H-Code: A Hybrid MDS Array Code to Optimize Partial Stripe Writes in RAID-6

Chentao Wu[1*], Shenggang Wan[2†], Xubin He[1*], Qiang Cao[2‡], and Changsheng Xie[2‡]
[1]Department of Electrical & Computer Engineering, Virginia Commonwealth University, Richmond, VA 23284, USA
[2]Wuhan National Lab for Optoelectronics, Huazhong University of Science & Technology, Wuhan, China 430074
*{wuc4,xhe2}@vcu.edu, †wanshenggang@gmail.com, ‡{caoqiang, cs_xie}@hust.edu.cn

***Abstract*—RAID-6 is widely used to tolerate concurrent failures of any two disks to provide a higher level of reliability with the support of erasure codes. Among many implementations, one class of codes called Maximum Distance Separable (MDS) codes aims to offer data protection against disk failures with optimal storage efficiency. Typical MDS codes contain horizontal and vertical codes. Due to the horizontal parity, in the case of *partial stripe write* (refers to I/O operations that write new data or update data to a subset of disks in an array) in a row, horizontal codes may get less I/O operations in most cases, but suffer from unbalanced I/O distribution. They also have limitation on high single write complexity. Vertical codes improve single write complexity compared to horizontal codes, while they still suffer from poor performance in partial stripe writes.**

In this paper, we propose a new XOR-based MDS array code, named Hybrid Code (H-Code), which optimizes partial stripe writes for RAID-6 by taking advantages of both horizontal and vertical codes. H-Code is a solution for an array of $(p+1)$ disks, where p is a prime number. Unlike other codes taking a dedicated anti-diagonal parity strip, H-Code uses a special anti-diagonal parity layout and distributes the anti-diagonal parity elements among disks in the array, which achieves a more balanced I/O distribution. On the other hand, the horizontal parity of H-Code ensures a partial stripe write to continuous data elements in a row share the same row parity chain, which can achieve optimal partial stripe write performance. Not only within a row but also within a stripe, H-Code offers optimal partial stripe write complexity to two continuous data elements and optimal partial stripe write performance among all MDS codes to the best of our knowledge. Specifically, compared to RDP and EVENODD codes, H-Code reduces I/O cost by up to 15.54% and 22.17%. Overall, H-code has optimal storage efficiency, optimal encoding/decoding computational complexity, optimal complexity of both single write and partial stripe write.

***Index Terms*—RAID-6; MDS Code; Partial Stripe Write; Performance Evaluation; Data Reliability**

I. Introduction

Redundant **A**rrays of **I**nexpensive (or **I**ndependent) **D**isks (**RAID**) [20] is an efficient approach to supply high reliability and high performance storage services with acceptable spatial and monetary cost. In recent years, RAID-6 has received much attention because it can tolerate concurrent failures of any two disks. It has been shown to be of increasing importance due to technology trends [24], [6] and the fact that the possibility of concurrent disk failures increases [27] [21] as the system scale grows.

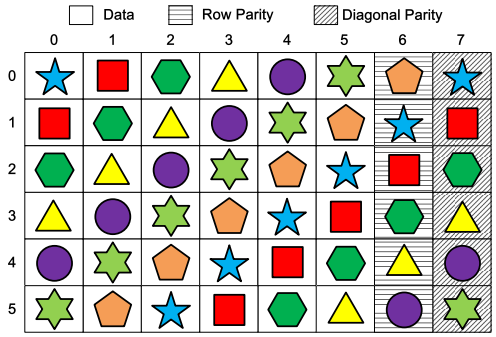
Among many implementations of RAID-6 based on various erasure coding technologies, one class of codes called **M**aximum **D**istance **S**eparable (**MDS**) codes [26], [3], [1], [6], [2], [23], [5], [33], [17], [10] aims to offer data protection against disk failures with given amount of redundancy. In other words, MDS codes offer optimal storage efficiency, leading to optimal *full stripe write*[1] complexity. Except for the full stripe write complexity, the complexity of partial stripe write and single write is also a concern of storage system designers [7], [19], [18], [4], [11]. Typical MDS codes can be further categorized into horizontal codes [26], [3], [1], [6], [2], [23] and vertical codes [5], [33], [17].

A typical horizontal RAID-6 storage system is composed of k+2 disk drives. The first k disk drives are used to store original data, and the last two, named P and Q, are used as parity disk drives. Due to the P parity, in the case of partial stripe write in a row, horizontal codes might get fewer I/O operations most of the time, but suffer from unbalanced I/O distribution. Let's take an example of RDP codes [6] whose diagonal parity layout is shown in Figure 1(a). If there is a partial stripe write to w continuous elements[2] in a row as shown in Figure 1(b), it results in w reads and w writes to the Q parity disk (disk 7), but one read and one write to other disks (disk 0, 1, 2, 3 and 6). As the system scale grows, this problem cannot be resolved by shifting the stripes' parity strips among all the disks just as RAID-5. In addition, horizontal codes also have limitation on high single write complexity. Blaum *et al.* have proved that with a i-row-j-column matrix of data elements, at least $(i*j+j-1)$ data elements participate in the generation of Q parities [2]. Thus the cost of a single block write in horizontal codes requires more than two additional writes on average, which is the lower bound of a theoretically ideal RAID-6 codes.

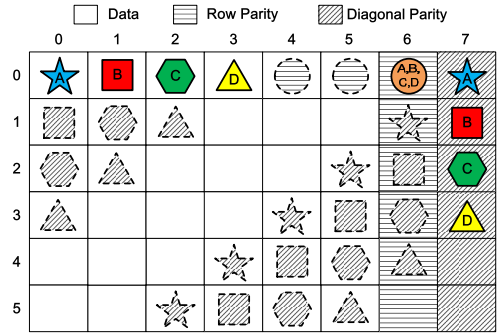
Vertical codes, such as X-Code [33], Cyclic code [5], and P-Code [17], typically offer good single write complexity and encoding/decoding computational complexity as well as high storage efficiency. With special data/parity layout, they do not adopt row parity. In vertical codes, partial stripe write to multiple data elements in a row involves the generation of different parity elements. Let's take an example of X-Code whose anti-diagonal parity layout is shown in Figure 2(a).

[1] In this paper, "write" can be a new write or an update.

[2] Thereafter "continuous" means logically continuous among the disk arrays in encoding/decoding.

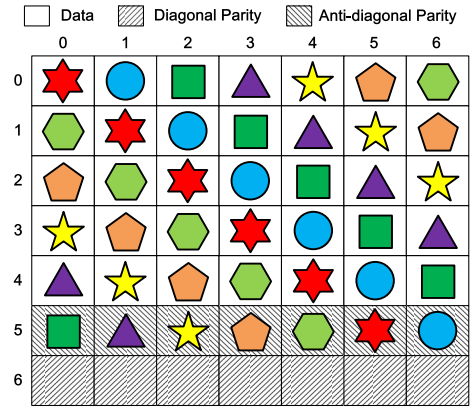


(a) Diagonal parity layout of RDP Code with $p+1$ disks ($p=7$): a diagonal element can be calculated by XOR operations among the corresponding elements, e.g., $C_{0,7} = C_{0,0} \oplus C_{5,2} \oplus C_{4,3} \oplus C_{3,4} \oplus C_{2,5} \oplus C_{1,6}$.

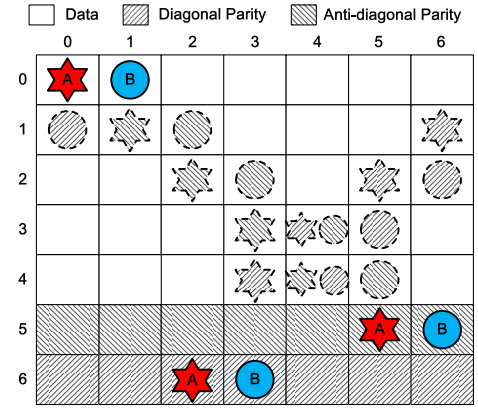


(b) A partial stripe write to 4 continuous data elements: A, B, C and D. The I/O operations in disks 0, 1, 2, 3 and 6 are all 1 read and 1 write, while in disk 7 are 4 reads and 4 writes. It shows that the workload in Disk 7 is very high, which may lead to a sharp decrease of reliability and performance of the system.

Fig. 1. Partial stripe write problem in RDP code for an 8-disk array ($p=7$).



