HV Code: An All-around MDS Code to Improve Efficiency and Reliability of RAID-6 Systems

Zhirong Shen, Jiwu Shu[†]
Department of Computer Science and Technology, Tsinghua University
[†]*Corresponding author: shujw@tsinghua.edu.cn*
zhirong.shen2601@gmail.com

***Abstract*—The increasing expansion of data scale leads to the widespread deployment of storage systems with larger capacity and further induces the climbing probability of data loss or damage. The *Maximum Distance Separable* (MDS) code in RAID-6, which tolerates the concurrent failures of any two disks with minimal storage requirement, is one of the best candidates to enhance the data reliability. However, most of the existing works in this literature are more inclined to be specialized and cannot provide a satisfied performance under an all-round evaluation.**

Aiming at this problem, we propose an all-round MDS code named *Horizontal-Vertical Code* (HV Code) by taking advantage of horizontal parity and vertical parity. HV Code achieves the perfect I/O balancing and optimizes the operation of partial stripe writes to continuous data elements, while preserving the optimal encode/decode/update efficiency. Moreover, it owns a shorter parity chain which grants it a more efficient recovery for one disk failure. HV Code also behaves well for the degraded read operation and accelerates the process to reconstruct two disabled disks by executing four recovery chains in parallel. The performance evaluation demonstrates that HV Code well balances the I/O distribution and eliminates up to 27.6% and 32.4% I/O requests for partial stripe writes operation when compared with RDP Code and HDP Code. Moreover, compared to RDP Code, HDP Code, X-Code and H-Code, HV Code reduces up to 5.4%∼39.8% I/O requests per element for the single disk reconstruction, decreases 6.6%∼28.3% I/O requests for degraded read operations, and achieves the same efficiency of X-Code for double disk recovery by shortening 47.4%∼59.7% recovery time compared with other three codes.

***Keywords*-RAID-6; Storage System; Load Balancing; Partial Stripe Writes; Disk Recovery; Degraded Read**

I. Introduction

With the rapid development of cloud storage, the size of created data to be kept is amazingly expanding, resulting in the strong demand of the storage systems with larger capacity (e.g., GFS [6] and Windows Azure [26]). Increasing the storage volume is usually achieved by equipping with more disks, but it also comes with the rising probability of multiple disk failures [1] [2] with the system scales up. To provide a reliable and economical storage service with high performance, *Redundant Arrays of Inexpensive (or Independent) Disks* (RAID) receives tremendous attention and is widely adopted nowadays. Among the various branches of RAID, the *Maximum Distance Separable* (MDS) code of RAID-6 offering the tolerance for concurrent failures of any two disks with optimal storage efficiency, becomes one of the most popular solutions.

In RAID-6 system, the original data will be partitioned into many pieces with constant size (denoted as "data elements") and the redundant information with the same size (denoted as "parity elements") is calculated over a subgroup of data elements. Once a disk failure happens (e.g., hardware malfunction or software error), the surviving data elements and parity elements will be selectively retrieved to reconstruct the elements on the dispirited disk. In the meantime of disk failure, the storage system may always receive the read operation to the data elements resided on the corrupted disk (this operation is referred as degraded read operation). Therefore, to timely response of user's I/O requests and improve data reliability, it is extremely critical to efficiently process data recovery and degraded read.

Meanwhile, in general, "healthy" RAID-6 storage systems also have to cope with the frequent write accesses to the hosted data (especially for partial stripe writes to continuous data elements). A write operation to a data element will trigger the update to the associated parity elements. Therefore, in the circumstance with intensive write requests, storage systems may easily suffer from the unbalanced load and are likely to be exhausted with the absorption of a considerable amount of extra I/O requests. Thus, to accelerate the write operation and improve the system reliability, a RAID-6 storage system that can well balance the load and be efficient to cope with the partial stripe writes is an imperative need.

Based on the above concerns, we extract the following five metrics to roughly evaluate the performance of RAID-6 storage systems. 1) The capability to balance the I/O distribution; 2) The performance of partial stripe writes; 3) The efficiency to reconstruct the failed disk (disks); 4) The overhead of degraded read operation; 5) The encode/decode/update complexity. Deep investigations (e.g., [7] [3] [10] [4] [8] [5]) have been endeavored to pursue the optimization on one of these metrics, *yet all of them are attending to one goal and losing another*. On the aspect of balancing the I/O requests, both X-Code [7] and HDP Code [3] can evenly disperse the load to the whole disk array, but the former has a poor performance on partial stripe writes

978-1-4799-2233-8/14 $31.00 © 2014 IEEE
DOI 10.1109/DSN.2014.57

while the latter suffers high update computation complexity. On the metric of partial stripe writes and degraded read, H-Code [10] achieves excellent performance for the writes to two continuous data, but it has little effect on both balancing the load and still undergoes unsatisfied repair efficiency to recover the corrupted disks. In the case of disk reconstruction, though P-Code [8] requires less elements, its complex data manipulation and unsatisfied performance on partial stripe writes and degraded read make it less competitive. More details about the weaknesses among existing MDS codes are discussed in the next section.

