HDP Code: A Horizontal-Diagonal Parity Code to Optimize I/O Load Balancing in RAID-6

Chentao Wu, Xubin He, Guanying Wu
Department of Electrical & Computer Engineering
Virginia Commonwealth University
Richmond, VA 23284, USA
\{wuc4,xhe2,wug\}@vcu.edu

Shenggang Wan, Xiaohua Liu, Qiang Cao, Changsheng Xie
Wuhan National Laboratory for Optoelectronics
Huazhong University of Science & Technology
Wuhan, China 430074
\{wanshenggang, lxhhust350\}@gmail.com,
\{caoqiang, cs_xie\}@hust.edu.cn

Abstract—With higher reliability requirements in clusters and data centers, RAID-6 has gained popularity due to its capability to tolerate concurrent failures of any two disks, which has been shown to be of increasing importance in large scale storage systems. Among various implementations of erasure codes in RAID-6, a typical set of codes known as Maximum Distance Separable (MDS) codes aim to offer data protection against disk failures with optimal storage efficiency. However, because of the limitation of horizontal parity or diagonal/anti-diagonal parities used in MDS codes, storage systems based on RAID-6 suffers from unbalanced I/O and thus low performance and reliability.

To address this issue, in this paper, we propose a new parity called Horizontal-Diagonal Parity (HDP), which takes advantages of both horizontal and diagonal/anti-diagonal parities. The corresponding MDS code, called HDP code, distributes parity elements uniformly in each disk to balance the I/O workloads. HDP also achieves high reliability via speeding up the recovery under single or double disk failure. Our analysis shows that HDP provides better balanced I/O and higher reliability compared to other popular MDS codes.

Index Terms—RAID-6; MDS Code; Load Balancing; Horizontal Parity; Diagonal/Anti-diagonal Parity; Performance Evaluation; Reliability

I. INTRODUCTION

Redundant Arrays of Inexpensive (or Independent) Disks (RAID) \cite{25, 7} has become one of most popular choices to supply high reliability and high performance storage services with acceptable spatial and monetary cost. In recent years, with the increasing possibility of multiple disk failures in large storage systems \cite{33, 20}, RAID-6 has received too much attention due to the fact that it can tolerate concurrent failures of any two disks.

There are many implementations of RAID-6 based on various erasure coding technologies, of which Maximum Distance Separable (MDS) codes are popular. MDS codes can offer protection against disk failures with given amount of redundancy \cite{6}. According to the structure and distribution of different parities, MDS codes can be categorized into horizontal codes \cite{30, 5, 3, 8, 4, 28, 27} and vertical codes \cite{6, 41, 42, 23}. A typical horizontal code based RAID-6 storage system is composed of \( m + 2 \) disk drives. The first \( m \) disk drives are used to store original data, and the last two are used as parity disk drives. Horizontal codes have a common disadvantage that \( m \) elements must be read to recover any one other element. Vertical codes have been proposed that disperse the parity across all disk drives, including X-Code \cite{42}. Cyclic code \cite{6}, and P-Code \cite{23}. All MDS codes have a common disadvantage that \( m \) elements must be read to recover any one other element. This limitation reduces the reconstruction performance during single disk or double disk failures.

However, most MDS codes based RAID-6 systems suffer from unbalanced I/O, especially for write-intensive applications. Typical horizontal codes have dedicated parities, which need to be updated for any write operations, and thus cause higher workload on parity disks. Unbalanced I/O also occurs on some vertical codes like P-Code \cite{23} consisting of a prime number of disks due to unevenly distributed parities in a stripe. Even if many dynamic load balancing approaches \cite{12, 32, 22, 1, 15, 2} are given for disk arrays, it is still difficult to adjust the high workload in parity disks and handle the override on data disks. Although some vertical codes such as X-Code \cite{42} can balance the I/O but they have high cost to recover single disk failure as shown in the next section.

The unbalanced I/O hurts the overall storage system performance and the original single/double disk recovery method has some limitation to improve the reliability. To address this issue, we propose a new parity called Horizontal-Diagonal Parity (HDP), which takes advantage of both horizontal parity and diagonal/anti-diagonal parity to achieve well balanced I/O. The corresponding code using HDP parities is called Horizontal-Diagonal Parity Code (HDP Code) and distributes all parities evenly to achieve balanced I/O. Depending on the number of disks in a disk array, HDP Code is a solution for \( p - 1 \) disks, where \( p \) is a prime number.