(a) Anti-diagonal parity layout of X-Code with p disks ($p=7$): an anti-diagonal element can be calculated by XOR operations among the corresponding data elements, e.g., $C_{5,0} = C_{0,2} \oplus C_{1,3} \oplus C_{2,4} \oplus C_{3,5} \oplus C_{4,6}$.



(b) A partial stripe write to two continuous data elements: A and B. The I/O operations are $(2+4)$ reads and $(2+4)$ writes.

Fig. 2. Partial stripe write problem in X-Code for a 7-disk array.

Consider the following scenario as described in Figure 2(b). There is a partial stripe write to two data elements in the first row of a X-Code based 7-disk RAID-6 matrix. We notice that the two elements are in four different parity chains. Therefore, this partial stripe write results in $(2+4)$ reads and $(2+4)$ writes, for a total of 12 I/O operations. If the codes adopt row parity, the two elements might be only in three different parity chains. It can result in $(2+3)$ reads and $(2+3)$ writes, or 10 I/O operations in total. In other words, it reduces two I/O operations by adopting row parity. We evaluate the partial stripe write complexity in general and only focus on the short partial stripe write, which effects no more than $(n-3)$ data elements for an array of n disks. Consider the following case, a partial stripe write to w continuous data elements in v parity chains. Typically, it results in one read and one write to each data element, and one read and one write to each parity element.

Therefore, the number of disk I/O operations (denoted by S_w) are,

$$S_w = 2 * (w + v) \tag{1}$$

When the number of data elements affected by a partial stripe write in a row increases, the disk I/O reduced by adopting a row parity will increase. Similar problem also exists in Cyclic Code and P-code. However, almost all the horizontal codes adopt row parity, which might cause less partial stripe write cost. Jin *et al.* [17] mentioned a tweaked RDP code, which is a semi-vertical code composed of row parity and diagonal parity. However, for its individual diagonal parity disk, it still suffers from the unbalanced I/O distribution caused by partial stripe write to multiple data elements in a row as same as the horizontal codes.

We propose a novel XOR-based RAID-6 code, named **Hybrid Code (H-Code)**, which takes advantages of both horizontal and vertical codes. The parities in H-Code are classical row parity and anti-diagonal parity. H-code does not have a dedicated anti-diagonal parity strip, while it distributes the anti-diagonal parity elements among disks in the array. Its horizontal parity ensures a partial stripe write to continuous data elements in a row share the same row parity chain, which

achieves optimal partial stripe write performance. Depending on the number of disks in an MDS array, we design H-Code, which is a solution for n disks ($n = p+1$), where p is a prime number.

We make the following contributions in this paper:

- We propose a novel and efficient XOR-based RAID-6 code (H-Code) to take advantages of both horizontal and vertical MDS codes and offer optimal partial stripe write performance compared to existing MDS codes.
- We quantitatively analyze partial stripe write performance of H-Code and prove that H-Code has not only the optimal property demonstrated by traditional vertical MDS codes including storage efficiency, encoding/decoding computational complexity and single write complexity, but also the optimized partial stripe write complexity to multiple data elements.

The rest of this paper continues as follows: Section II briefly overviews the background and related work. The design of H-Code is described in detail in Section III. Property analysis and evaluation are given in Section IV and Section V. Finally we conclude the paper in Section VI.

II. Background and Related Work

Reliability has been a critical issue for storage systems since data are extremely important for today's information business. Among many solutions to provide storage reliability, RAID-6 is known to be able to tolerate concurrent failures of any two disks. Researchers have presented many RAID-6 implementations based on various erasure coding technologies, such as Reed-Solomon code [26], Cauchy Reed-Solomon code [3], EVENODD code [1], RDP code [6], Blaum-Roth code [2], Liberation code [23], Liber8tion code [22], Cyclic code [5], X-Code [33], and P-Code [17]. These codes are **M**aximum **D**istance **S**eparable (**MDS**) codes, which offer protection against disk failures with given amount of redundancy [5]. RSL-Code [8], RL-Code [9] and STAR [16] are MDS codes tolerating concurrent failures of three disks. In addition to MDS codes, some non-MDS codes, such as WEAVER [13], HOVER [14], MEL code [31], Pyramid code [15], Flat XOR-Code [12] and Code-M [29], also offer resistance to concurrent failures of any two disks, but they have higher storage overhead. There are also some approaches to improve the efficiency of different codes [28][30][32]. In this paper, we focus on MDS codes, which can be further divided into two categories: horizontal codes and vertical codes.

A. RAID-6 based on Horizontal MDS Codes

Reed-Solomon code [26] is based on addition and multiply operations over certain finite fields $GF(2^{\mu})$. The addition in Reed-Solomon code can be implemented by XOR operation, but the multiply is much more complex. Due to high computational complexity, Reed-Solomon code is not very practical. Cauchy Reed-Solomon code addresses this problem and improves Reed-Solomon code by changing the multiply operations over $GF(2^{\mu})$ into additional XOR operations.

Unlike the above generic erasure coding technologies, EVENODD [1] is a special erasure coding technology only for RAID-6. It is composed of two types of parity: the P parity, which is similar to the horizontal parity in RAID-4, and the Q parity, which is generated by the elements on the diagonals. RDP [6] is another special erasure coding technology dedicated for RAID-6. The P parity of RDP is just the same as that of EVENODD. However, it uses a different way to construct the Q parity to improve construction and reconstruction computational complexity.

There is a special class of erasure coding technologies called *lowest density codes*. Blaum *et al.* [2] point out that in a typical horizontal code for RAID-6, if the P parity is fixed to be horizontal parity, then with i-row-j-column matrix of data elements there must be at least $(i * j + j - 1)$ data elements joining in the generation of the Q parities to achieve the lowest density. Blaum-Roth, Liberation, and Liber8tion codes are all lowest density codes. Compared to other horizontal codes for RAID-6, the lowest density codes share a common advantage that they have the near-optimal single write complexity.

However, horizontal codes have some limitations. First, the single write complexity is one of the weaknesses for most horizontal MDS RAID-6 codes. Second, partial stripe write to continuous data elements in a row of a horizontal MDS RAID-6 codes, which may result in unbalanced I/O distribution, although the horizontal parity can reduce I/O operations in case of partial stripe write to data elements in a row.

B. RAID-6 based on Vertical MDS Codes

X-Code, Cyclic code, and P-Code are vertical codes and their parities are dispersed over all disks instead of dedicated redundant disks. This layout achieves better encoding/decoding computational complexity, and improved single write complexity.

X-Code [33] uses diagonal parity and anti-diagonal parity and the number of columns (or disks) in X-Code must be a prime number.

Cyclic code [5] offers a scheme to support more column number settings with vertical MDS RAID-6 codes. The column number of Cyclic Code is typically $(p-1)$ or $2*(p-1)$.

P-Code [17] is another example of vertical code. Construction rule in P-Code is very simple. The columns are labeled with an integer from 1 to $(p-1)$. In P-Code, the parity elements are deployed in the first row, and the data elements are in the remaining rows. The parity rule is that each data element takes part in the generation of two parity elements, where the sum of the column numbers of the two parity elements mod p is equal to the data element's column number.

From the constructions of above vertical codes, we observe that they suffer from high complexity of partial stripe write to continuous data.

Figures 1 and 2 show the parity construction of two typical MDS codes: a horizontal code that is an RDP code and a vertical code that is an X-Code. As explained in Section I, these figures show that RAID-6 schemes based on MDS codes

TABLE I
SYMBOLS OF H-CODE

Parameters & Symbols	Description
n	number of disks in a disk array
p	a prime number
i, r	row ID
j	column ID or disk ID
$C_{i,j}$	element at the ith row and jth column
f_1, f_2 ($f_1 < f_2$)	two random failed columns with IDs f_1 and f_2
$\sum$ (e.g., $\sum_{j=0}^{5} C_{i,j}$)	XOR operations between/among elements ($C_{i,0} \oplus C_{i,1} \oplus \cdots \oplus C_{i,5}$)
$\langle\ \rangle$ (e.g., $\langle i \rangle_p$)	modular arithmetic (i mod p)
w ($2 \leq w \leq n-3$)	number of continuous data elements in a partial stripe write
N_s	total access frequency of all stripe writes in the ideal sequence
$F(C_{i,j})$	access frequency of a partial stripe write with beginning data element $C_{i,j}$
$P(C_{i,j})$	access probability of a partial stripe write with beginning data element $C_{i,j}$
$S_w(C_{i,j})$	number of I/O operations caused by a partial stripe write to w continuous data elements with beginning element $C_{i,j}$
$S_{max.}(w)$	maximum number of I/O operations of a partial stripe write to w continuous data elements
$S_{avg.}(w)$	average number of I/O operations of a partial stripe write to w continuous data elements
$S_w^j(C_{i,j})$	number of I/O operation in column j of a partial stripe write to w continuous data elements with beginning element $C_{i,j}$
$S_{avg.}^j(w)$	average number of I/O operations in column j of a partial stripe write to w continuous data elements

suffer from a common disadvantage of partial stripe write to continuous data elements.