In this paper, we propose a novel XOR-based MDS RAID-6 code called *Horizontal-Vertical Code* (HV Code) by taking advantage of horizontal parity and vertical parity. By evenly dispersing parity elements across the disk arrays, HV Code can well balance the I/O load. It accelerates the repair process for disk (or disks) reconstruction as well by reducing the number of data elements involved in the generation of parity elements, so that less elements should be retrieved for every lost element. To optimize the efficiency of the writes to two continuous data elements, HV Code utilizes the advantage of horizontal parity which only renews the horizontal parity element once for the updated data elements in the same row, and designs a dedicate construction for the vertical parity to ensure the last data element in the i-th row will share a same vertical parity element with the first data element in the $(i+1)$-th row. In addition, HV Code also provides competitive performance on degraded read by utilizing the horizontal parity and still retains the optimal encode/decode/update computation complexity. Our contributions can be concluded as follows:

1) We propose an all-around MDS code named HV Code, which well balances the load to the disks, offers an optimized partial stripe write experience, reduces the average recovery I/O to repair every failed element, provides fast degraded read performance, and retains the best encode/decode/update efficiency.
2) To demonstrate the efficiency of HV Code, we conduct a series of intensive experiments on the metrics of load balancing, partial stripe writes, degraded read operation, and reconstruction for the single/double disk failures. The results show HV Code achieves the same load balancing rate as X-code and HDP Code, significantly decreases about 27.6%∼32.4% write requests caused by the partial stripe writes, and eliminates 6.6%∼28.3% read requests for degraded read operation. With the aspect of recovery I/O, HV Code reduces up to 5.4%∼39.8% I/O requests to repair a lost element during the single disk reconstruction compared with its competitors (i.e., RDP Code, HDP Code, X-Code and H-Code). It achieves nearly the same time efficiency of X-Code in double disk recovery by decreasing 47.4%∼59.7% recovery time compared with other three typical codes.

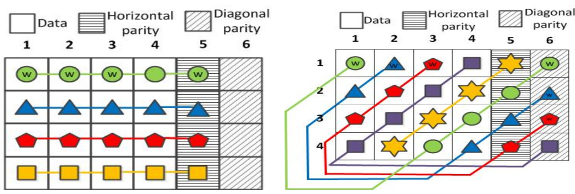
The rest of this paper is organized as follows. We first introduce the background knowledge of MDS codes and the motivation of this paper in Section II and then present the detailed design of HV Code in Section III. The property analysis of HV Code will be given in Section IV and a series of intensive evaluations are conducted to evaluate the performance of HV Code and other representative codes in Section V. Finally, we conclude our work in Section VI.

II. Background and Motivation

A. Terms and Notations

To give a better understanding of the research background of RAID-6 codes, we first summarize the terms and notations that will be frequently referred throughout this paper.

- **Data Element and Parity Element.** Element is the basic operated unit in RAID-6 systems and can be treated as an unit of a disk, such as a byte or a sector. The *data element* contains the original data information, while the *parity element* keeps the redundant information. In Figure 1(a), $E_{1,1}$ is data element and $E_{1,5}$ is parity element.
- **Stripe.** A maximal set of data elements and parity elements that have dependent relationship connected by an erasure code. Figure 1 shows the layout of a stripe in RDP Code.
- **Disk Array.** The storage system constructed over multiple disk drives, which supports file striping to improve the I/O throughput.
- **Data Disk and Parity Disk.** As the name implies, the data disk is used to place the data elements only and the parity disk is served as the storage of parity elements. For example, in Figure 1, the disk $D_1 \sim D_4$ are data disks while D_5 and D_6 are parity disks.
- **Horizontal Parity.** It is a method to compute the parity element and is also referred as "row parity". The horizontal parity element is calculated by performing the XOR operations among the data elements in the same row. For instance, in Figure 1(a), the horizontal parity element $E_{1,5} := \sum_{i=1}^{4} E_{1,i} := E_{1,1} \oplus \cdots \oplus E_{1,4}$.
- **Diagonal Parity and Anti-Diagonal Parity.** The diagonal (resp. anti-diagonal) parity connects the elements following the diagonal (resp. anti-diagonal) line. In RDP and EVENODD codes, the horizontal parity element will participate in the calculation of diagonal parity element. For example, the diagonal parity element $E_{1,6} := E_{1,1} \oplus E_{4,3} \oplus E_{3,4} \oplus E_{2,5}$ as shown in Figure1(b).
- **Vertical Parity.** It is usually adopted by vertical codes, such as P-Code [8] and B-Code [12]. In the vertical parity calculation, the candidate data elements should be picked out first and then performed the XOR operations.



(a) The Horizontal Parity (b) The Diagonal Parity

Figure 1. The Layout of RDP Code with $p+1$ Disks ($p = 5$). $\{E_{1,1}, \cdots, E_{1,5}\}$ is a horizontal parity chain and $\{E_{1,1}, E_{4,3}, E_{3,4}, E_{2,5}, E_{1,6}\}$ is a diagonal parity chain. Their length is 5.

