity syndrome respectively. Unlike the conventional definition, a parity syndrome here is the XOR of the surviving bits in the corresponding parity constraint, *excluding* the ones contained in an *unknown common expression*. With these notations, from (1) and (2) we can easily deduce the following equations:

$$\begin{aligned}
E_0 \oplus b_{0,3} &= S_0^P &&= b_{0,0} \oplus b_{0,4} \oplus b_{0,5} \\
E_1 \oplus b_{1,1} &= S_1^P &&= b_{1,0} \oplus b_{1,2} \oplus b_{1,5} \\
E_2 \oplus b_{2,3} &= S_2^P &&= b_{2,2} \oplus b_{2,4} \oplus b_{2,5} \\
E_3 \oplus b_{3,1} &= S_3^P &&= b_{3,0} \oplus b_{3,4} \oplus b_{3,5} \\
b_{4,1} \oplus b_{4,3} &= S_4^P &&= b_{4,0} \oplus b_{4,2} \oplus b_{4,4} \oplus b_{4,5} \\
b_{1,1} \oplus b_{3,3} &= S_0^Q &&= b_{0,0} \oplus b_{2,2} \oplus b_{4,4} \oplus b_{0,6} \\
b_{2,1} \oplus b_{4,3} \oplus E_3 &= S_1^Q &&= b_{1,0} \oplus b_{0,4} \oplus b_{1,6} \\
b_{3,1} \oplus b_{0,3} \oplus E_2 &= S_2^Q &&= b_{4,2} \oplus b_{1,4} \oplus b_{2,6} \\
b_{4,1} \oplus E_1 &= S_3^Q &&= b_{3,0} \oplus b_{0,2} \oplus b_{3,6} \\
b_{2,3} \oplus E_0 &= S_4^Q &&= b_{4,0} \oplus b_{3,4} \oplus b_{4,6}
\end{aligned}$$

From these equations we can retrieve the missing bits through the following 3 steps. First, evaluate all the parity syndromes directly according to the above equations. Then, find a starting point by adding a certain subset of the above equations: $b_{3,1} \leftarrow S_0^P \oplus S_4^Q \oplus S_2^P \oplus S_2^Q$. Finally, iteratively retrieve the other missing bits by using row and anti-diagonal parity constraints alternately, as follows:

1) $E_3 \leftarrow b_{3,1} \oplus S_3^P$
2) $S_1^Q \leftarrow E_3 \oplus S_1^Q$; $b_{3,3} \leftarrow b_{3,2} \oplus E_3$
3) $b_{1,1} \leftarrow b_{3,3} \oplus S_0^Q$
4) $E_1 \leftarrow b_{1,1} \oplus S_1^P$
5) $b_{1,3} \leftarrow E_1 \oplus b_{1,4}$; $b_{4,1} \leftarrow E_1 \oplus S_3^Q$
6) $b_{4,3} \leftarrow b_{4,1} \oplus S_4^P$
7) $b_{2,1} \leftarrow b_{4,3} \oplus S_1^Q$
8) $E_2 \leftarrow b_{2,0} \oplus b_{2,1}$
9) $b_{2,3} \leftarrow E_2 \oplus S_2^P$
10) $E_0 \leftarrow b_{2,3} \oplus S_4^Q$
11) $b_{0,1} \leftarrow E_0 \oplus b_{0,2}$; $b_{0,3} \leftarrow E_0 \oplus S_0^P$

The above procedure uses 39 XOR's, i.e., 3.9 XOR's per missing bit. Indeed, this is even lower than the "lower bound", but note that other erasure patterns may have a higher decoding complexity, and hence the average value is still not less than the lower bound.

Actually, the 3 steps decoding procedure is quite similar with that of the EVENODD and RDP codes. However, due to the existence of common expressions and the absence of the imaginary 0-row, we need to use some tricks to minimize the decoding complexity:

1) As with encoding, common expressions (except the *unknown* ones) should be reused while computing both types of parity syndromes;
2) There are two ways to find a starting point, choose the one with less XOR's;
3) Each iteration in Step 3 retrieves either a missing bit or an *unknown common expression*, if it is the latter, then use its value twice in the next iteration.

Note that in the above example E_2 has been used once while computing the starting point $b_{3,1}$, so it only appears once in Line 9.