We make the following contributions in this work:

- We propose a novel and efficient XOR-based RAID-6 code (HDP Code) to offer not only the property provided by typical MDS codes such as optimal storage efficiency,

A stripe means a complete (connected) set of data and parity elements that are dependently related by parity computation relations \cite{17}. Typically, a stripe is a matrix as in our paper.
but also best load balancing and high reliability due to horizontal-diagonal parity.

- We conduct a series of quantitative analysis on various codes, and show that HDP Code achieves better load balancing and higher reliability compared to other existing coding methods.

The rest of this paper continues as follows: Section II discusses the motivation of this paper and details the problems of existing RAID-6 coding methods. HDP Code is described in detail in Section III. Load balancing and reliability analysis are given in Section IV and V. Section VI briefly overviews the related work. Finally we conclude the paper in Section VII.

II. PROBLEMS OF EXISTING MDS CODES AND MOTIVATIONS OF OUR WORK

To improve the efficiency, performance, and reliability of the RAID-6 storage systems, different MDS coding approaches are proposed while they suffer from unbalanced I/O and high cost to recover single disk. In this section we discuss the problems of existing MDS codes and the motivations of our work. To facilitate the discussion, we summarize the symbols used in this paper in Table I.

A. Load Balancing Problem in Existing MDS Codes

RAID-5 keeps load balancing well based on the parity declustering approach. Unfortunately, due to dedicated distribution of parities, load balancing is a big problem in all horizontal codes in RAID-6. For example, Figure 1 shows the load balancing problem in RDP and the horizontal parity layout shown in Figure 1(a). Assuming \( C_{i,j} \) delegates the element in \( i \)th row and \( j \)th column, in a period there are six reads and six writes to a stripe as shown in Figure 1(c). For a single read on a data element, there is just one I/O operation. However, for a single write on a data element, there are at least six I/O operations\(^2\) one read and one write on data elements, two read and two write on the corresponding parity elements. Then we can calculate the corresponding I/O distribution and find that it is an extremely unbalanced I/O as shown in Figure 1(d). The number of I/O operations in column 6 and 7 are five and nine times higher than column 0, respectively. It may lead to a sharp decrease of reliability and performance of a storage system.

Some vertical coding methods have unevenly distributed parities, which also suffer from unbalanced I/O. For example, Figure 2 shows the load balancing problem in P-Code. From the layout of P-Code as shown in Figure 2(a), column 6 goes without any parity element compared to the other columns, which leads to unbalanced I/O as shown in Figure 2(b), though uniform access happens. We can see that column 6 has very low workload while column 0’s workload is very high (six times of column 6). This can decrease the performance and reliability of the storage system.

Many dynamic load balancing approaches are proposed which can adjust the unbalanced I/O in data disks according to various access patterns of different workloads. While in RAID-6 storage system, it is hard to transfer the high workload in parity disks to any other disks, which breaks the construction of erasure code thus damages the reliability of the whole storage system. Except for parity disks, if there is an overwrite upon the existing data in a data disk, because of the lower cost to update the original disks compared to new writes on other disks and related parity disks, it is difficult to adapt the workload to other disks even if the workload of this data disk is very high. All these mean that unbalanced write to parity disks or override write to data disks cannot be adapted by dynamic load balancing methods in RAID-6. Further more, typical dynamic load balancing approaches in disk array consume some resources to monitor the status of various storage devices and find the disks with high or low workload, then do some dynamic adjustment according to the monitoring results. This brings additional overhead to the disk array.

Industrial products based on RAID-6 like EMC CLARiiON uses static parity placement to provide load balancing in each eight stripes, but it also has additional overhead to handle the data placement and suffers from unbalanced I/O in each stripe.

B. Reduce I/O Cost of Single Disk Failure

In 2010, Xiang et al. proposes a hybrid recovery approach named RDOR, which uses both horizontal and diagonal

\(^2\)For some horizontal codes like RDP, some diagonal parities are calculated by the corresponding data elements and horizontal parity element. So, it could be more than six operations for a single write.