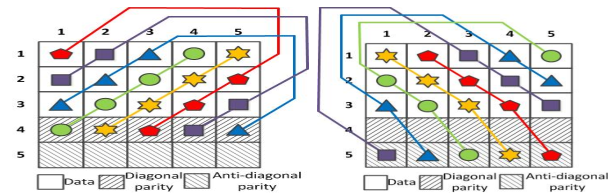
- **Parity Chain and its length.** A parity chain is composed of a group of data elements and the generated parity element. For example, $E_{1,1}$ involves in two parity chains in Figure 1, i.e., $\{E_{1,1}, E_{1,2}, \cdots, E_{1,5}\}$ for horizontal parity and $\{E_{1,1}, E_{4,3}, E_{3,4}, E_{2,5}, E_{1,6}\}$ for diagonal parity. The length of a parity chain is denoted by the number of the included elements.
- **Recovery Chain.** It is constituted by a subgroup of failed elements that have dependence in double disk reconstruction. The elements in a recovery chain will be repaired in an order. For example, there are four recovery chains in Figure 5, where $E_{2,3}$, $E_{1,1}$, $E_{1,3}$, and $E_{2,1}$ belong to the same recovery chain.

B. The MDS Codes of RAID-6 Storage Systems

The research of RAID-6 implementation has been a focus in recent years. According to the storage efficiency, current erasure codes to realize RAID-6 function can be divided into two categories: *Maximum Distance Separable* (MDS) codes and non-MDS codes. MDS codes reach optimal storage efficiency while non-MDS codes sacrifice storage efficiency to run after the improvement on other recovery metrics, for example, the reduction of recovery I/O. The representative MDS codes for RAID-6 realization include Reed-Solomon Code [19], Cauchy Reed-Solomon Code [20], EVENODD Code [5], RDP Code [4], B-Code [12], X-Code [7], Liberation Code [9], Liber8tion Code [13], P-Code [8], HDP Code [3], and H-Code [10]. The typical non-MDS codes are Pyramid Code [14], WEAVER Code [18], Code-M [16], HoVer Code [17], Local Reconstruction Codes [23] and its application [24], and Flat XOR-Code [15]. In this paper, we mainly consider MDS codes in RAID-6, which can be classified into horizontal codes and vertical codes according to the placement of parity elements.

The Horizontal Codes of RAID-6 Systems: Horizontal codes are well known as the first studies of RAID-6 systems. They are usually constructed over $m+2$ disks and demanded to reserve 2 dedicated disks to place parity elements.

As the ancestor of horizontal codes, Reed-Solomon Code is constructed over Galois field $GF(2^w)$ by employing Vandermonde matrix. Its operations (i.e., multiplication and division) are usually implemented in Galois field and this



(a) The Diagonal Parity (b) The Anti-diagonal Parity

Figure 2. The Layout of X-Code with p Disks ($p = 5$)

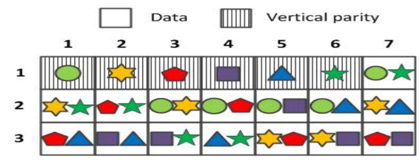


Figure 3. The Layout of P-Code with p Disks ($p = 7$). For example, data element $E_{2,1}$ joins the generation of parity P_2 (i.e., $E_{1,2}$) and parity P_6 (i.e., $E_{1,6}$), since $(2+6) \bmod 7 = 1$.

high computation complexity seriously limits its realization in practice. To mitigate this overhead, Cauchy Reed-Solomon Code introduces the binary bit matrix to convert the complex Galois field arithmetic operations into single XOR operations.

EVENODD Code and RDP Code are the typical parity array codes. By performing the XOR operations instead of finite field arithmetic, they outperform Reed-Solomon Code on the metrics of realization and efficiency. Both of them utilize the horizontal parity and diagonal parity to realize their constructions and RDP makes some differences when building the diagonal parity to achieve a better performance.

Horizontal codes own an advantage that it can be built on any number of disks, but they usually cannot approach optimal update complexity.

The Vertical Codes of RAID-6 Systems: Rather than separating the storage of data elements and parity elements, vertical codes store them together in the disk.

X-Code [7] (as shown in Fig. 2) is construed over p disks (p is a prime number) by using both diagonal parity and anti-diagonal parity. HDP-Code [3] is proposed to balance the load in a stripe by employing horizontal-diagonal parity, in which the diagonal parity element joins the calculation of horizontal parity element. H-Code [10] optimizes the partial stripe writes to continuous data elements. It gathers the horizontal parity elements on a dedicated disk and spreads the $(p-1)$ anti-diagonal parity elements over other p disks.