From the above, the core of decoding lies in the Step 2, i.e., finding a starting point to initiate the iteration process. Therefore, we first give an efficient algorithm to show how to find the starting point. Suppose the lth and rth columns are erased, where $0 \le l < r \le p-1$. The following algorithm return two sets of indices of parity constraints being involved in computing the starting point $b_{x,r}$. If the starting point is in the lth column, the algorithm will fail and return -1. In this case we may exchange the values of l and r, and call the algorithm again.

Algorithm 2: Finding the Starting Point

Input: l, r, p
Output: S^P, S^Q, x

```
1  extraL ← p − 1 − ⟨(p − 1)l/2⟩;
2  extraR ← p − 1 − ⟨(p − 1)r/2⟩;
3  specialQL ← ⟨extraL + 1 − l⟩;
4  specialQR ← ⟨extraR + 1 − r⟩;
5  curQ ← ⟨specialQR − 1 + (r − l)⟩;
6  S^Q ← {specialQR};
7  S^P ← {extraR};
8  while (curQ ≠ specialQL or l = 0) and
     curQ ≠ specialQR do
9      S^Q ← S^Q ∪ {curQ};
10     S^P ← S^P ∪ {⟨curQ + r⟩};
11     curQ ← ⟨curQ + (r − l)⟩;
12 end
13 if curQ = specialQR then
14     x ← extraR + 1;
15 else
16     x ← −1;
17 end
```

Note: extraL and extraR refer to the row indices of the two extra bits in the erased columns, while specialQL and

specialQR refer to the indices of the two anti-diagonal parity constraints containing 3 unknown elements.

Next, we need to compute the two types of parity syndromes. This step is quite similar to the encoding procedure, and hence we just present the algorithm, without further detailed explanations. Suppose the lth and rth columns are erased, then we use $b_{i,l}$ to store the i-th row syndrome, and $b_{\langle i+r\rangle,r}$ to store the i-th anti-diagonal syndrome.

Algorithm 3: Syndromes Computation

Input: A stripe with the l-th and r-th strips being erased, p, k
Output: A stripe with the l-th and r-th strips containing row and anti-diagonal syndromes respectively

for $j \in \{1, 2, \cdots, k-1\}$ **do**
  row $\leftarrow \left\langle \frac{p+1}{2} j - 1 \right\rangle$;
  if $l \in \{j-1, j\}$ *or* $r \in \{j-1, j\}$ **then** continue;;
  $b_{\text{row},l} \leftarrow b_{\text{row},j-1} \oplus b_{\text{row},j}$;
  $b_{\langle p-1-\text{row}+r\rangle,r} \leftarrow b_{\text{row},l}$;
end
for $j \in \{0, 1, \cdots, k-1\}$ **do**
  if $j \in \{l, r\}$ **then** continue;;
  for $i \in \{0, 1, \cdots, p-1\}$ **do**
    if $\left\langle i + \frac{p-1}{2} j \right\rangle = \frac{p-1}{2}$ *and* $i \neq p-1$ **then** continue;;
    row $\leftarrow \langle i - j + r \rangle$;
    if $b_{\text{row},r}$ *has not been accessed before* **then**
      $b_{\text{row},r} \leftarrow b_{i,j}$;
    else
      $b_{\text{row},r} \leftarrow b_{\text{row},r} \oplus b_{i,j}$;
    end
    if $\left\langle i + \frac{p-1}{2} j \right\rangle = p-1$ *and* $i \neq p-1$ **then** continue;;
    if $b_{i,l}$ *has not been accessed before* **then**
      $b_{i,l} \leftarrow b_{i,j}$;
    else
      $b_{i,l} \leftarrow b_{i,l} \oplus b_{i,j}$;
    end
  end
end
for $i \in \{0, 1, \cdots, p-1\}$ **do**
  $b_{i,l} \leftarrow b_{i,l} \oplus b_{i,k}$;
  $b_{i,r} \leftarrow b_{i,r} \oplus b_{\langle i-r\rangle,k+1}$;
end

Based on the above two algorithms, a formal decoding algorithm now can be described as follows.