### Table I: Symbols

<table>
<thead>
<tr>
<th>Parameters &amp; Symbols</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>number of disks in a disk array</td>
</tr>
<tr>
<td>( p )</td>
<td>a prime number</td>
</tr>
<tr>
<td>( i, j )</td>
<td>row ID</td>
</tr>
<tr>
<td>( j )</td>
<td>column ID or disk ID</td>
</tr>
<tr>
<td>( C_{i,j} )</td>
<td>element at the ( i )th row and ( j )th column</td>
</tr>
<tr>
<td>( f_1, f_2 )</td>
<td>two random failed columns with IDs ( f_1 ) and ( f_2 )</td>
</tr>
<tr>
<td>( \sum )</td>
<td>XOR operations between/among elements (e.g., ( \sum_{j=0}^{5} C_{i,j} = C_{i,0} \oplus C_{i,1} \oplus \cdots \oplus C_{i,5} ))</td>
</tr>
<tr>
<td>( \langle \cdot \rangle )</td>
<td>modular arithmetic (e.g., ( \langle i \rangle_p = i \mod p ))</td>
</tr>
<tr>
<td>( O_j(C_{i,j}) )</td>
<td>number of I/O operations to column ( j ), which is caused by a request to data element(s) with beginning element ( C_{i,j} )</td>
</tr>
<tr>
<td>( O(j) )</td>
<td>total number of I/O operations of requests to column ( j ) in a stripe</td>
</tr>
<tr>
<td>( O_{\text{max}}, O_{\text{min}} )</td>
<td>maximum/minimum number of I/O operations among all columns</td>
</tr>
<tr>
<td>( L )</td>
<td>metric of load balancing in various codes</td>
</tr>
<tr>
<td>( R_t )</td>
<td>average time to recover a data/parity element</td>
</tr>
<tr>
<td>( R_p )</td>
<td>average number of elements can be recovered in a time interval ( R_t )</td>
</tr>
</tbody>
</table>
parities to recover single disk failure. It can minimize I/O cost and has well balanced I/O on disks except for the failed one. For example, as shown in Figure 3, data elements A, B and C are recovered by their diagonal parity while D, E and F are recovered by their horizontal parity. With this method, some elements (e.g., C_{3,0}) can be shared to recover another parity, thus the number of read operations can be reduced. Actually, up to 22.60% of disk access time and 12.60% of recovery time are decreased in the simulation experiments [40].

This approach has less effect on vertical codes like X-Code. As shown in Figure 4, data elements A, B, C and F are recovered by their anti-diagonal parity while D, E and G are recovered by their diagonal parity. Although X-Code recovers more elements compared to RDP in these examples, X-Code share fewer elements than RDP thus has less effects on reducing the I/O cost when single disk fails.

RDOR is an efficient way to recover single disk failure, but it cannot reduce I/O cost when parity disk fails. For example, as shown in Figure 3 when column 7 fails, nearly all data should be read and the total I/O cost is relatively high.

C. Summary on Different Parities in Various Coding Methods

To find out the root of unbalanced I/O in various MDS coding approaches, we analyze the features of various parities in these code. We classify these approaches into four categories (including the horizontal-diagonal parity introduced in this paper): horizontal parity, diagonal/anti-diagonal parity, vertical parity, and horizontal-diagonal parity (HDP).

1) Horizontal Parity (HP): Horizontal parity is the most important feature for all horizontal codes, such as EVENODD [3] and RDP [8], etc. The horizontal parity layout is shown in Figure 1(a) and can be calculated by,

$$C_{i,n-2} = \sum_{j=0}^{n-3} C_{i,j}$$  \hspace{1cm} (1)

In some papers, horizontal parity is referred to as “row parity”.

---

Fig. 1. Load balancing problem in RDP code for an 8-disk array (The I/O operations in columns 0, 1, 2, 3, 4 and 5 are very low, while in columns 6 and 7 are very high. These high workload in parity disks may lead to a sharp decrease of reliability and performance of a storage system).
From Equation 1, we can see that typical HP can be calculated by a number of XOR computations. For a partial stripe write on continuous data elements, it could have lower cost than the other parities due to the fact that these elements can share the same HP (e.g., partial write cost on continuous data elements \( C_{2,1} \) and \( C_{2,2} \) is shown in Figure 1(b)). Through the layout of HP, horizontal codes can be optimized to reduce the recovery time when data disk fails.

However, there is an obvious disadvantage for HP, which is workload imbalance as discussed in Section II-A.

2) Diagonal/Anti-diagonal Parity (DP/ADP): Diagonal or anti-diagonal parity typically appears in horizontal codes or vertical codes, such as in RDP and X-Code (Figure 1(c) and 4(a)), which achieve a well-balanced I/O. The anti-diagonal parity of X-Code can be calculated by,

\[
C_{n-1,i} = \sum_{k=0}^{n-3} C_{k,(i-k-2)_n}
\]

From Equation 2, some modular computation to calculate the corresponding data elements (e.g., \((i - k - 2)_n\)) are introduced compared to Equation 1.