P-Code [8] (as shown in Fig. 3) is organized by using vertical parity. Its parity calculation follows the simple rule of $i+j=k$, where P_i and P_j are two involved parities of a data element and k is the id of the disk where the data element resides on. Nevertheless, determining the involved parities of a specified data element is a bit complex in P-Code. For example, for a data elements in disk #1, it may join either the generations of P_2 and P_6, or the computations of P_3 and P_8. To determine the parity tuples of $E_{3,1}$, one has to know the parity tuples of $E_{2,1}$, to ensure these two data elements are assigned with the generation of different parity elements. This manipulation is so troublesome that one has to sacrifice additional storage capacity to keep a mapping table for a fast parity location, otherwise a considerable computation overhead (the complexity is $O(p^2)$) will be wasted for every lookup operation. This tedious lookup operation will also lead to the complexity of $O(p^5)$ when addressing single disk reconstruction.

C. The Remained Problems of Existing MDS Codes

Though the continuous efforts are made to greatly promote the diversity and maturity of RAID-6 storage systems, most of the existing works cannot simultaneously address the following problems. These problems will potentially threaten the system reliability and degrade the system performance.

Load Balancing: Given the unbalanced I/O to a disk array will extend the operation time and even hurt the system reliability by causing uneven burden to disks, the study of balancing I/O to disks has been considered for a period of time [11] [3].

The traditional method adopts "stripe rotation" (i.e., rotationally choose the disk to serve as the parity disks among different stripes) to uniformly distribute I/O request across all the stripes. This method only takes effect when the workload is uniform among the stripes, which actually does not accord with all the I/O distributions in the real application. In the scenario that different stripes have different access frequencies, even the "stripe rotation" is applied, the stripe hosting hotter (resp. colder) data will receive more (resp. less) access requests, still causing unbalanced I/O distribution. Therefore, to well balance the load, a better method is to evenly disseminate the parity elements among the stripe.

For RDP Code [4], EVENODD Code [5], Liberation Code [9], and H-Code [10], which require dedicated disk to place the parity elements, will easily cause non-uniform I/O distribution.

Partial Stripe Writes: The partial stripe writes to continuous data elements is a frequent operation, such as backup and virtual machine migration. As a data element is usually associated with the generation of 2 parity elements in RAID-6 codes, an update to a data element will also renew at least 2 related parity elements. This property also awards horizontal parity an advantage that the update to the data elements in a row only needs to update the shared horizontal parity once.

For X-Code [7], in which any two continuous data elements do not share a common parity element, therefore the partial stripe writes will induce more extra write requests to the parity elements compared to the codes utilizing horizontal parity.

For RDP Code [4] and EVENODD Code [5], the partial stripe writes to continuous data elements will put a heavy update burden to the disk hosting diagonal parities. For example, when $E_{1,2}$, $E_{1,3}$ and $E_{1,4}$ are updated in Figure 1, then disk #5 will only need to update $E_{1,5}$ while disk #6 has to update $E_{2,6}$, $E_{3,6}$ and $E_{4,6}$, respectively. This unbalanced I/O distribution will easily delay the write operation and even threaten the system reliability by making some disks tired out.

For HDP Code [3], its high update complexity will induce considerable I/O requests. Moreover, it does not make any optimization in the case of continuous writes across rows.

Recovery Cost for Disk Failure: The recovery of RAID-6 systems can be classified into the single disk failure recovery and the reconstruction of double disk failure.

For the single disk recovery, a general way firstly proposed by Xiang et al [22] is repair the invalid elements by mixing two kinds of parity chains subjecting to the maximum overlapped elements to be retrieved, so as to achieve the minimum recovery I/O. For example, suppose the disk #1 is disabled in Figure1, then recover $E_{1,1}$ and $E_{2,1}$ by using horizontal and diagonal parity chain respectively can make $E_{1,2}$ overlapped and save one element's retrieval. The rebuild cost by adopting this method is greatly reduced but still relates to the length of parity chain. Due to the long parity chain in the existing codes, there is still room for reconstruction efficiency improvement.

Different from the selective retrieval in single disk recovery, double disk recovery demands to fetch all the elements in the survived disks. Though different codes have various layouts, the time to read all the remained elements into the main memory is the same if the parallel read is applied. Moreover, by utilizing the parallel technology, the failed elements locating at different recovery chains can be simultaneously reconstructed. For example, in Fig. 5, the element $E_{6,3}$ and $E_{5,3}$ that are not in the same recovery chain can be rebuilt at the same time. Therefore, the parallelism of recovery chains is critical in double disk reconstructions. Among the existing MDS array codes, RDP Code [4] can only serially execute this process and all of HDP Code [3], H-Code [10] and P-Code [8] achieve low parallelism of this repair by building only two recovery chains.

Degraded Read Efficiency: Degraded read operation is seriously considered when the system devotes to providing timely responses for user's read requests, even there happens to be a corrupted disk. In this case, additional elements are usually taken to recover the corrupted element to ensure the smooth proceeding of this read operation. Suppose the disk #1 fails in Figure 1, and the elements $E_{1,1}$, $E_{1,2}$, and $E_{1,3}$ are requested at that time, then extra elements $E_{1,4}$ and $E_{1,5}$ will be also attached to re-calculate the failed element $E_{1,1}$ by the horizontal parity chain. This example also reveals the horizontal parity owns an advantage to the degraded read efficiency, because some of the requested elements (e.g., $E_{1,2}$ and $E_{1,3}$) may involve in the re-calculation of the corrupted element (e.g., $E_{1,1}$) with great probability.