Algorithm 4: Optimal Decoding

Input: A stripe with the l-th and r-th strips being erased, p, k
Output: A recovered stripe

; `// Find the starting point`
$(S^P, S^Q, x) \leftarrow \text{getStartingPoint}(p, l, r)$;
if $x = -1$ **then**
  Exchange the values of l and r;
  $(S^P, S^Q, x) \leftarrow \text{getStartingPoint}(p, l, r)$;
end
Compute syndromes using Algorithms 3;
; `// Evaluate the starting element` $b_{x,r}$
$\delta \leftarrow \langle r - l \rangle$;
for *each* $i \in S^Q$ **do**
  if $x = \langle i + r \rangle$ **then** continue;;
  $b_{x,r} \leftarrow b_{x,r} \oplus b_{\langle i+r\rangle,r}$;
end
for *each* $i \in S^P$ **do**
  $b_{x,r} \leftarrow b_{x,r} \oplus b_{i,l}$;
end
; `// Retrieve other missing elements`
for *each* $t \in \{0, 1, \cdots, p-1\}$ **do**
  $b_{x,l} \leftarrow b_{x,l} \oplus b_{x,r}$;
  if $\left\langle x + \frac{p-1}{2} r \right\rangle = p-1$ *and* $x \neq p-1$ *and* $\delta \neq 1$ **then**
    $b_{x,l} \leftarrow b_{x,l} \oplus b_{x,r-1}$;
  else if $\left\langle x + \frac{p-1}{2} r \right\rangle = \frac{p-1}{2}$ *and* $x \neq p-1$ **then**
    $b_{x,r} \leftarrow b_{x,r} \oplus b_{x,r+1}$;
  end
  if $\left\langle x + \frac{p-1}{2} l \right\rangle = p-1$ *and* $x \neq p-1$ **then**
    $b_{\langle x+1+\delta\rangle,r} \leftarrow b_{\langle x+1+\delta\rangle,r} \oplus b_{x,l}$;
    $b_{x,l} \leftarrow b_{x,l} \oplus b_{x,l-1}$;
  end
  if $t < p-1$ **then** $b_{\langle x+\delta\rangle,r} \leftarrow b_{\langle x+\delta\rangle,r} \oplus b_{x,l}$;
  if $\left\langle x + \frac{p-1}{2} l \right\rangle = \frac{p-1}{2}$ *and* $x \neq p-1$ *and* $\delta \neq 1$ **then**
    $b_{x,l} \leftarrow b_{x,l} \oplus b_{x,\langle l+1\rangle}$;
  end
  $x \leftarrow \langle x + \delta \rangle$;
end

IV. Performance Evaluation

There are two major ways to evaluate the encoding /decoding performance of xor-based erasure codes, namely, the number of XOR's being used during the encoding/decoding procedure, and the observed encoding/decoding throughputs. We will use both approaches to show the effect of the proposed algorithms.

In the following, we distinguish between the two cases of (a) p varying with k and (b) p being fixed. Case (a) happens if the RAID-6 system does not intend to alter the number of disks after initial deployment, then p is usually set to be the first prime number that is greater than k (or $k+1$ for RDP). In this way, the column size is minimized, which will result in the minimum memory consumption during encoding and decoding procedures. 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 p to be a fixed, sufficiently large prime number. The p 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" [24].

A. *Encoding/Decoding Complexity*

As mentioned before, the encoding (decoding) complexity is defined as the average number of XOR operations required per parity (missing) bit. As the lower bounds of encoding and decoding complexities are both $k-1$ for k data disks, we can normalize the encoding/decoding complexity by dividing it by $k-1$. Then, 1 stands for the theoretical lower bound of the normalized encoding/decoding complexity.

Figure 5 shows the normalized encoding complexity of different RAID-6 codes, where p is varying with k. It is quite clear that the proposed encoding algorithm enables the Liberation codes to reach the lower bound for any value of k, outperforming all the other competitors. In contrast, the original encoding performs only slightly better than the EVENODD and notably worse than the RDP.