III. H-CODE FOR AN ARRAY OF $p+1$ DISKS

To overcome the shortcomings of vertical and horizontal MDS codes, we present a hybrid MDS code scheme, named H-Code, to take advantage of both vertical and horizontal codes and is a solution for n disks ($n = p+1$), where p is a prime number. The symbols of H-Code are summarized in Table I.

A. Data/Parity Layout and Encoding of H-Code

H-Code is represented by a $(p-1)$-row-$(p+1)$-column matrix with a total of $(p-1)*(p+1)$ elements. There are three types of elements in the matrix: ***data elements***, ***horizontal parity elements***, and ***anti-diagonal parity elements***. Assume $C_{i,j}$ ($0 \leq i \leq p-2$, $0 \leq j \leq p$) represents the element at the ith row and the jth column. The last column (column p) is used for horizontal parity. Excluding the first (column 0) and the last (column p) columns, the remaining matrix is a $(p-1)$-row-$(p-1)$-column square matrix. H-Code uses the anti-diagonal part of this square matrix to represent anti-diagonal parity.

Horizontal parity and anti-diagonal parity elements of H-Code are constructed based on the following encoding equations.

Horizontal parity:

$$C_{i,p} = \sum_{j=0}^{p-1} C_{i,j} \quad (j \neq i+1) \tag{2}$$

Anti-diagonal parity:

$$C_{i,i+1} = \sum_{j=0}^{p-1} C_{\langle p-2-i+j \rangle_p, j} \quad (j \neq i+1) \tag{3}$$

Figure 3 shows an example of H-Code for an 8-disk array ($p = 7$). It is a 6-row-8-column matrix. Column 7 is used for horizontal parity and the anti-diagonal elements ($C_{0,1}$, $C_{1,2}$, $C_{2,3}$, etc.) are used for anti-diagonal parity.

The horizontal parity encoding of H-Code is shown in Figure 3(a). We use different shapes to indicate different sets of horizontal elements and the corresponding data elements. Based on Equation 2, we calculate all horizontal elements. For example, the horizontal element $C_{0,7}$ can be calculated by $C_{0,0} \oplus C_{0,2} \oplus C_{0,3} \oplus C_{0,4} \oplus C_{0,5} \oplus C_{0,6}$. The element $C_{0,1}$ is not involved in this example because of $j = i+1$.

The anti-diagonal parity encoding of H-Code is given in Figure 3(b). The anti-diagonal elements and their corresponding data elements are also distinguished by various shapes. According to Equation 3, the anti-diagonal elements can be calculated through modular arithmetic and XOR operations. For example, to calculate the anti-diagonal element $C_{1,2}$ ($i = 1$), first we should get the proper data elements ($C_{\langle p-2-i+j \rangle_p, j}$). If $j = 0$, by using Equation 3, $p-2-i+j = 4$ and then $\langle 4 \rangle_p = 4$, we get the first data element $C_{4,0}$. The following data elements which take part in XOR operations can be calculated similarly (the following data elements are $C_{5,1}$, $C_{0,3}$, $C_{1,4}$, $C_{2,5}$ and $C_{3,6}$). Second, the corresponding anti-diagonal element ($C_{1,2}$) is constructed by performing an XOR operation on these data elements, i.e., $C_{1,2} = C_{4,0} \oplus C_{5,1} \oplus C_{0,3} \oplus C_{1,4} \oplus C_{2,5} \oplus C_{3,6}$.

B. Construction Process

Based on the above data/parity layout and encoding scheme, the construction process of H-Code is straightforward.

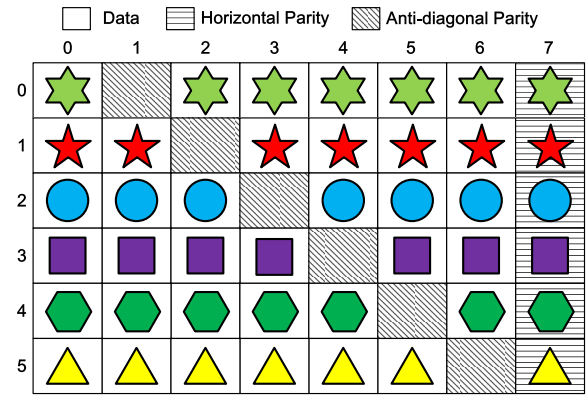
- Label all data elements.
- Calculate both horizontal and anti-diagonal parity elements according to Equations 2 and 3.

C. Proof of Correctness

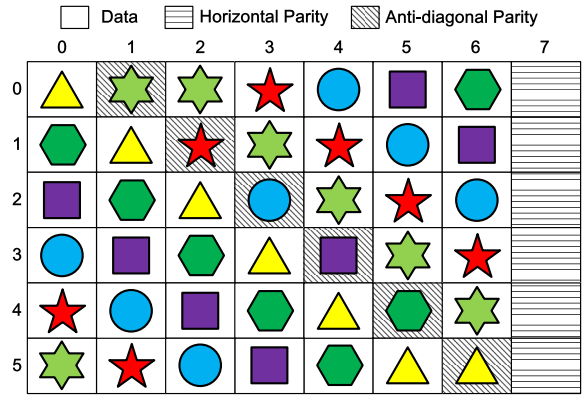
To prove that H-Code is correct, we consider one stripe. The reconstruction of multiple stripes is just a matter of scale and similar to the reconstruction of one stripe. In a stripe, we have the following lemma and theorem,

Lemma 1: We can find a sequence of a two-integer tuple (T_k, T'_k) where

$$T_k = \left\langle p - 1 + \frac{k+1+\frac{1+(-1)^k}{2}}{2}(f_2 - f_1) \right\rangle_p,$$
$$T'_k = \frac{1+(-1)^k}{2} f_1 + \frac{1+(-1)^{k+1}}{2} f_2 \quad (k = 0, 1, \cdots, 2p-1)$$



(a) Horizontal parity coding of H-Code: a horizontal element can be calculated by XOR operations among the corresponding data elements in the same row. For example, $C_{0,7} = C_{0,0} \oplus C_{0,2} \oplus C_{0,3} \oplus C_{0,4} \oplus C_{0,5} \oplus C_{0,6}$.



(b) Anti-diagonal parity coding of H-Code: an anti-diagonal element can be calculated by XOR operations among the corresponding data elements in all columns (except its column and column p). For example, $C_{1,2} = C_{4,0} \oplus C_{5,1} \oplus C_{0,3} \oplus C_{1,4} \oplus C_{2,5} \oplus C_{3,6}$.

Fig. 3. H-Code ($p = 7$).

with a prime number of p and $0 < f_2 - f_1 < p$, the endpoints are $(p-1, f_1)$ and $(p-1, f_2)$, and all two-integer tuples $(0, f_1)$, $(0, f_2)$, $\cdots$, $(p-1, f_1)$, $(p-1, f_2)$ occur exactly once in the sequence. Similar proof of this lemma can be found in many literatures in RAID-6 codes such as [1], [6], [33], [17].

Theorem 1: A $(p-1)$-row-$(p+1)$-column stripe constructed according to the formal description of H-Code can be reconstructed under concurrent failures from any two columns.

Proof: There are two cases of double failures, depending on whether column 0 fails or not.

Case I: column 0 doesn't fail.

There are further two subcases, depending on whether the horizontal parity column fails or not.

Case I-I: Double failures, one is from the horizontal parity column, and the other is from the anti-diagonal parity.

From the construction of H-Code, any two of the lost data elements and parity element are not in a same parity chain. Therefore, each of them can be recovered through the anti-diagonal parity chains. When all lost data elements are recovered, the horizontal parity elements can be reconstructed using the above Equation 2.

Case I-II: Double failures of any two columns other than the horizontal parity of the stripe.

Each column j, in the anti-diagonal parity part of the stripe, intersects all horizontal parity chains except the horizontal parity chain in the $j-1$th row. Therefore, each column misses a different horizontal parity chain.