For X-Code [7] and P-Code [8], based on diagonal/anti-

Table I
THE FREQUENTLY USED SYMBOLS

Symbols	Description
p	the prime number
$\langle\rangle$, $\langle i\rangle_p$	modular arithmetic, i mod p
$E_{i,j}$	the element at the i-th row and j-th column
$\sum\{E_{i,j}\}$	the sum of XOR operation among the elements $\{E_{i,j}\}$
L	the length of continuous data elements to write
$\langle\frac{i}{j}\rangle_p$	if $k := \langle\frac{i}{j}\rangle_p$, then $\langle k\cdot j\rangle_p = \langle i\rangle_p$

diagonal parity and vertical parity respectively, behave a bit frustratingly on degraded read when compared to the codes constructed on horizontal parity.

Another influence factor to the degraded read performance is the length of parity chain. Longer parity chain probably incurs more unplanned read elements. Derived from this excuse, EVENODD Code [5], RDP Code [4], H-Code [10], and HDP Code [3] still easily introduce a considerable amount of additional I/O requests.

III. HV CODE FOR AN ARRAY OF p-1 DISKS

To simultaneously address the above remaining limitations, an all-around RAID-6 MDS array code should satisfy the following conditions: 1) be expert in balancing the load; 2) optimize the performance of partial stripe writes to continuous data; 3) be efficient to deal with single (resp. double) disk failure (resp. failures); 4) have a good performance on degraded read operation; 5) retain the optimal properties, such as encode/decode/update efficiency.

To this end, we propose an MDS code named HV Code, which makes use of horizontal parity and vertical parity and can be constructed over $(p-1)$ disks (p is a prime number). Before presenting the construction of HV Code, we first list the frequently used symbols in Table I.

A. Data/Parity Layout and Encoding of HV Code

A stripe of HV Code can be represented by a $(p-1)$-row-$(p-1)$-column matrix with a total number of $(p-1)\times(p-1)$ elements. There are three kinds of elements in the matrix: *data elements*, *horizontal parity elements*, and *vertical parity elements*. Suppose $E_{i,j}(1 \le i,j \le p-1)$ denotes the element at the *i*-th row and *j*-th column. In HV Code, the horizontal parity elements and the vertical parity elements are calculated by the following equations.

Horizontal parity element encoding:

$$E_{i,\langle 2i\rangle_p} := \sum_{j=1}^{p-1} E_{i,j} \quad (j \neq \langle 2i\rangle_p, j \neq \langle 4i\rangle_p) \qquad (1)$$

Vertical parity element encoding:

$$E_{i,\langle 4i\rangle_p} := \sum_{j=1}^{p-1} E_{k,j} \quad (j \neq \langle 8i\rangle_p, j \neq \langle 4i\rangle_p) \qquad (2)$$

k, j, i should satisfy the condition: $\langle 2k + \langle 4i\rangle_p\rangle_p = j$. This expression can also be simplified as $\langle 2k + 4i\rangle_p = j$. Then we can obtain k according to the following equations.

$$k := \langle\frac{j-4i}{2}\rangle_p := \begin{cases} \frac{1}{2}\langle j-4i\rangle_p & (\langle j-4i\rangle_p = 2t) \\ \frac{1}{2}(\langle j-4i\rangle_p + p) & (\langle j-4i\rangle_p = 2t+1) \end{cases}$$

Notice that if u satisfies the condition $\langle u\cdot j\rangle_p = \langle i\rangle_p$, then we express u as $u := \langle\frac{i}{j}\rangle_p$. Fig. 4 shows the layout of HV Code for a 6-disk array ($p = 7$). A horizontal parity element (represented in horizontal shadow) and a vertical parity element (represented in vertical shadow) are labeled in every row and every column.

Figure 4(a) illustrates the process of encoding the horizontal parity elements. By following Equation (1), the horizontal parity elements can be calculated by simply performing modular arithmetic and XOR operations on the data elements with the same shape. For example, the horizontal parity element $E_{1,2}$ (the row id $i = 1$) can be calculated by $E_{1,1} \oplus E_{1,3} \oplus E_{1,5} \oplus E_{1,6}$. The vertical parity element $E_{1,4}$ ($i = 1$) should not be involved in the encoding of $E_{1,2}$, because $E_{1,4}$ is at the $\langle 4i\rangle_p$-th column.