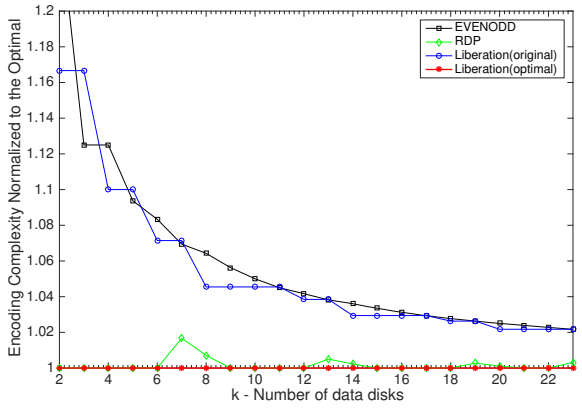


Fig. 5. Normalized encoding complexities of different RAID-6 codes

For the case of p being fixed, we set $p = 31$, and study how the encoding complexity varies as k decreases. Figure 6 shows the encoding complexities of different RAID-6 codes as k varies from 2 to 23. We can see that the encoding complexities of the EVENODD and RDP both increase substantially as k shrinks, while the the curves of the Liberation codes are flat. That is why we mentioned in the Introduction that the Liberation codes have better scalability than the EVENODD and RDP codes. In this case, the original encoding of the Liberation codes is already very close to the theoretical lower bound, thus the difference between the proposed algorithm and the original one is negligible.

For decoding complexity, since the number of XOR's required for decoding usually depends on the specific erasure pattern, we test all the possible erasure patterns and use their average value as the decoding complexity. Figure 7 shows the normalized decoding complexity of different RAID-6 codes. Compared to the original decoding algorithm, the proposed decoding has a complexity that is very close to the theoretical lower bound, and is about $15 \sim 20$ percent lower than the original one. Although it is still up to 2 percent higher than the decoding complexity of RDP, the difference is really marginal.

For the case of p being fixed, as with encoding, we set $p = 31$ and study how the decoding complexity varies as

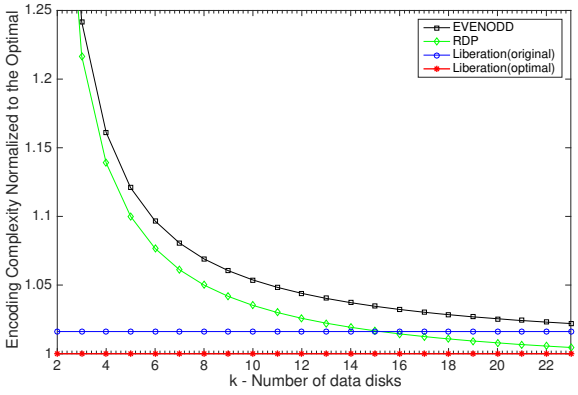


Fig. 6. Normalized encoding complexities of different RAID-6 codes($p = 31$)

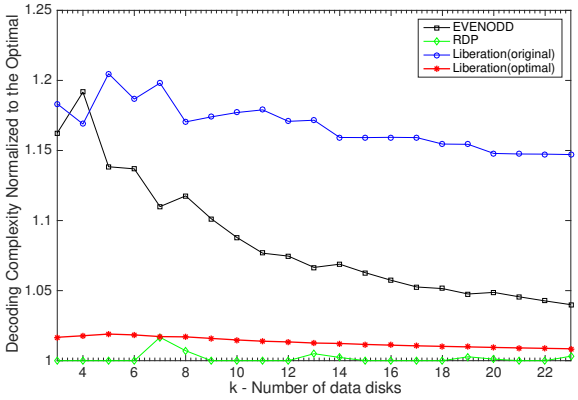


Fig. 7. Normalized decoding complexities of different RAID-6 codes

k decreases. Figure 8 shows the decoding complexities of different RAID-6 codes as k varies from 2 to 23. We can see that the decoding complexities of the EVENODD and RDP both increase dramatically as k shrinks, while the decoding complexities of the Liberation codes are mostly $10 \sim 15$ percent higher than the theoretical lower bound. In contrast, the decoding complexity of our proposed algorithm is only $0 \sim 2.5$ percent higher than the theoretical lower bound.

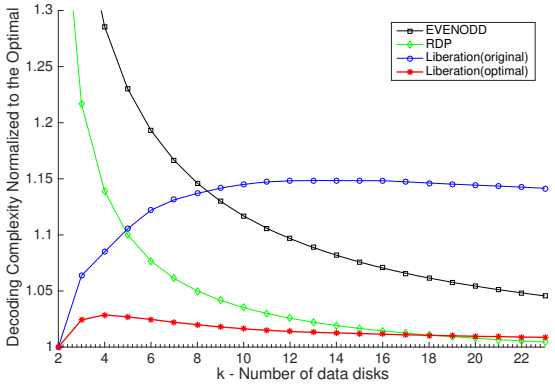


Fig. 8. Normalized decoding complexities of different RAID-6 codes($p = 31$)

B. Encoding/Decoding Throughputs

In order to test the real performance, and verify the correctness of our proposed algorithms, we implemented the optimal encoding and decoding algorithms in the Jerasure liberary [14]. We use all the default compile and build options, and use the test programe provided in the liberary to get the following results. The expriments are running on a Macbook Pro 2016 with Intel 2.6GHz Core i7, 16GB 2133MHz LPDDR3, and 256GB PCI-E SSD.