First, we make an assumption that there is a pseudo row under the last row of H-Code matrix as shown in Figure 4(a). Each element of the additional row is all-zero-bit element and takes part into the generation of parity of its anti-diagonal, where it does not change the original value of anti-diagonal parity elements. This additional all-zero-bit element just participates in the generation of the parity element in its column. We assume that the two failed columns are f_1 and f_2, where $0 < f_1 < f_2 < p$.

From the construction of H-Code, each horizontal parity chain in the ith row intersects all columns in anti-diagonal part of the stripe except the $(i+1)$th column. Any two anti-diagonal parity elements cannot be placed in the same row. For any two concurrent failed columns f_1 and f_2, the two horizontal parity chains that are not intersected by both columns are in the $(f_1 - 1)$th row and the $(f_2 - 1)$th row. Since each of these horizontal parity chains only misses one data element, the missing element can be reconstructed along that horizontal parity chain with its horizontal parity. Since the horizontal parity column does not fail which means all horizontal parity elements are available, we can start reconstruction process from the data element on each of the two missing columns using the horizontal parity.

For the failed columns f_1 and f_2, if a data element C_{i,f_2} on column f_2 can be reconstructed from the horizontal parity in horizontal parity chain in the ith row, we can reconstruct the missing data element $C_{\langle i+f_1-f_2\rangle_p,f_1}$ on the same anti-diagonal parity chain if we have its anti-diagonal parity element. Similarly, a data element C_{i,f_1} in column f_1 can be reconstructed from the horizontal parity in horizontal parity chain in the ith row, we can reconstruct the missing data element $C_{\langle i+f_2-f_1\rangle_p,f_2}$ on the same anti-diagonal parity chain if we have its anti-diagonal parity element parity element.

From the above discussion, the two missing anti-diagonal parity elements are just the two endpoints we mentioned in Lemma 1. If there are no missing parity elements, we start the reconstruction process from data element C_{f_1-1,f_2} on the f_2th column to the corresponding endpoint (element C_{p-1,f_1} on the f_1th column). In this reconstruction process, all data elements can be reconstructed and the reconstruction sequence is based on the sequence of the two-integer tuple in Lemma 1. Similarly, with no missing parity elements, we start the reconstruction process from data element C_{f_2-1,f_1} on the f_1th

column to the corresponding endpoint (element C_{p-1,f_2} on the f_2th column).

However, the two missing anti-diagonal parity elements are not reconstructed in this case. After we start the reconstruction from the two data elements C_{f_1-1,f_2} and C_{f_2-1,f_1}, the missing anti-diagonal parity elements C_{p-1,f_1} and C_{p-1,f_2} cannot be recovered, because their corresponding horizontal parity and anti-diagonal parity are both missing. Actually, we do not need to reconstruct these two elements for they are not really in our code matrix.

In summary, all missing data elements are recoverable. After all data elements are recovered, we can reconstruct the two missing anti-diagonal parity elements.

Case II: column 0 fails.

This case is similar to *Case I*. The difference is that in *subcase II-II*, there is only one reconstruction sequence in reconstruction process. This process starts at $C_{f_2-1,0}$ and all lost data elements can be recovered. ■

D. Reconstruction

We first consider how to recover a missing data element since any missing parity element can be recovered based on Equations 2 and 3. If we save the horizontal parity element and the related $p-2$ data elements, we can recover the missing data element (assume it's C_{i,f_1} in column f_1 and $0 \le f_1 \le p-1$) using the following equation,

$$C_{i,f_1} = \sum_{j=0}^{p} C_{i,j} \quad (j \neq i+1 \quad and \quad j \neq f_1) \tag{4}$$

If there exists an anti-diagonal parity element and its $p-2$ data elements, to recover the data element (C_{i,f_1}), first we should find the corresponding anti-diagonal parity element. Assume it is in row r and this anti-diagonal parity element can be represented by $C_{r,r+1}$ based on Equation 3, we have,

$$r = \langle p-2-i+f_1 \rangle_p \tag{5}$$

And then according to Equation 3, the lost data element can be recovered,

$$\begin{aligned} C_{i,f_1} &= C_{r,r+1} \oplus \sum_{j=0}^{p-1} C_{\langle i-f_1+j \rangle_p, j} \\ &(j \neq f_1 \quad and \quad j \neq \langle p-1-i+f_1 \rangle_p) \end{aligned} \tag{6}$$

Based on Equations 2 to 6, we can easily recover the elements with single disk failure. If two disks fail (for example, column f_1 and column f_2, $0 \le f_1 < f_2 \le p$), based on Theorem 1, we have our reconstruction algorithm of H-Code, shown in Figure 5.

As the proof of Theorem 1, there are two cases in our reconstruction algorithm of H-Code: failure in column 0 or not. Each has two subcases, *subcases I-I* and *II-I* focus on the scenario where at least one failure involves a horizontal parity column while in *subcases I-II* and *II-II*, failures don't involve the horizontal parity. The reconstruction examples of *subcases I-II* and *II-II* are shown in Figure 4(a) and Figure 4(b), which are situations with different numbers of recovery chains.

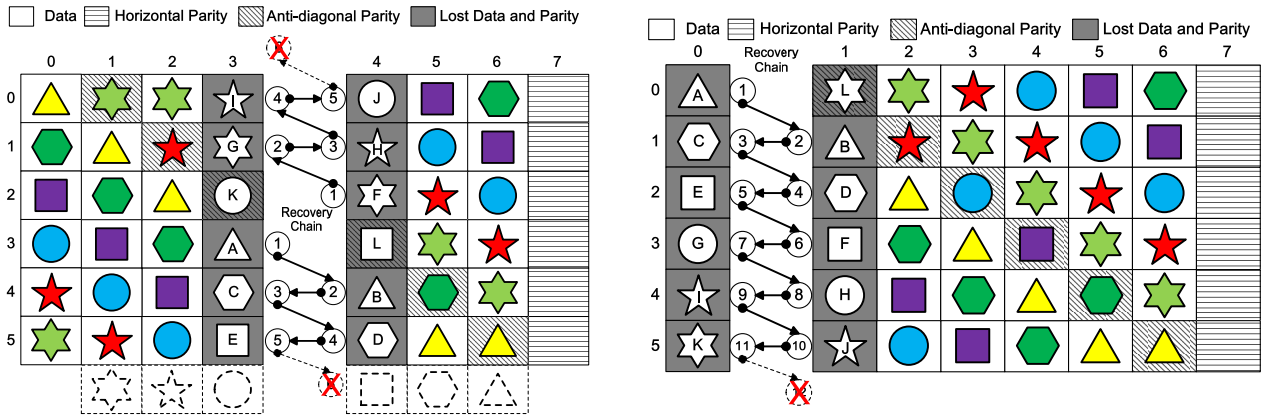
Algorithm 1: Reconstruction Algorithm of H-Code