Figure 4(b) shows the process of encoding a vertical parity element. Every vertical parity element is calculated by the data elements with the same shape according to Equation (2). For example, to calculate the vertical parity element $E_{1,4}$ (the row id $i = 1$), we should first pick out the involved data elements $\{E_{k,j}\}$ based on Equation (2). When $j = 1$, then $j = \langle 8i\rangle_p$, which violates the requirements in Equation (2). When $j = 2$, then $k := \langle\frac{j-4i}{2}\rangle_p := \langle -1\rangle_p := 6$ and $E_{6,2}$ is positioned. By tracking this path, the following data elements (i.e., $E_{3,3}$, $E_{4,5}$, and $E_{1,6}$) are then fetched. Second, by performing XOR operations among these data elements, the vertical parity element $E_{1,4}$ will be computed as $E_{1,4} := E_{6,2} \oplus E_{3,3} \oplus E_{4,5} \oplus E_{1,6}$.

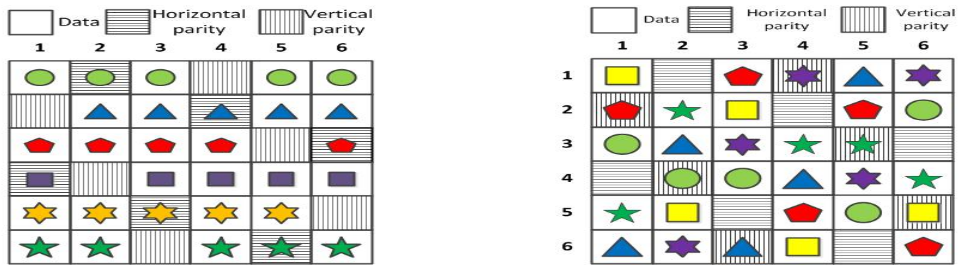
B. Construction Process

Based on the layout and encoding principle, we take the following steps to construct HV Code.

1) partition the disk according to the layout of HV Code and label the data elements in each disk;
2) encode the horizontal parity elements and the vertical parity elements respectively according to Equation (1) and Equation (2).

C. Proof of Correctness

We mainly prove HV Code is correct when applied in one stripe and the correctness over multiple stripes can be similarly deduced. We first propose the following lemma and theorem. This proof method is also adopted by [3] and [10].



(a) Horizontal Parity Encoding: a horizontal parity element can be calculated by XOR operations among the data elements in the same row. For example, $E_{1,2} := E_{1,1} \oplus E_{1,3} \oplus E_{1,5} \oplus E_{1,6}$.

(b) Vertical Parity Encoding: a vertical parity element $E_{i,j}$ can be calculated by XOR operations on selected data elements $\{E_{i',j'}\}$ satisfying the condition: $\langle 2i' + j \rangle_p = j'$. For example, $E_{1,4} := E_{6,2} \oplus E_{3,3} \oplus E_{4,5} \oplus E_{1,6}$.

Figure 4. The Layout of HV Code with $(p-1)$ Disks ($p = 7$)

Lemma 1: We can find a sequence of a two-integer tuple (T_k, T'_k) ($k := 0, \cdots, 2p-1$), where

$$T_k := \begin{cases} \langle \frac{k-1-\frac{(-1)^k+1}{2}}{2} \frac{f_1-f_2}{2} \rangle_p, & f_1 - f_2 = 2t \\ \langle \frac{k-1-\frac{(-1)^k+1}{2}}{2} \frac{f_1-f_2+p}{2} \rangle_p, & f_1 - f_2 = 2t+1 \end{cases} \quad (3)$$

$$T'_k := \frac{1+(-1)^{k+1}}{2} f_1 + \frac{1+(-1)^k}{2} f_2 \quad (4)$$

with $1 \le f_1 < f_2 \le p-1$. In this sequence, all two-integer tuples $(0, f_1), (0, f_2), \cdots, (p-1, f_1), (p-1, f_2)$ will appear only once.

Proof: To prove Lemma 1, we should only show that the two-integer tuples $\{(T_k, T'_k)\}_{k=0}^{2p-1}$ in the column f_1 and f_2 are different from each other. Due to the page limits, we only consider the case when $f_1 - f_2$ is even and this methodology can be applied to the other case.

Because the tuples at different columns will be easily differentiated by T'_k, we only need to prove the tuples at the same column are different. For the tuples (T_{k_1}, T'_{k_1}) and (T_{k_2}, T'_{k_2}) (assume $k_1 < k_2$) at column f_1, since $T'_{k_1} = T'_{k_2} = f_1$, both k_1 and k_2 are chosen from the range $(1, 3, \cdots, 2p-1)$ whose size is p. Then

$$\begin{aligned} \langle T(k_1) - T(k_2) \rangle_p &= \langle \frac{k_1 - k_2 + \frac{(-1)^{k_1} - (-1)^{k_2}}{2}}{2} \cdot \frac{f_1 - f_2}{2} \rangle_p \\ &= \langle \frac{k_1 - k_2}{2} \cdot \frac{f_1 - f_2}{2} \rangle_p \end{aligned}$$

Where $-p < \frac{k_1-k_2}{2}, \frac{f_1-f_2}{2} < 0$. Because p is a prime number and is not a factor of both $\frac{k_1-k_2}{2}$ and $\frac{f_1-f_2}{2}$, $\langle T(k_1) - T(k_2) \rangle_p \ne 0$, meaning that the p tuples resides on the column f_1 will appear only once (i.e., $(0, f_1), \cdots, (p-1, f_1)$). Based on the same principle, the p tuples (T_k, T'_k) on the column f_2 will be $(0, f_2), \cdots, (p-1, f_2)$. Therefore, the tuples $(0, f_1), (0, f_2), \cdots, (p-1, f_1), (p-1, f_2)$ will only appear once when $0 \le k \le 2p-1$. ■

Theorem 1: A stripe constructed by $(p-1)$-row-$(p-1)$-column of HV Code can tolerate the concurrent failures of any two columns.