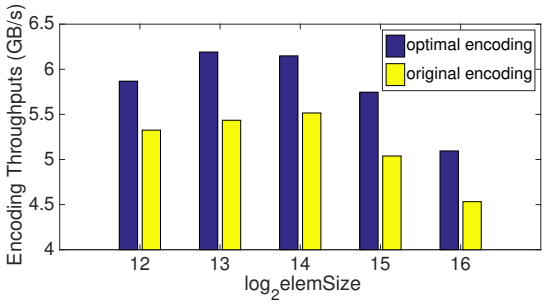
Note that there are *two main factors* that may affect the real performance of an erasure code, namely, the encoding/decoding complexity and the way it is implemented. Therefore, in this section we only compare the proposed encoding and decoding algorithms with the original ones, whose implementation is open-sourced and publicly available. We do not compare the proposed algorithms with the EVENODD and RDP codes due to the following reasons:

- EVENODD and RDP are both patented and their official implementations are unavailable;
- If we implement these codes by ourselves, then it is unclear that whether they are implemented in the best way;
- The target of this work is optimizing the Liberation codes, rather than beating other RAID-6 codes.

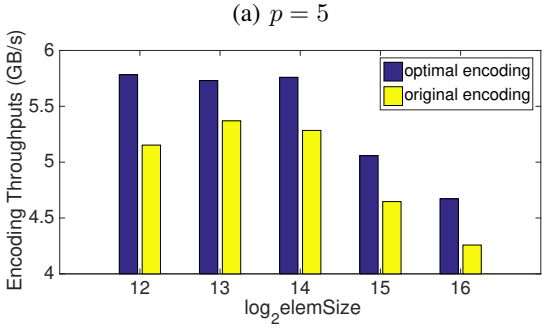
Since the *element* size may affect the encoding/decoding performance, we first find the element sizes with the best performance. Figure 9 shows the encoding throughputs with different element size for $p = 5, 7, 11$. It is clear that the Liberation codes achieve the best encoding performance when the element size is 8KB. Another important element size is 4KB, which is the default memory page size, as well as most of SSDs' page size. Thus, in what follows we will focus our study on the encoding/decoding performance of the Liberation codes with the element size being set to be 8KB and 4KB.

Figures 10 and 11 show the encoding throughputs of Liberation codes with varing p and fixed p respectively. From the figures, both algorithms suffer a performance loss as k increases, since larger k implies more XOR's and larger stripe size, which may increase the number of cache misses. It is also clear that the performance difference between the two algorithms is always notable no matter how k varies, though their encoding complexities tend to be identical as k grows. This may be caused by the additional overhead of the bitmatrix scheduling, on which the original encoding algorithm is based.

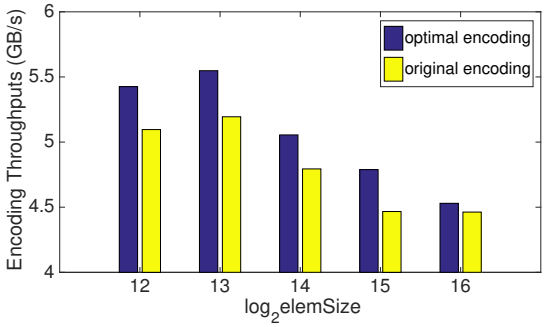
To obtain decoding throughputs, we test the throughputs of all the possible erasure patterns, and get the average value of these results. Figures 12 and 13 show the decoding throughputs of Liberation codes with varing p and fixed p respectively. As expected, the performance difference between the two decoding algorithms is much greater than their difference in decoding complexity. In particular, the proposed decoding algorithm outperforms the original one nearly 100%(element_size = 8KB) or even more than 150%(element_size = 4KB) when k is large enough. This is mainly caused by the additional overhead of the bitmatrix scheduling, which includes time consuming