For DP/ADP, it has a little effect on reducing the probability of single disk failure as discussed in Section II-B.
3) Vertical Parity (VP): Vertical parity normally appears in vertical codes, such as in B-Code [41] and P-Code [23]. Figure 2(a) shows the layout of vertical parity. The construction of vertical parity is more complex: first, some data elements are selected as a set, upon which the modular calculations are done. Thus, the computation cost for a vertical parity is higher than other parities. Except for some exceptions like P-Code shown in Figure 2(a) most vertical codes achieve a well-balanced I/O. Due to the complex layout, reducing the I/O cost of any single disk is infeasible for vertical codes like P-Code.

Table I summarizes the major features of different parities, all of which suffer from unbalanced I/O and the poor efficiency on reducing the I/O cost of single disk failure. To address these issues, we propose a new parity named HDP to take advantage of both horizontal and diagonal/anti-diagonal parities to achieve low computation cost, high reliability, and balanced I/O. The layout of HDP is shown in Figure 5(a). In next section we will discuss how HDP parity is used to build HDP code.

III. HDP Code

To overcome the shortcomings of existing MDS codes, in this section we propose the HDP Code, which takes the advantage of both horizontal and diagonal/anti-diagonal parities in MDS codes. HDP is a solution for $p - 1$ disks, where $p$ is a prime number.

A. Data/Parity Layout and Encoding of HDP Code

HDP Code is composed of a $p - 1$-row-$p - 1$-column square matrix with a total number of $(p - 1)^2$ elements. There are three types of elements in the square matrix: data elements, horizontal-diagonal parity elements, and anti-diagonal parity elements. $C_{i,j}$ $(0 \leq i \leq p - 2, 0 \leq j \leq p - 2)$ denotes the element at the $i$th row and the $j$th column. Two diagonals of this square matrix are used as horizontal-diagonal parity and anti-diagonal parity, respectively.

Horizontal-diagonal parity and anti-diagonal parity elements of HDP Code are constructed according to the following encoding equations:

Horizontal parity:

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

Anti-diagonal parity:

$$C_{i,p-2-i} = \sum_{j=0}^{p-2} C_{(2i+j+2)p,j} \quad (j \neq p - 2 - i \text{ and } j \neq (p - 3 - 2i)p)$$

Figure 5 shows an example of HDP Code for an 6-disk array ($p = 7$). It is a 6-row-6-column square matrix. Two diagonals of this matrix are used for the horizontal-diagonal parity elements ($C_{0,0}, C_{1,1}, C_{2,2}, \text{etc.}$) and the anti-diagonal parity elements ($C_{0,5}, C_{1,4}, C_{2,3}, \text{etc.}$).

The encoding of the horizontal-diagonal parity of HDP Code is shown in Figure 5(a). We use different icon shapes to denote different sets of horizontal-diagonal elements and the corresponding data elements. Based on Equation 3, all horizontal elements are calculated. For example, the horizontal element $C_{0,0}$ can be calculated by $C_{0,1} + C_{2,2} + C_{0,3} + C_{0,4} + C_{0,5}$.

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

B. Construction Process

According to the above data/parity layout and encoding scheme, the construction process of HDP Code is straightforward:

- Label all data elements.
- Calculate both horizontal and anti-diagonal parity elements according to Equations 3 and 4.

C. Proof of Correctness

To prove the correctness of HDP Code, we take the case of one stripe for example here. 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

$$T_k = \left\langle p - 2 + \frac{k+1+\frac{1}{2}(k-1)}{2}, f_2 - f_1 \right\rangle$$

with $0 < f_2 - f_1 < p - 1$, all two-integer tuples $(0, f_1), (0, f_2), \cdots, (p - 2, f_1), (p - 2, f_2)$ occur exactly once in the sequence.

The similar proof of this lemma can be found in many papers on RAID-6 codes [3] [8] [42] [23].
We may start the reconstruction process from data element \( C_{0,0} \) in Lemma 1. Similarly, without any missing parity elements, all data elements can be reconstructed and the reconstruction can be performed under concurrent failures from any two columns.

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

**Proof:** The two failed columns are denoted as \( f_1 \) and \( f_2 \), where \( 0 < f_1 < f_2 < p \).

In the construction of HDP Code, any two horizontal-diagonal parity elements cannot be placed in the same row, as well for any two anti-diagonal parity elements. For any two concurrently failed columns \( f_1 \) and \( f_2 \), based on the layout of HDP Code, two data elements \( C_{p-f_2+f_1-1} \) and \( C_{p-f_1-1} \) can be reconstructed since the corresponding anti-diagonal parity element does not appear in the other failed column.