Proof: Before giving the detailed proof, let's first review the layout of HV Code. In a stripe of HV Code, each row and column consists of $(p-3)$ data elements, a horizontal parity element and a vertical parity element. There are totally $2(p-1)$ parity chains in a stripe of HV Code, including $(p-1)$ horizontal parity chains and $(p-1)$ vertical parity chains. If a horizontal parity element and a vertical parity element are placed in the same column, the two parity chains generated by them will not intersect and both of them will go through the same $(p-2)$ columns. In addition, a parity chain, no matter horizontal parity chain or vertical parity chain, will pass through $(p-2)$ columns and intersect with any column of them only once.

Suppose the two failed columns are f_1 and f_2, where f_1 and f_2 are even and satisfy the condition $1 \le f_1 < f_2 \le p-1$ (the proof of other cases is similar). Actually, the failed elements on the two corrupted disks f_1 and f_2 can be mapped to the sequence in Lemma 1. For the non-vertical parity elements, the mapping $E_{i,f_j} \longrightarrow (i, f_j)$ establishes where ($1 \le i \le p-1, i \ne \langle \frac{f_j}{4} \rangle_p, j = 1, 2$). The vertical parity elements $E_{\langle \frac{f_j}{4} \rangle_p, f_j}$ will be mapped to the tuples $(0, f_j)$ where $j = 1, 2$. [1] Thus, we can also use the tuples to represent the elements. For example, in Fig. 5, the data element $E_{1,1}$ and the vertical parity element $E_{2,1}$ can be represented by the tuples (1,1) and (0,1).

According to the layout of HV Code, we can first find out two parity chains (including a horizontal parity chain and a vertical parity chain) that intersect at the column f_1 but miss the column f_2 and another two parity chains that bypass the column f_1 but intersect at the column f_2. Then four start elements of the recovery chains can be obtained, where $(\langle \frac{f_1}{4} \rangle_p, f_2)$ and $(\langle \frac{f_2}{4} \rangle_p, f_1)$ are the start elements recovered by horizontal parity chains, while $(\langle \frac{f_2 - \frac{f_1}{2}}{2} \rangle_p, f_2)$ and $(\langle \frac{f_1 - \frac{f_2}{2}}{2} \rangle_p, f_1)$ are the ones reconstructed by vertical parity chains. Besides the four parity chains, other parity

[1]There are also another two "virtual tuples", i.e., $(\langle \frac{f_1}{4} \rangle_p, f_1)$ and $(\langle \frac{f_2}{4} \rangle_p, f_2)$, which do not have any usage in the double disk repair.

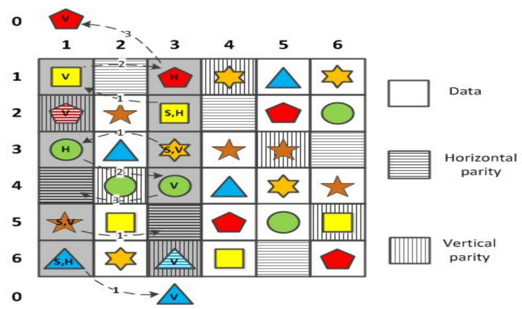


Figure 5. An Example to Recover Disk #1 and Disk #3. The elements labeling "SH" and "SV" indicate **S**tart elements recovered by **H**orizontal parity chains, and the **S**tart elements reconstructed by **V**ertical parity chains, respectively. The element labeling "H" (resp, "V") indicates it is recovered through **H**orizontal (resp. **V**ertical) parity chain. The arrow line assigned with number denotes the recovery direction and its recovery order. There are four recovery chains, such as $\{E_{5,1}, E_{5,3}\}$ and $\{E_{3,3}, E_{3,1}, E_{4,3}, E_{4,1}\}$.

chains will intersect with both f_1 and f_2.

We can summarize the recovery rule as follows: For the failed columns f_1 and f_2, on one hand, if data elements E_{i,f_1} and E_{j,f_2} are recovered from the involved vertical parities, then elements E_{i,f_2} and E_{j,f_1} (may be data elements or parity elements) could be reconstructed by following the horizontal parity chains. On the other hand, if data elements E_{i,f_1} and E_{j,f_2} are recovered from the involved horizontal parities, then in the next step we can reconstruct the elements E_{i',f_2} and E_{j',f_1} by utilizing the corresponding vertical parity chains. When E_{i',f_2} and E_{j',f_1} are data elements, they should satisfy the conditions $\langle f_1 - 2i\rangle_p = \langle f_2 - 2i'\rangle_p$ and $\langle f_2 - 2j\rangle_p = \langle f_1 - 2j'\rangle_p$, respectively. When E_{i',f_2} and E_{j',f_1} are vertical parity elements, the conditions change to $\langle 4i'\rangle_p = f_2$ and $\langle 4j'\rangle_p = f_1$.