(a) $p = 5$



(b) $p = 7$



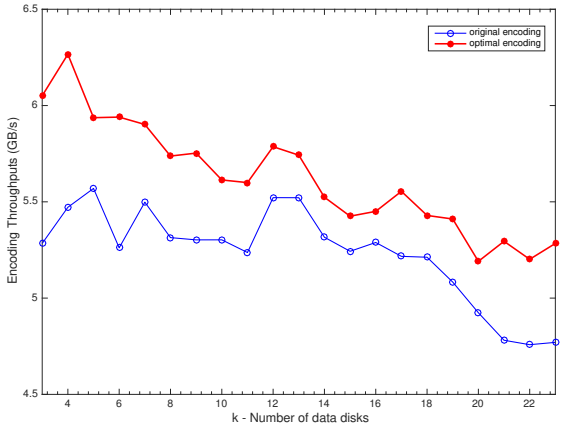
(c) $p = 11$

Fig. 9. Encoding throughputs with different element size

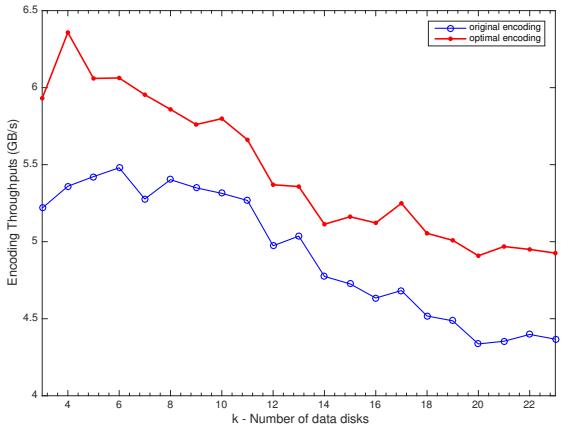
matrix operations, especially when p is large. In contrast, the proposed optimal decoding does not depend on any matrix operations, hence avoids severe performance loss as k grows.

V. Conclusion

We have presented novel optimal encoding and decoding algorithms for the Liberation codes — an important class of xor-based erasure codes for RAID-6. The Liberation codes are open-sourced and have been well recognized in both academia and storage industry due to their optimal update performance and better scalability comparing with other representative RAID-6 codes. However, these codes have fair encoding performance and, more importantly, relatively poor decoding performance due to the bitmatrix presentation on which they are based. To address this issue, we used an alternative way to present the Liberation codes, and proposed iterative encoding and decoding algorithms based on the alternative presenta-