For the failed columns \( f_1 \) and \( f_2 \), if a data element \( C_{i,f_2} \) on column \( f_2 \) can be reconstructed, we can reconstruct the missing data element \( C_{i+f_1-f_2} \) on the same anti-diagonal parity chain if its corresponding anti-diagonal parity elements exist. Similarly, a data element \( C_{i,f_1} \) in column \( f_1 \) can be reconstructed, we can reconstruct the missing data element \( C_{i+f_2-f_1} \) on the same anti-diagonal parity chain if its anti-diagonal parity element parity element exist. Let us consider the construction process from data element \( C_{f_2-1} \) on the \( f_2 \)-th column to the corresponding endpoint (element \( C_{f_1-1} \) on the \( f_1 \)-th 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, without any missing parity elements, we may start the reconstruction process from data element \( C_{f_2-1} \) on the \( f_1 \)-th column to the corresponding endpoint (element \( C_{f_2-1} \) on the \( f_2 \)-th column).

In conclusion, HDP Code can be reconstructed under concurrent failures from any two columns.

**D. Reconstruction Process**

We first consider how to recover a missing data element since any missing parity element can be recovered based on Equations 3 and 4. If the horizontal-diagonal parity element and the related \( p - 2 \) data elements exist, we can recover the missing data element (assuming it’s \( C_{i,f_1} \) in column \( f_1 \) and \( 0 \leq f_1 \leq p-2 \)) using the following equation,

\[
C_{i,f_1} = \sum_{j=0}^{p-2} C_{i,j} \quad (j \neq f_1) \quad (5)
\]

If there exists an anti-diagonal parity element and its \( p-3 \) data elements, to recover the data element \( C_{i,f_1} \), first we should recover the corresponding anti-diagonal parity element. Assume it is in row \( r \) and this anti-diagonal parity element can be represented by \( C_{r,p-2-r} \) based on Equation 4, then we have:

\[
i = \langle 2r + f_1 + 2 \rangle_p \quad (6)
\]

So \( r \) can be calculated by (\( k \) is an arbitrary positive integer):

\[
r = \begin{cases} 
(i - f_1 + p - 2)/2 & (i - f_1 = \pm 2k + 1) \\
(i - f_1 + 2p - 2)/2 & (i - f_1 = -2k) \\
(i - f_1 - 2)/2 & (i - f_1 = 2k)
\end{cases} \quad (7)
\]

According to Equation 4, the missing data element can be recovered,

\[
C_{i,f_1} = C_{r,p-2-r} \oplus \sum_{j=0}^{p-2} C_{i+j+j+2,p} \quad (j \neq f_1 \text{ and } j \neq (p - 3 - 2i)_p) \quad (8)
\]
Algorithm 1: Reconstruction Algorithm of HDP 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.

\[
\begin{array}{l}
\text{switch } 0 \leq f_1 < f_2 \leq p - 2 \text{ do} \\
\text{Step 2-A: Compute two starting points } (C_{p-1-f_1+f_1}, f_1) \text{ and } (C_{f_2-f_2+1}, f_2) \text{ of the recovery chains based on Equations 6 to 8.} \\
\text{Step 2-B: Recover the lost data elements in the two recovery chains.} \\
\text{Two cases start synchronously:} \\
\text{case starting point is } C_{p-1-f_1+f_1}, f_1 \text{ repeat.} \\
(1) \text{ Compute the next lost data element (in column } f_2 \text{) in the recovery chain based on Equation 5;} \\
(2) \text{ Until at the endpoint of the recovery chain: } C_{f_1,f_1}. \\
\text{case starting point is } C_{f_2-f_2+1}, f_2 \text{ repeat.} \\
(1) \text{ Compute the next lost data element (in column } f_1 \text{) in the recovery chain based on Equation 5;} \\
(2) \text{ Until at the endpoint of the recovery chain: } C_{f_1,f_1}. \\
\end{array}
\]

Fig. 7. Reconstruction Algorithm of HDP Code.

Based on Equations 5 to 8 we can easily recover the elements upon single disk failure. If two disks failed (for example, column \( f_1 \) and column \( f_2 \), \( 0 \leq f_1 < f_2 \leq p - 2 \)), based on Theorem 1, we have our reconstruction algorithm of HDP Code shown in Figure 7.

In this section, we evaluate HDP to demonstrate its effectiveness on load balancing.