Step 1: Identify the double failure columns: f_1 and f_2 ($f_1 < f_2$).
Step 2: Start reconstruction process and recover the lost data and parity elements.
switch $0 \le f_1 < f_2 \le p$ **do**
- **case** *I:* $f_1 \neq 0$ *(column 0 is saved)*
  - **case** *I-I:* $f_2 = p$ *(horizontal parity column is lost)*
    - **Step 2-I-IA:** Recover the lost data elements in colomn f_1.
    - **repeat**
      - Compute the lost data elements (C_{i,f_1}, $i \neq f_1 - 1$) based on Equations 5 and 6.
    - **until** *all lost data elements are recovered.*
    - **Step 2-I-IB:** Recover the lost anti-diagonal parity element (C_{f_1-1,f_1}) based on Equation 3.
    - **Step 2-I-IC:** Recover the lost horizontal parity elements in colomn f_2.
    - **repeat**
      - Compute the lost horizontal parity elements (C_{i,f_2}) based on Equation 2.
    - **until** *all lost horizontal parity elements are recovered.*
  - **case** *I-II:* $f_2 \neq p$ *(horizontal parity column is saved)*
    - **Step 2-I-IIA:** Compute two starting points (C_{f_2-1,f_1} and C_{f_1-1,f_2}) of the recovery chains based on Equation 4.
    - **Step 2-I-IIB:** Recover the lost data elements in the two recovery chains.
    - **Two cases start synchronously:**
    - **case** *starting point is* C_{f_2-1,f_1} **repeat**
      - (1) Compute the next lost data element (in colomn f_2) in the recovery chain based on Equations 5 and 6;
      - (2) Then compute the next lost data element (in colomn f_1) in the recovery chain based on Equation 4.
    - **until** *at the endpoint of the recovery chain.*
    - **case** *starting point is* C_{f_1-1,f_2} **repeat**
      - (1) Compute the next lost data element (in colomn f_1) in the recovery chain based on Equations 5 and 6;
      - (2) Then compute the next lost data element (in colomn f_2) in the recovery chain based on Equation 4.
    - **until** *at the endpoint of the recovery chain.*
    - **Step 2-I-IIC:** Recover the lost anti-diagonal parity element in colomn f_1 and f_2.
    - **repeat**
      - Compute the lost anti-diagonal parity elements (C_{f_1-1,f_1} and C_{f_2-1,f_2}) based on Equation 3.
    - **until** *all lost anti-diagonal parity elements are recovered.*
- **case** *II:* $f_1 = 0$ *(column 0 is lost)*
  - **case** *II-I:* $f_2 = p$ *(horizontal parity column is lost)*
    - **Step 2-II-IA:** Recover the lost data elements in colomn 0.
    - **repeat**
      - Compute the lost data elements ($C_{i,0}$) based on Equations 5 and 6.
    - **until** *all lost data elements are recovered.*
    - **Step 2-II-IB:** Recover the lost horizontal parity elements in colomn f_2.
    - **repeat**
      - Compute the lost horizontal parity elements (C_{i,f_2}) based on Equation 2.
    - **until** *all lost horizontal parity elements are recovered.*
  - **case** *II-II:* $f_2 \neq p$ *(horizontal parity column is saved)*
    - **Step 2-II-IIA:** Compute the starting point of the recovery chain ($C_{f_2-1,0}$) based on Equation 4.
    - **Step 2-II-IIB:** Recover the lost data elements in the recovery chain.
    - **repeat**
      - (1) Compute the next lost data element in colomn f_2 based on Equations 5 and 6;
      - (2) Then compute the next lost data element in colomn 0 based on Equation 4.
    - **until** *at the endpoint of the recovery chain.*
    - **Step 2-II-IIC:** Recover the lost diagonal parity element in colomn f_2.
    - Compute the lost anti-diagonal parity element (C_{f_2-1,f_2}) based on Equation 3.

Fig. 5. Reconstruction Algorithm of H-Code.

IV. Property Analysis

In this section, we first prove that H-Code shares some optimal properties as other vertical codes, including: optimal storage efficiency, optimal encoding/decoding computational



(a) Reconstruction by two recovery chains (there are double failures in columns 3 and 4): First we identify the two starting points of recovery chain: data elements A and F. Second we reconstruct data elements according to the corresponding recovery chains until they reach the endpoints (data elements E and J). The next anti-diagonal elements after E and J do not exist (C_{p-1,f_1} and C_{p-1,f_2} in the proof of Theorem 1, we use two "Xs" here), so the recovery chains end. The orders to recover data elements are: one is A→B→C→D→E, the other is F→G→H→I→J. Finally we reconstruct anti-diagonal parity elements K and L according to Equation 3.

(b) Reconstruction by one recovery chain (there are double failures in columns 0 and 1): First we identify the starting point of recovery chain: data element A. Second we reconstruct data elements according to the corresponding recovery chain until it reaches endpoint (data element K). The next anti-diagonal element after K does not exist (we use an "X" here), so the recovery chain ends. The order to recover data elements is: A→B→C→D→E→F→G→H→I→J→K. Finally we reconstruct the anti-diagonal parity element L according to Equation 3.

Fig. 4. Reconstruction Process of H-Code.

complexity, optimal single write cost. Then, we prove that H-Code has optimal complexity of partial stripe write to two data elements in the same row. Furthermore, H-Code has optimal complexity of partial stripe write to two data elements in a stripe. Finally, we evaluate partial stripe write cost of different codes in two aspects: in the same row and across two rows.

A. Optimal Property of H-Code

Vertical codes have the optimal storage efficiency, optimal encoding/decoding computational complexity, optimal single write complexity [33], [5], [17]. We will prove that H-Code shares the optimal property as other vertical codes.

*1) **Optimal Storage Efficiency:*** From the proof of H-Code's correctness, H-Code is a MDS code. Since all MDS codes have optimal storage efficiency [33], [5], [17], H-Code is storage efficient. Because H-Code doesn't use a dedicated anti-diagonal strip, it will not suffer from the unbalanced I/O distribution as evidenced by our results shown in Figure 11, which will be discussed in more detail later. By shifting the stripes' horizontal parity strips among all the disks just as RAID-5, H-Code does not suffer from the intensive I/O operations on dedicated parity disk caused by random writes among stripes, either.

*2) **Optimal Encoding/Decoding Computational Complexity:*** From the construction of H-Code, to generate all the $2*(p-1)$ parity elements in a $(p-1)$-row-$(p+1)$-column constructed H-Code, each of the remaining $(p-1)*(p-1)$ data elements needs to take part into two XOR operations. Thus, the encoding computational complexity of H-Code is $[2*(p-1)*(p-1)-2*(p-1)]/[(p-1)*(p-1)]$ XOR operations per data element on average. To reconstruct $2*(p-1)$ failed elements in the case of double disk failures, it need use $2*(p-1)$ parity chains. Every parity chain in H-Code has the same length of $(p-1)$. Thus, the decoding computational complexity of H-Code is $(p-3)$ XOR operations per lost element on average. P-Code [17] has already proved that an i-row-j-column constructed code with x data elements, has an optimal encoding computational complexity of $(3x-i*j)/x$ XOR operations per data element on average, and decoding computational complexity of $(3x-i*j)/(i*j-x)$ XOR operations per lost element on average. Therefore, H-Code's encoding/decoding computational complexity is optimal.

*3) **Optimal Single Write Property:*** From the construction of H-Code, each of the data elements takes part into the generation of two and only two parity elements. Therefore, a single write on one data element in H-Code only causes one additional write on each of the two parity elements, which has been proved to be optimal in a double disk failures tolerated codes [33], [5], [17].

B. Partial Stripe Writes to Two Continuous Data Elements in H-Code

Now, we prove that the cost of any partial stripe write to two continuous data elements of H-Code is optimal among all vertical lowest density MDS codes.

Theorem 2: Any two data elements in a stripe of a lowest density vertical MDS code are in at least three parity chains.

Proof: From [33], [5], [17], it has been proved that, in a stripe of a lowest density MDS code, any one data element takes part into the generation of two and only two parity elements. Assume there exist two data elements in two or less parity chains. Consider the following case: both data elements

are lost when double disk failures occur. Now, these two lost data elements are unrecoverable, because all parity chains in the code include either messages of both of these two data elements or none of them. Thus, the assumption is invalid. Therefore, in a stripe of a lowest density MDS code, any two data elements should be in at least three parity chains. ■

From the construction, any two continuous data elements in the same row of H-Code share a same horizontal parity and must not share the same anti-diagonal parity. In other words, any two continuous data elements in the same row are in three different parity chains including a horizontal parity chain and two different anti-diagonal parity chains. From Equation 1, any partial stripe write to two continuous data elements in a row of H-Code causes $2 * (2 + 3) = 10$ I/O operations.

Furthermore, as shown in Figure 6, from the construction of H-Code, any two continuous data elements across two different rows share the same anti-diagonal parity and must not share a same horizontal parity. From Equation 1, in this condition H-Code also causes 10 I/O operations per partial stripe write.

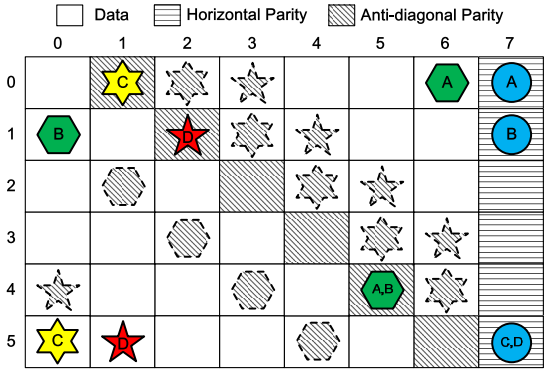


Fig. 6. Partial stripe writes to two continuous data elements in H-Code for an 8-disk array (a partial stripe write to data elements A and B in two different rows, and the other partial stripe write to data elements C and D in the same row. In these two cases, there are only 3 parity elements modified for each partial stripe write, which shows that our H-Code reduces partial stripe write cost and improves the performance of storage system).

In summary, any partial stripe write to two continuous data elements of H-Code are in three different parity chains. This is the lowest bound we proved in Theorem 2. From Equation 1 and Theorem 2, the cost of any partial stripe write to two continuous data elements locally of H-Code is 10 I/O operations, which is optimal.