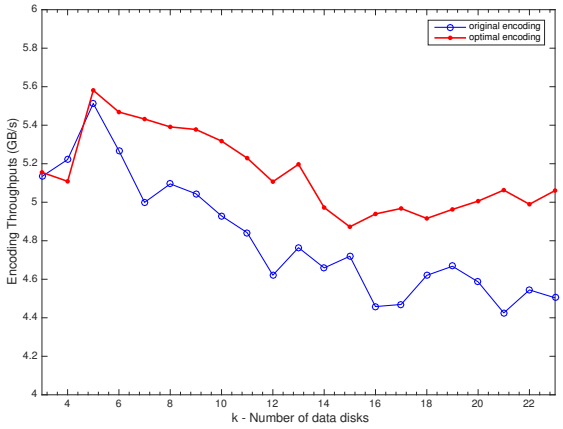


(a) element_size = 4KB

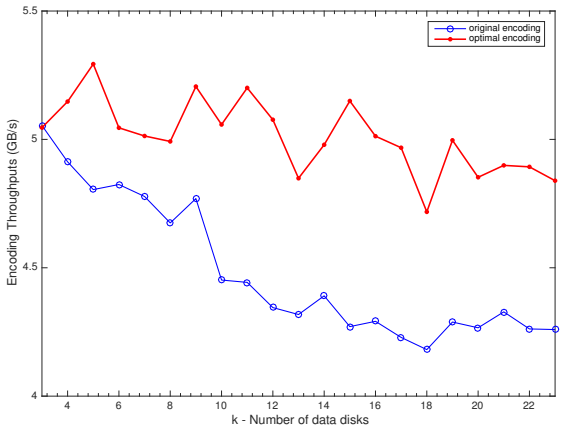


(b) element_size = 8KB

Fig. 10. Encoding throughputs of Liberation codes with p varing with k



(a) element_size = 4KB



(b) element_size = 8KB

Fig. 11. Encoding throughputs of Liberation codes with $p = 31$

tion. The proposed algorithms completely eliminate redundant computations during the encoding and decoding procedures by extracting and reusing common expressions between the two types of parity constraints, and do not involve any matrix operations. The experiment results show that compared with the original solution, the proposed encoding and decoding algorithms reduce the number of XOR's by up to 16 percent and 15 $\sim$ 20 percent respectively, and the encoding and decoding throughputs are increased by up to 22.3 percent and at most 155 percent respectively. Moreover, the encoding complexity reaches the theoretical lower bound, while the decoding complexity is also very close to the theoretical lower bound. With the proposed optimal encoding and decoding algorithms, the Liberation codes are very likely to be the best alternative for RAID-6 implementors.

ACKNOWLEDGMENT

This research is partially supported by research grants from the U.S. National Science Foundation (NSF) under Grant Nos. CCF-1704504 and CCF-1629625, the National Key R&D Program of China under Grant No. 2017YFB1001600, and the National Science Foundation of China (NSFC) under Grant Nos. 61832020, 61702569, 61602120 and 61872392. We would like to thank the anonymous reviewers for their constructive comments.

REFERENCES

[1] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 192–202, 1995.

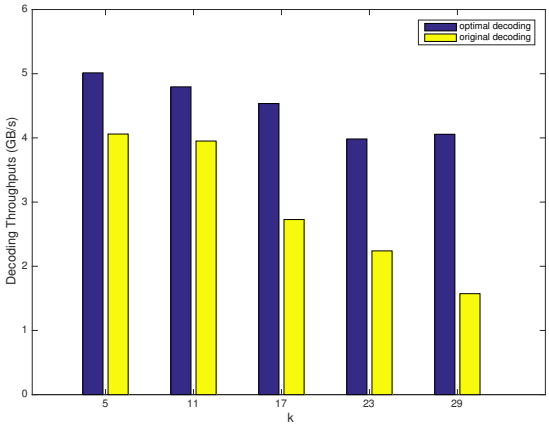
[2] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *the 3rd USENIX Conference on File and Storage Technologies (FAST'04)*, 2004, pp. 1–14.

[3] J. S. Plank, "The RAID-6 liberation codes," in *the 6th USENIX Conference on File and Storage Technologies (FAST'08)*. USENIX Association, 2008, pp. 97–110.

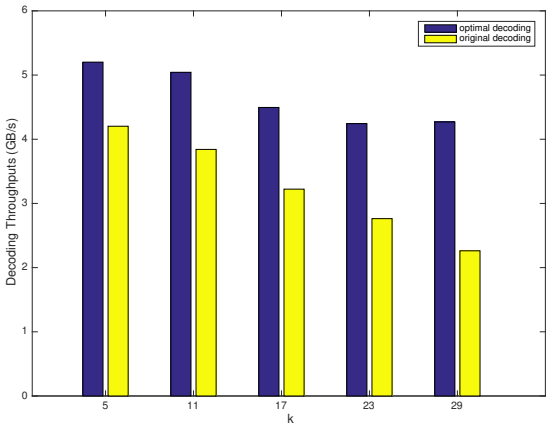
[4] D. A. Patterson, G. A. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *the 1988 ACM International Conference on Management of Data (SIGMOD'88)*, ser. SIGMOD '88. New York, NY, USA: ACM, 1988, pp. 109–116.

[5] 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 (CSUR)*, vol. 26, no. 2, pp. 145–185, 1994. [Online]. Available: http://dl.acm.org/citation.cfm?id=176981

[6] J. L. Hafner, "Weaver codes: highly fault tolerant erasure codes for storage systems," in *the 4th USENIX Conference on File and Storage Technologies (FAST'05)*. USENIX Association, 2005, pp. 211–224.