A. Evaluation Methodology

We compare HDP Code with following popular codes in typical scenarios (when \( p = 5 \) and \( p = 7 \)):
- Codes for \( p - 1 \) disks: HDP Code;
- Codes for \( p \) disks: P-Code [23];
- Codes for \( p + 1 \) disks: RDP code [8];
- Codes for \( p + 2 \) disks: EVENODD code [3].

For each coding method, we analyze a trace as shown in Figure 8. We can see that most of the write requests are 4KB and 8KB in Microsoft Exchange application. Typically a stripe size is 256KB [9] and a data block size is 4KB, so single write and partial stripe write to two continuous data elements are dominant and have significant impacts on the performance of disk array.

Fig. 8. Write request distribution in Microsoft Exchange trace (most write requests are 4KB and 8KB).

Based on the analysis of exchange trace, we select various types of read/write requests to evaluate various codes as follows:
- Read (R): read only;
- Single Write (SW): only single write request without any read or single write to continuous data elements;
- Continuous Write (CW): only write request to two continuous data elements without any read or single write;
- Mixed Read/Write (MIX): mixed above three types. For example, “RSW” means mixed read and single write requests, “50R50SW” means 50% read requests and 50% single write requests.

To show the status of I/O distribution among different codes, we envision an ideal sequence to read/write requests in a stripe as follows,

For each data element [4] it is treated as the beginning read/written element at least once. If there is no data element

\[\text{Percentage (\%)}\]

<table>
<thead>
<tr>
<th>Request Size (Bytes)</th>
<th>Data</th>
<th>HDP</th>
<th>ADP</th>
<th>Lost Data and Parity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>17.78</td>
<td>32.57</td>
<td>31.32</td>
<td>8.46</td>
</tr>
<tr>
<td>2K</td>
<td>0.87</td>
<td>4.73</td>
<td>3.49</td>
<td>0.78</td>
</tr>
<tr>
<td>4K</td>
<td>21.37</td>
<td>32.21</td>
<td>31.32</td>
<td>8.03</td>
</tr>
<tr>
<td>8K</td>
<td>14</td>
<td>3.14</td>
<td>3.49</td>
<td>0.78</td>
</tr>
<tr>
<td>16K</td>
<td>7</td>
<td>4.73</td>
<td>3.49</td>
<td>0.78</td>
</tr>
<tr>
<td>32K</td>
<td>3.49</td>
<td>8.46</td>
<td>0.78</td>
<td>0.78</td>
</tr>
<tr>
<td>64K</td>
<td>0.78</td>
<td>8.46</td>
<td>0.78</td>
<td>0.78</td>
</tr>
<tr>
<td>128K</td>
<td>0.78</td>
<td>8.46</td>
<td>0.78</td>
<td>0.78</td>
</tr>
</tbody>
</table>

Fig. 6. Reconstruction by two recovery chains (there are double failures in columns 2 and 3): First we identify the two starting points of recovery chain: data elements \( A \) and \( G \). Second we reconstruct data elements according to the corresponding recovery chains until they reach the endpoints (data elements \( F \) and \( L \)). The orders to recover data elements are: one is \( A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \), the other is \( G \rightarrow H \rightarrow I \rightarrow J \rightarrow K \rightarrow L \).

E. Property of HDP Code

From the proof of HDP Code’s correctness, HDP Code is essentially an MDS code, which has optimal storage efficiency [42] [6] [23].
at the end of a stripe, the data element at the beginning of the stripe will be written.

Based on the description of ideal sequence, for HDP Code shown in Figure 5 (which has \( (p - 3) \times (p - 1) \) total data elements in a stripe), the ideal sequence of CW requests to \( (p-3)*(p-1) \) data elements is: \( C_{0,1}C_{0,2}, C_{0,2}C_{0,3}, C_{0,3}C_{0,4}, \) and so on, \( \cdots \), the last partial stripe write in this sequence is \( C_{p-2,p-3}C_{0,0} \).

We also select two access patterns combined with different types of read/write requests:

- **Uniform access.** Each request occurs only once, so for read requests, each data element is read once; for write requests, each data element is written once; for continuous write requests, each data element is written twice.

- **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 \(^6\) as the frequencies of partial stripe writes in the sequence one after another. These 50 random numbers are shown in Table III. For example, in HDP Code, the first number in the table, “221” is used as the frequency of a CW request to elements \( C_{0,1}C_{0,2} \).