C. Different Cases of Partial Stripe Writes to w Continuous Data Elements

There are two cases for a partial stripe write to w continuous data elements: one is the w written continuous data elements in the same row, the other is these data elements across rows.

The case of partial stripe writes to w continuous data elements ($2 \leq w \leq n-3$) in the same row is very simple, we only need to calculate the number of parity chains based on Equation 1. For example, when a partial stripe write to w continuous data elements in H-code, these data elements to be written share one horizontal parity chain, but are in w different anti-diagonal parity chains. According to Equation 1, the total I/O operations are $(4w + 2)$.

However, for a partial stripe write to w continuous data elements ($2 \leq w \leq n-3$), there are many scenarios where partial stripe writes are crossing different rows (will be discussed in Section V), which are not as simple as partial stripe writes to two continuous data elements. For example, as shown in Figure 1, if $w = 3$, the cost of a partial stripe write to three elements $C_{0,4}C_{0,5}C_{1,0}$ in RDP is different from a partial stripe write to $C_{1,4}C_{1,5}C_{2,0}$.

V. Performance Evaluation

In this section, we give our evaluations to demonstrate the effectiveness of our H-Code for partial stripe writes to w continuous data elements ($2 \leq w \leq n-3$).

A. Evaluation Methodology

We compare H-Code with following popular codes in typical scenarios (when $p = 5$ and $p = 7$):

(1) **Codes for $p-1$ disks**: P-Code-1[3] [17] and Cyclic code [5];

(2) **Codes for p disks**: X-Code [33] and P-Code-2 [17];

(3) **Codes for $p+1$ disks**: H-Code and RDP code [6];

(4) **Codes for $p+2$ disks**: EVENODD code [1].

To reflect the status of partial stripe writes among different codes, we envision an ideal sequence to partial stripe writes as follows,

For each data element[4], it is treated as the beginning written element at least once in a partial stripe write to w continuous data elements ($2 \leq w \leq n-3$, including partial stripe writes in the same row and across two rows). If there is no data element at the end of a stripe, the data element at the beginning of the stripe will be written[5].

Based on the description of ideal sequence, for H-Code shown in Figure 3 (which has $(p-1)^2$ total data elements in a stripe), the ideal sequence to w partial stripe writes is: $C_{0,0}C_{0,2}\cdots$, $C_{0,2}C_{0,3}\cdots$, $C_{0,3}C_{0,4}\cdots$, and so on, $\cdots\cdots$, the last partial stripe write in this sequence is $C_{p-2,p-2}C_{0,0}\cdots$.

We use two types of access patterns based on this ideal sequence,

(a) **Uniform access**. Each partial stripe write occurs only once, so each data element is written w times.

(b) **Random access**. Since the number of total data elements in a stripe is less than 49 when $p = 5$ and $p = 7$, we use 50 random numbers (ranging from 1 to 1000) generated by a random integer generator [25] as the frequencies of partial stripe writes in the sequence one after another. These 50

[3]P-Code has two variations, which are denoted by P-Code-1 and P-Code-2.

[4]For $p = 5$ and $p = 7$ in different codes, the number of data elements in a stripe is less than $7 * 7 = 49$.

[5]A partial stripe write across different stripes is a little different from the one in our ideal sequence, which need more I/O operations but has little effect on the evaluation results.

TABLE II
50 RANDOM INTEGER NUMBERS

221	811	706	753	34	862	353	428	99	502
969	800	32	346	889	335	361	209	609	11
18	76	136	303	175	71	427	143	870	855
706	297	50	824	324	212	404	199	11	56
822	301	430	558	954	100	884	410	604	253

random numbers are shown in Table II. For example, the first number in the table,"221" is used as the frequency of the first stripe writes $C_{0,0}C_{0,2}\cdots$ (for H-Code).

As summarized in Table I, we use $F(C_{i,j})$ and $P(C_{i,j})$ to denote the access frequency and the access probability of a partial stripe write to w continuous data elements starting from $C_{i,j}$, respectively. If N_s is the total access frequency of all stripe writes in the ideal sequence, we have,

$$P(C_{i,j}) = \frac{F(C_{i,j})}{N_s}, \quad \sum P(C_{i,j}) = 1 \tag{7}$$

For example, in uniform access, the access frequency and access probability of any stripe write are,

$$F(C_{i,j}) = 1, \quad P(C_{i,j}) = \frac{1}{N_s} \tag{8}$$

We evaluate H-Code and other codes in terms of the following metrics. The first metric is denoted by $S_w(C_{i,j})$, which is the number of I/O operations a partial stripe write to w continuous data elements starting from $C_{i,j}$. We define "**average number of I/O operations of a partial stripe write to w continuous data elements** ($S_{avg.}(w)$)" to evaluate different codes. The smaller value of $S_{avg.}(w)$ is, the lower cost of partial stripe writes and the higher performance of storage system is. $S_{avg.}(w)$ can be calculated by,

$$S_{avg.}(w) = \sum S_w(C_{i,j}) \cdot P(C_{i,j}) \tag{9}$$

For uniform access in H-Code, $N_s = (p-1)^2$. According to Equation 8, the average number of I/O operations of the ideal sequence of partial stripe writes is calculated using the following equation,

$$S_{avg.}(w) = \frac{\sum_{i=0}^{p-2}\left[\sum_{j=0}^{p-1} S_w(C_{i,j})\right]}{(p-1)^2} \quad (j \neq i+1) \tag{10}$$

As described in Section IV-B and Figure 6, it takes 10 I/O operations for any partial stripe write to two continuous data elements. According to Equation 12, we have the $S_{avg.}(w)$ value of H-Code,

$$S_{avg.}(w=2) = \frac{10(p-1)^2}{(p-1)^2} = 10$$

According to Equation 1, for a random partial stripe write, the numbers of read and write I/O are the same, so we use average number of I/O operations to evaluate different codes.

Our next metric is the "**maximum number of I/O operations of a partial stripe write to w continuous data elements** ($S_{max.}(w)$)". It is the maximum number of I/O operations of all partial stripe writes in the sequence and calculated by,

$$S_{max.}(w) = \max\left[S_w(C_{i,j})\right] \tag{11}$$

To show the I/O distribution among different disks, we use another metric, $S_w^j(C_{i,j})$, to denote the number of I/O operations in column j of a partial stripe write to w continuous data elements starting from $C_{i,j}$. We define "**average number of I/O operations in column j of a partial stripe write to w continuous data elements** ($S_{avg.}^j(w)$)" as follows,

$$S_{avg.}^j(w) = \sum S_w^j(C_{i,j}) \cdot P(C_{i,j}) \tag{12}$$

For H-Code, we have the following equation,

$$S_{avg.}^j(w) = \frac{\sum_{i=0}^{p-2}\left[\sum_{j=0}^{p-1} S_w^j(C_{i,j})\right]}{(p-1)^2} \quad (j \neq i+1) \tag{13}$$

B. Numerical Results

In this subsection, we give the numerical results of H-Code compared to other typical codes using above metrics.

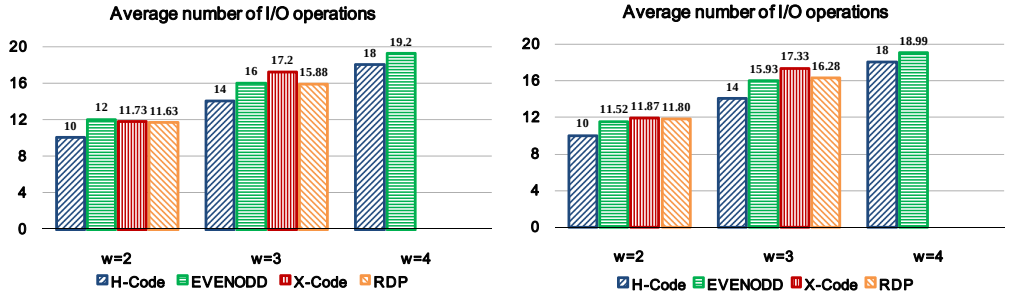
*1) **Average I/O Operations:*** First we calculate the average I/O operation counts ($S_{avg.}(w)$ values) for different codes[6] with various w and p shown in Figure 7 and Figure 8. The results show that H-Code can reduce the cost of partial stripe writes, and thus improve the performance of storage system.

We also summarize the costs in terms of I/O operations of our H-Code compared to other codes, which are shown in Table III and IV. It is obvious that H-Code has the lowest I/O cost of partial stripe writes. For uniform access, there is a decrease of I/O operations up to 14.68% and 20.45% compared to RDP and EVENODD codes, respectively. For random access, compared to RDP and EVENODD codes, H-Code reduces the cost by up to 15.54% and 22.17%, respectively.