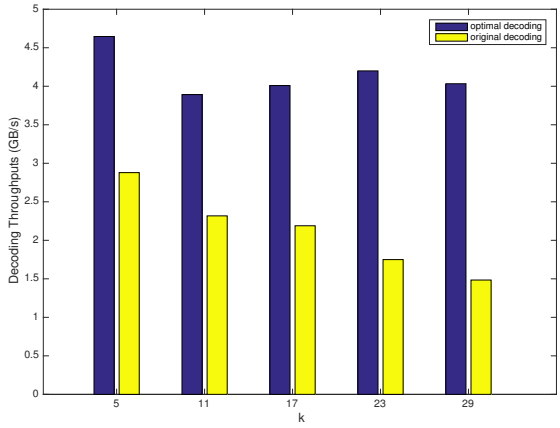


(a) element_size = 4KB

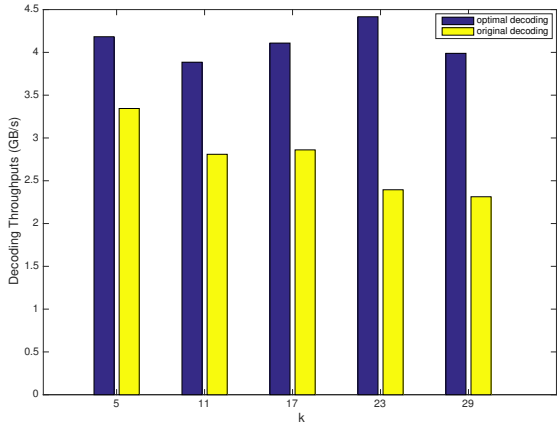


(b) element_size = 8KB

Fig. 12. Decoding throughputs of Liberation codes with p varing with k



(a) element_size = 4KB



(b) element_size = 8KB

Fig. 13. Decoding throughputs of Liberation codes with $p = 31$

[7] 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

[8] Z. Huang, H. Jiang, K. Zhou, Y. Zhao, and C. Wang, "Lowest density MDS array codes of distance 3," *IEEE Communications Letters*, vol. 19, no. 10, pp. 1670–1673, Oct 2015.

[9] C. Jin, H. Jiang, D. Feng, and L. Tian, "P-Code: A new RAID-6 code with optimal properties," in *the 23rd ACM International Conference on Supercomputing (ICS'09)*. ACM, 2009, pp. 360–369.

[10] L. Xu and J. Bruck, "X-Code: MDS array codes with optimal encoding," *IEEE Transactions on Information Theory*, vol. 45, pp. 272–276, 1999.

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

[12] Y. Wang, X. Yin, and X. Wang, "MDR codes: A new class of RAID-6 codes with optimal rebuilding and encoding," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 1008–1018, 2014.

[13] M. Blaum and R. M. Roth, "On lowest density MDS codes," *IEEE Transactions on Information Theory*, vol. 45, no. 1, pp. 46–59, 1999.

[14] 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.

[15] SNIA, "The SNIA dictionary," http://www.snia.org/education/dictionary/r, 2019, storage Networking Industry Association.

[16] M. Blaum, P. G. Farrell, van Tilborg, and C. A. Henk, "Chapter on array codes," *Handbook of Coding Theory*, vol. 2, pp. 1855–1909, 1998.

[17] J. S. Plank and C. Huang, "Tutorial: Erasure coding for storage applications," Slides presented at the 11th Usenix Conference on File and Storage Technologies (FAST'13), San Jose, February 2013.

[18] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, ser. North-Holland Mathematical Library (Book 16). Amsterdam, The Netherlands: North Holland Publishing Co., 1977.

[19] Z. Huang, H. Jiang, and K. Zhou, "An improved decoding algorithm for generalized RDP codes," *IEEE Communications Letters*, vol. PP, no. 99, pp. 1–1, 2016, iEEE Early Access Articles.

[20] Z. Huang, H. Jiang, H. Che, N. Xiao, and N. Li, "Efficient MDS array codes for correcting multiple column erasures," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 1602–1606.

[21] Y. Cassuto and J. Bruck, "Cyclic lowest density MDS array codes," *IEEE Transactions on Information Theory*, vol. 55, no. 4, pp. 1721–1729, 2009.

[22] Z. Huang, H. Jiang, K. Zhou, C. Wang, and Y. Zhao, "XI-Code: A family of practical lowest density MDS array codes of distance 4," *IEEE Transactions on Communications*, vol. 64, no. 7, pp. 2707–2718, July 2016.

[23] Z. Huang, H. Jiang, and N. Xiao, "Efficient lowest density MDS array codes of column distance 4," in *2017 IEEE International Symposium on Information Theory (ISIT)*, June 2017, pp. 834–838.

[24] 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, pp. 16:1–16:22, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1970340