We define \( O_j(C_{i,j}) \) as the number of I/O operations in column \( j \), which is caused by a request to data element(s) with beginning element \( C_{i,j} \). And we use \( O(j) \) to delegate the total number of I/O operations of requests to column \( j \) in a stripe, which can be calculated by,

\[
O(j) = \sum O_j(C_{i,j})
\]

We use \( O_{max} \) and \( O_{min} \) to delegate maximum/minimum number of I/O operations among all columns, obviously,

\[
O_{max} = \max O(j), \quad O_{min} = \min O(j)
\]

We evaluate HDP Code and other codes in terms of the following metrics. We define “metric of load balancing (\( L \))” to evaluate the ratio between the columns with the highest I/O \( O_{max} \) and the lowest I/O \( O_{min} \). The smaller value of \( L \) is, the better load balancing is. \( L \) can be calculated by,

\[
L = \frac{O_{max}}{O_{min}}
\]

For continuous write requests, each data element is written once; for read requests, each data element is read once; for horizontal codes, especially for uniform write requests, we also use \( L \) to delegate maximum/minimum number of I/O operations appears in their data disk. The smaller value of \( L \) is, the better load balancing is.

We define \( p \) as the number of total data elements in a stripe, which can be calculated by,

\[
0 = \frac{O_{max}}{O_{min}} = 1
\]

**B. Numerical Results**

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

First, we calculate the metric of load balancing (\( L \) values) for different codes \(^6\) with various \( p \) shown in Figure 9 and Figure 10. The results show that HDP Code has the lowest \( L \) value and thus the best balanced I/O compared to the other codes.

Second, to discover the trend of unbalanced I/O in horizontal codes, especially for uniform write requests, we also calculate the \( L \) value in RDP, EVENODD and HDP.

For HDP Code, which is well balanced as calculated in Equation 12 for any uniform request. For RDP and EVENODD codes, based on their layout, the maximum number of I/O operations appears in their diagonal parity disk and the minimum number of I/O operations appears in their data disk. By this method, we can get different \( L \) according to various values of \( p \). For example, for uniform SW requests in RDP code,

\[
L = \frac{O_{max}}{O_{min}} = \frac{2(p-1)^2 + 2(p-2)^2}{2(p-1)} = 2p-4+\frac{1}{p-1}
\]

We summarize the results in Table IV and give the trend of \( L \) in horizontal codes with different \( p \) values in Figure 11 and 12. With the increasing number of disks (\( p \) becomes larger), RDP and EVENODD suffer extremely unbalanced I/O while HDP code gets well balanced I/O in the write-intensive environment.

**V. RELIABILITY ANALYSIS**

By using special horizontal-diagonal and anti-diagonal parities, HDP Code also provides high reliability in terms of fast recovery on single disk and parallel recovery on double disks.

**A. Fast Recovery on Single Disk**

As discussed in Section II-B, some MDS codes like RDP can be optimized to reduce the recovery time when single disk fails. This approach also can be used for HDP Code to get higher reliability. For example, as shown in Figure 13.

\[^6\]In the following figures, because there is no I/O in parity disks, the minimum I/O \( O_{min} = 0 \) thus \( L = \infty \).
when column 0 fails, not all elements need to be read for recovery. Because there are some elements like $C_{0,4}$ can be shared to recover two failed elements in different parities. By this approach, when $p = 7$, HDP Code reduces up to 37% read operations and 26% total I/O per stripe to recover single disk, which can decrease the recovery time thus increase the reliability of the disk array.

We summarize the reduced read I/O and total I/O in different codes in Table V. We can see that HDP achieves highest gain on reduced I/O compared to RDP and X-Code. Based on the layout of HDP Code, we also keep well balanced read I/O when one disk fails. For example, as shown in Figure 14, the remaining disks all 4 read operations, which is well balanced.

B. Parallel Recovery on Double Disks

According to Figures 6 and 7, HDP Code can be recovered by two recovery chains all the time. It means that HDP Code can be reconstructed in parallel when any two disks fail, which...
**A. MDS Codes in RAID 6**

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, including Reed-Solomon code [30], Cauchy Reed-Solomon code [5], EVENODD code [3], RDP code [8], Blaum-Roth code [4], Liberation code [28], Liberation code [27], Cyclic code [6], B-Code [41], X-Code [42], and P-Code [23]. These codes are Maximum Distance Separable (MDS) codes, which offer protection against disk failures with given amount of redundancy [6]. RSL-Code [10], RL-Code [11] and STAR [21] are MDS codes tolerating concurrent failures of three disks. In addition to MDS codes, some non-MDS codes, such as WEAVER [15], HoVer [17], MEL code [39], Pyramid code [20], Flat XOR-Code [13] and Code-M [36], also offer resistance to concurrent failures of any two disks, but they have higher storage overhead. And there are some approaches to improve the efficiency of different codes [33][38][40][24]. In this paper, we focus on MDS codes in RAID-6, which can be further divided into two categories: horizontal codes and vertical codes.