*2) **Maximum I/O Operations:*** Next we evaluate the maximum I/O operations shown in Figure 9 and 10. It clearly shows that H-Code has the lowest maximum number of I/O operations compared to other coding methods in all cases.

The above evaluations demonstrate that H-Code outperforms other codes in terms of average and maximum I/O operations. The reasons that H-Code has the lowest partial stripe write cost are: First, H-Code has a special anti-diagonal parity (the last data element in a row and the first data element in the next row share the same anti-diagonal parity) , which can decease the partial stripe write cost when continuous data elements are crossing rows like vertical codes. Second, we keep the horizontal parity similar to EVENODD and RDP

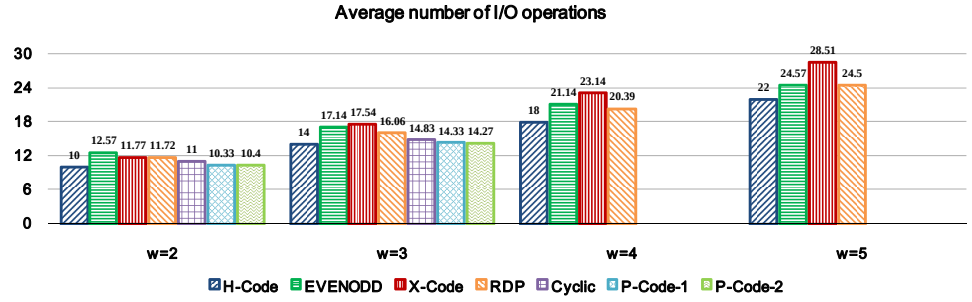
[6]In the following figures and tables, due to the constraint of $2 \leq w \leq n-3$, the average/maximum number of I/O operations in some codes are not available.



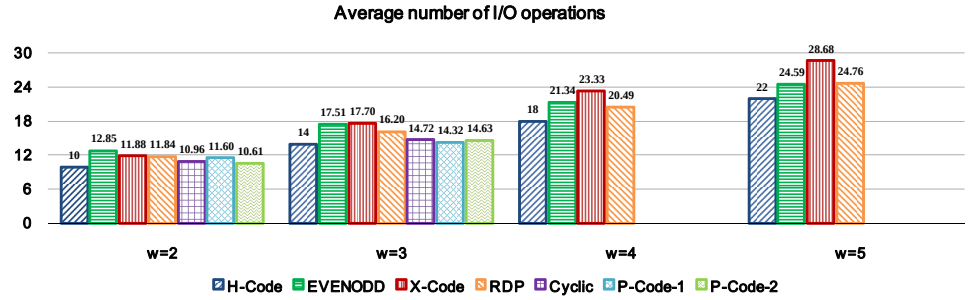
(a) uniform access

(b) random access

Fig. 7. Average number of I/O operations of a partial stripe write to w continuous data elements of different codes with different value of w when $p = 5$ (5 disks for X-Code, 6 disks for H-Code and RDP code, and 7 disks for EVENODD code).



(a) uniform access



(b) random access

Fig. 8. Average number of I/O operations of a partial stripe write to w continuous data elements of different codes with different value of w when $p = 7$ (6 disks for P-Code-1 and Cyclic code, 7 disks for X-Code and P-Code-2, 8 disks for H-Code and RDP code, and 9 disks for EVENODD code).

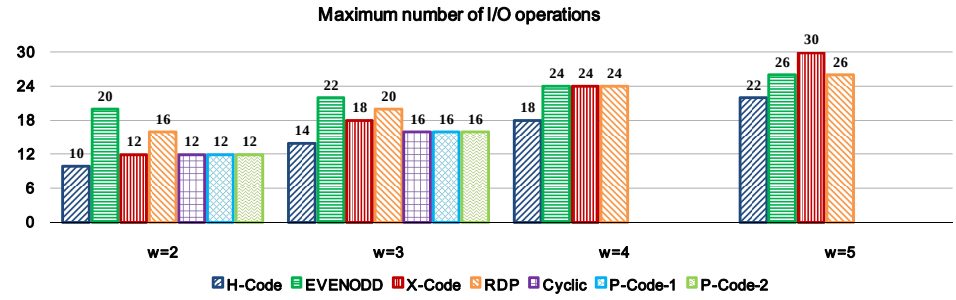


Fig. 10. Maximum number of I/O operations of a partial stripe write to w continuous data elements of different codes with different value of w when $p = 7$ (6 disks for P-Code-1 and Cyclic code, 7 disks for X-Code and P-Code-2, 8 disks for H-Code and RDP code, and 9 disks for EVENODD code).

TABLE III
IMPROVEMENT OF H-CODE OVER OTHER CODES IN TERMS OF AVERAGE PARTIAL STRIPE WRITE COST (UNIFORM ACCESS)

w & p	EVENODD code	X-Code	RDP code	Cyclic code	P-Code-1	P-Code-2
$w = 2$, $p = 5$	16.67%	14.75%	14.02%	–	–	–
$w = 3$, $p = 5$	12.50%	18.60%	11.84%	–	–	–
$w = 2$, $p = 7$	20.45%	15.04%	14.68%	9.09%	3.19%	3.85%
$w = 3$, $p = 7$	18.32%	20.18%	12.83%	5.60%	2.30%	1.89%
$w = 4$, $p = 7$	14.85%	22.21%	11.72%	–	–	–
$w = 5$, $p = 7$	10.46%	22.83%	10.20%	–	–	–

TABLE IV
IMPROVEMENT OF H-CODE OVER OTHER CODES IN TERMS OF AVERAGE PARTIAL STRIPE WRITE COST (RANDOM ACCESS)

w & p	EVENODD code	X-Code	RDP code	Cyclic code	P-Code-1	P-Code-2
$w = 2$, $p = 5$	13.19%	15.75%	15.25%	–	–	–
$w = 3$, $p = 5$	12.12%	19.22%	14.00%	–	–	–
$w = 2$, $p = 7$	22.17%	15.82%	15.54%	8.76%	13.79%	5.75%
$w = 3$, $p = 7$	20.04%	20.90%	13.58%	4.89%	2.23%	4.31%
$w = 4$, $p = 7$	15.65%	22.84%	12.15%	–	–	–
$w = 5$, $p = 7$	10.53%	23.29%	11.15%	–	–	–

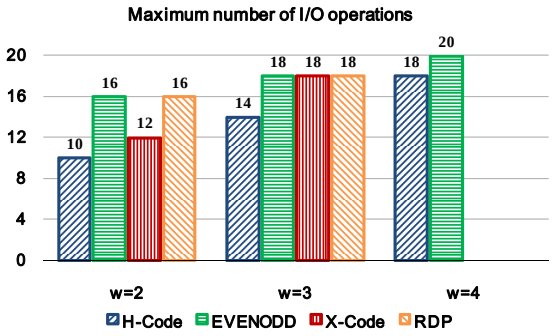


Fig. 9. Maximum number of I/O operations of a partial stripe write to w continuous data elements of different codes with different value of w when $p = 5$ (5 disks for X-Code, 6 disks for H-Code and RDP code, and 7 disks for EVENODD code).

codes, which is efficient for partial stripe writes to continuous data elements in the same row. Therefore, our H-Code takes advantages of both horizontal codes (such as EVENODD and RDP) and vertical codes (such as X-Code, Cyclic and P-Code).

*3) **Partial Stripe Write in the Same Row:*** Third, we evaluate the cost for a partial stripe write to w continuous data elements in the same row of H-Code and some other typical codes shown in Table V. Compared to EVENODD and X-Code, H-Code reduces I/O cost by up to 19.66% and 16.67%, respectively.

From Table V, we find that, for a partial stripe write within a row, H-code offers better partial stripe write performance compared to other typical codes. Due to the horizontal parity, H-code performs better than X-Code (e.g., 10 vs. 12 I/O operations when $w = 2$) in partial stripe write performance in the same row. H-Code has much lower cost than EVENODD because of different anti-diagonal construction schemes.