1) **Horizontal MDS Codes:** Reed-Solomon code [30] is based on addition and multiply operations over certain finite fields $GF(2^p)$. 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 [5] addresses this problem and improves Reed-Solomon code by changing the multiply operations over $GF(2^p)$ into additional XOR operations.
Unlike the above generic erasure coding technologies, EVENODD [3] 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 [8] 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. [4] 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 \((i \times j + j - 1)\) 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, most horizontal codes like RDP have unbalanced I/O distribution as shown in Figure 1, especially high workload in parity disks.

2) **Vertical MDS Codes:** X-Code, Cyclic code, and P-Code are vertical codes due to the fact that their parities are not in dedicated redundant disk drives, but dispersed over all disks. This layout achieves better encoding/decoding computational complexity, and improved single write complexity.

X-Code [22] uses diagonal parity and anti-diagonal parity and the number of columns (or disks) in X-Code must be a prime number.

Cyclic code [6] 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 \(2 \times (p - 1)\).

P-Code [25] 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 P-Code may suffer from unbalanced I/O due to uneven distribution of parities as shown in Figure 2. Although other vertical codes like X-Code keep load balancing well, they also have high I/O cost to recover single disk failure compared to horizontal codes (as shown in Figure 4).

We summarize our HDP code and other popular codes in Table VII.

### B. Load Balancing in Disk Arrays

Load balancing is an important issue in parallel and distributed systems area [43], and there are many approaches for achieving the load balancing in disk arrays. At the beginning of 1990s, Holland et al. [19] investigated that parity declustering was an effective way to keep load balancing in RAID-5, and Weikum et al. [37] found that dynamic file allocation can help achieving load balancing. Ganger et al. [12] compare the disk striping with the conventional data allocation in disk arrays, and find that disk striping is a better method and can provide significantly improved load balance with reduced complexity for various applications. Scheuermann et al. [31] propose a data partitioning method to optimize disk striping and achieves load balance by proper file allocation and dynamic redistributions of the data access. After 2000, some patents by the industry solve the load balancing problem in disk array [22] [1] [18] [2]. Recently, with the development of virtualization in computer systems, a few approaches focus on dynamic load balancing in virtual storage devices and virtual disk array [34] [14] [15].

### VII. Conclusions

In this paper, we propose the Horizontal-Diagonal Parity Code (HDP Code) to optimize the I/O load balancing for RAID-6 by taking advantage of both horizontal and diagonal/anti-diagonal parities in MDS codes. HDP Code is a solution for an array of \(p - 1\) disks, where \(p\) is a prime number. The parities in HDP Code include horizontal-diagonal parity and anti-diagonal parity, where all parity elements are distributed among disks in the array to achieve well balanced I/O. Our mathematic analysis shows that HDP Code achieves the best load balancing and high reliability compared to other MDS codes.

### Acknowledgments

We thank anonymous reviewers for their insightful comments. This research is sponsored by the U.S. National Science Foundation (NSF) Grants CCF-1102605, CCF-1102624, and CNS-1102629, 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


### TABLE VII

**SUMMARY OF DIFFERENT CODING METHODS IN RAID-6**

<table>
<thead>
<tr>
<th>Code Methods</th>
<th>HP</th>
<th>DP/ADP</th>
<th>VP</th>
<th>HDP</th>
<th>Computation cost</th>
<th>Workload balance</th>
<th>Reduced I/O cost to recover single disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENODD code</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>high</td>
<td>unbalance</td>
<td>high</td>
</tr>
<tr>
<td>B-Code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>high</td>
<td>balance</td>
<td>none</td>
</tr>
<tr>
<td>X-Code</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>high</td>
<td>balance</td>
<td>low</td>
</tr>
<tr>
<td>RDP code</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>low</td>
<td>unbalance</td>
<td>high</td>
</tr>
<tr>
<td>HoVer code</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>low</td>
<td>unbalance</td>
<td>high</td>
</tr>
<tr>
<td>F-Code</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>low</td>
<td>unbalance</td>
<td>none</td>
</tr>
<tr>
<td>Cyclic code</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>medium</td>
<td>unbalance</td>
<td>none</td>
</tr>
<tr>
<td>HDP code</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>low</td>
<td>balance</td>
<td>high</td>
</tr>
</tbody>
</table>