*4) **I/O Workload Balance:*** As we mentioned before, H-Code doesn't suffer from unbalanced I/O distribution which

TABLE V
COST OF A PARTIAL STRIPE WRITE TO w CONTINUOUS DATA ELEMENTS IN THE SAME ROW ($p = 7$, UNIFORM ACCESS)

	H-Code	EVENODD	X-Code
$w = 2$	10	12.44	12
$w = 3$	14	16.8	16
$w = 4$	18	20.5	20

is an issue in RDP code. To verify this, we calculate the $S^j_{avg.}(w)$ values for H-Code and RDP code as shown in Figure 11. It shows that for RDP code, the workload is not balance (in disk 6 and disk 7 it is very high, especially in disk 7). However, our H-Code balances the workload very well among all disks because of the dispersed anti-diagonal parity in different columns.

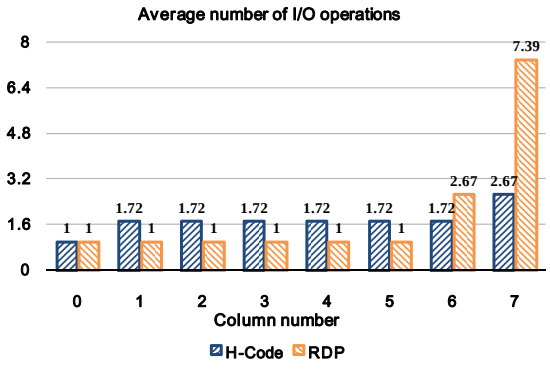


Fig. 11. Average number of I/O operations in column j of a partial stripe write to three continuous data elements of different codes in an 8-disk array ($p = 7, w = 3$).

VI. Conclusions

In this paper, we propose a Hybrid Code (H-Code), to optimize partial stripe writes for RAID-6 in addition to take advantages of both horizontal and vertical MDS codes. H-Code is a solution for an array of $(p+1)$ disks, where p is a prime number. The parities in H-Code include horizontal row parity and anti-diagonal parity, where anti-diagonal parity elements are distributed among disks in the array. Its horizontal parity ensures a partial stripe write to continuous data elements in a row share the same row parity chain to achieve optimal partial stripe write performance. Our theoretical analysis shows that H-Code is optimal in terms of storage efficiency, encoding/decoding computational complexity and single write complexity. Furthermore, we find H-Code offers optimal partial stripe write complexity to two continuous data elements and optimal partial stripe write performance among all MDS codes to the best of our knowledge. Our H-code reduces partial stripe write cost by up to 15.54% and 22.17% compared to RDP and EVENODD codes. In addition, H-Code achieves better I/O workload balance compared to RDP code.

Acknowledgments

We greatly appreciate Ozcan Ozturk and other anonymous reviewers for their constructive comments. This research is sponsored by the U.S. National Science Foundation (NSF) Grants CCF-0937799 and CCF-0937850, the National Basic Research 973 Program of China under Grant No. 2011CB302303, the National Natural Science Foundation of China under Grant No. 60933002, the National 863 Program of China under Grant No.2009AA01A402, and the Innovative Foundation of Wuhan National Laboratory for Optoelectronics. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

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*, 44(2):192–202, February 1995.

[2] M. Blaum and R. Roth. On lowest density MDS codes. *IEEE Transactions on Information Theory*, 45(1):46–59, January 1999.

[3] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. An XOR-based Erasure-Resilient coding scheme. Technical Report TR-95-048, International Computer Science Institute, August 1995.

[4] J. Bonwick. RAID-Z. http://blogs.sun.com/bonwick/entry/raidz, 2010.

[5] Y. Cassuto and J. Bruck. Cyclic lowest density MDS array codes. *IEEE Transactions on Information Theory*, 55(4):1721–1729, April 2009.

[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 *Proc. of the USENIX FAST'04*, San Francisco, CA, March 2004.

[7] E. David. Method for improving partial stripe write performance in disk array subsystems. US Patent No. 5333305, July 1994.

[8] G. Feng, R. Deng, F. Bao, and J. Shen. New efficient MDS array codes for RAID part I: Reed-Solomon-like codes for tolerating three disk failures. *IEEE Transactions on Computers*, 54(9):1071–1080, September 2005.

[9] G. Feng, R. Deng, F. Bao, and J. Shen. New efficient MDS array codes for RAID part II: Rabin-like codes for tolerating multiple ($\geq$ 4) disk failures. *IEEE Transactions on Computers*, 54(12):1473–1483, December 2005.

[10] H. Fujita and K. Sakaniwa. Modified Low-Density MDS array codes for tolerating double disk failures in disk arrays. *IEEE Transactions on Computers*, 56(4):563–566, April 2007.

[11] K. Gopinath, N. Muppalaneni, N. Kumar, and P. Risbood. A 3-tier RAID storage system with RAID1, RAID5 and compressed RAID5 for Linux. In *Proc. of the USENIX ATC'00*, San Diego, CA, June 2000.

[12] K. Greenan, X. Li, and J. Wylie. Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs. In *Proc. of the IEEE MSST'10*, Incline Village, NV, May 2010.

[13] J. Hafner. WEAVER codes: Highly fault tolerant erasure codes for storage systems. In *Proc. of the USENIX FAST'05*, San Francisco, CA, December 2005.

[14] J. Hafner. HoVer erasure codes for disk arrays. In *Proc. of the IEEE/IFIP DSN'06*, Philadelphia, PA, June 2006.

[15] C. Huang, M. Chen, and J. Li. Pyramid Codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. In *Proc. of the IEEE NCA'07*, Cambridge, MA, July 2007.

[16] C. Huang and L. Xu. STAR: An efficient coding scheme for correcting triple storage node failures. In *Proc. of the USENIX FAST'05*, San Francisco, CA, December 2005.

[17] C. Jin, H. Jiang, D. Feng, and L. Tian. P-Code: A new RAID-6 code with optimal properties. In *Proc. of the ICS'09*, Yorktown Heights, NY, June 2009.

[18] H. Jin, X. Zhou, D. Feng, and J. Zhang. Improving partial stripe write performance in RAID level 5. In *Proc. of the IEEE ICCDCS'98*, Isla de Margarita, Venezuela, March 1998.

[19] A. Kuratti. *Analytical Evaluation of the RAID 5 Disk Array*. PhD thesis, University of Arizona, 1994.

[20] D. Patterson, G. Gibson, and R. Katz. A case for Redundant Arrays of Inexpensive Disks (RAID). In *Proc. of the ACM SIGMOD'88*, Chicago, IL, June 1988.

[21] E. Pinheiro, W. Weber, and L. Barroso. Failure trends in a large disk drive population. In *Proc. of the USENIX FAST'07*, San Jose, CA, February 2007.

[22] J. Plank. A new minimum density RAID-6 code with a word size of eight. In *Proc. of the IEEE NCA'08*, Cambridge, MA, July 2008.

[23] J. Plank. The RAID-6 liberation codes. In *Proc. of the USENIX FAST'08*, San Jose, CA, February 2008.

[24] J. Plank, J. Luo, C. Schuman, L. Xu, and Z. Wilcox-O'Hearn. A performance evaluation and examination of open-source erasure coding libraries for storage. In *Proc. of the USENIX FAST'09*, San Francisco, CA, February 2009.

[25] RANDOM.ORG. Random Integer Generator. http://www.random.org/integers/, 2010.

[26] I. Reed and G.Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, pages 300–304, 1960.

[27] B. Schroeder and G. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *Proc. of the USENIX FAST'07*, San Jose, CA, February 2007.

[28] L. Tian, D. Feng, H. Jiang, L. Zeng, J. Chen, Z. Wang, and Z. Song. PRO: A popularity-based multi-threaded reconstructionoptimization for RAID-structured storage systems. In *Proc. of the USENIX FAST'07*, San Jose, CA, February 2007.

[29] S. Wan, Q. Cao, C. Xie, B. Eckart, and X. He. Code-M: A Non-MDS erasure code scheme to support fast recovery from up to two-disk failures in storage systems. In *Proc. of the IEEE/IFIP DSN'10*, Chicago, IL, June 2010.

[30] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao. WorkOut: I/O workload outsourcing for boosting RAID reconstruction performance. In *Proc. of the USENIX FAST'09*, San Francisco, CA, February 2009.

[31] J. Wylie and R. Swaminathan. Determining fault tolerance of XOR-based erasure codes efficiently. In *Proc. of the IEEE/IFIP DSN'07*, Edinburgh, UK, June 2007.

[32] L. Xiang, Y. Xu, J. Lui, and Q. Chang. Optimal recovery of single disk failure in RDP code storage systems. In *Proc. of the ACM SIGMETRICS'10*, New York, NY, June 2010.

[33] L. Xu and J. Bruck. X-Code: MDS array codes with optimal encoding. *IEEE Transactions on Information Theory*, 45(1):272–276, January 1999.