By following this recovery rule, every recovery chain will terminate at a parity element. The horizontal parity elements are the tuples of $(\langle \frac{f_1}{2}\rangle_p, f_1)$ and $(\langle \frac{f_2}{2}\rangle_p, f_2)$, while the vertical parity elements are $(0, f_1)$ and $(0, f_2)$. During the reconstruction process, the recovered elements track the sequence of the two-integer tuple in Lemma 1. ■

D. Reconstruction

In this section, we will discuss how to perform the reconstruction when a failure happens. We mainly consider three basic kinds of failures: *the failure of an element*, *the failure of a single disk*, and *concurrent failures of double disk*. Other possible failure cases tolerated by RAID-6 systems (e.g., multiple element failures in a disk) can be covered by the combination of these three basic failures.

There are two possibilities when an element fails, resulting in different construction methods. If the failed element is a parity element, the recovery can follow either Equation (1) or Equation (2) by using the related $(p-3)$ data elements. In the case when a data element (suppose $E_{i,j}$) is broken, it can be recovered by using either horizontal parity chain or vertical parity chain. When repairing $E_{i,j}$ by horizontal parity chain, the horizontal parity element $E_{i,\langle 2i\rangle_p}$ and other related $(p-4)$ data elements should be first retrieved, then $E_{i,j}$ can be repaired as:

$$E_{i,j} := \sum_{k=1}^{p-1} E_{i,k} \quad k \neq j, k \neq \langle 4i\rangle_p \tag{5}$$

Similarly, when reconstructing $E_{i,j}$ by the vertical parity chain, the vertical parity element $E_{s,\langle 4s\rangle_p}$ can be obtained first, where $\langle 4s\rangle_p := \langle j - 2i\rangle_p$. Then

$$E_{i,j} := \sum_{l=1}^{p-1} E_{k,l} \oplus E_{s,\langle 4s\rangle_p} \tag{6}$$

where $l \neq j$, $l \neq \langle 4s\rangle_p$, $l \neq \langle 8s\rangle_p$, and $\langle 2k + 4s\rangle_p = l$.

Based on Equations (1), (2), (5), (6), if a single disk fails, the missing elements, including data elements and parity elements can be successfully reconstructed. If double disk failures occur (suppose the corrupted disks are f_1 and f_2, where $1 \leq f_1 < f_2 \leq p-1$), the recovery process can follow the procedures in Theorem 1 and the detailed steps are shown in Algorithm 1.

Algorithm 1: The Procedures to Recover The Concurrent Failures of Two Disk in HV Code.

1. locate the failed disks f_1 and f_2 $(1 \leq f_1 < f_2 \leq p-1)$;
2. recover the four start elements first: $(\langle \frac{f_2 - \frac{f_1}{2}}{2}\rangle_p, f_2)$, $(\langle \frac{f_1 - \frac{f_2}{2}}{2}\rangle_p, f_1)$, $(\langle \frac{f_1}{4}\rangle_p, f_2)$, and $(\langle \frac{f_2}{4}\rangle_p, f_1)$.
3. reconstruct other missing elements by alternately shifting between f_1 and f_2.
  Case 1: *start with the tuple* $(\langle \frac{f_2}{4}\rangle_p, f_1)$ *in disk* f_1, ***do***
    re-build the elements in disk f_2 utilizing vertical parity chain and repair the elements in disk f_1 using horizontal parity chain;
    until reach a parity element.
  Case 2: *start with the tuple* $(\langle \frac{f_1}{4}\rangle_p, f_2)$ *in disk* f_2, ***do***
    re-build the elements in disk f_1 utilizing vertical parity chain and repair the elements in disk f_2 by horizontal parity chain;
    until reach a parity element.
  Case 3: *start with the tuple* $(\langle \frac{f_1 - \frac{f_2}{2}}{2}\rangle_p, f_1)$ *in disk* f_1, ***do***
    re-build the elements in disk f_2 utilizing horizontal parity chain and repair the elements in disk f_1 using vertical parity chain;
    until reach a parity element.
  Case 4: *start with the tuple* $(\langle \frac{f_2 - \frac{f_1}{2}}{2}\rangle_p, f_2)$ *in disk* f_2, ***do***
    re-build the elements in disk f_1 utilizing horizontal parity chain and repair the elements in disk f_2 using vertical parity chain;
    until reach a parity element.

IV. Property Analysis

*1) **Optimal Storage Efficiency**:* As proved above, HV Code is an MDS code and has optimal storage efficiency [19] [4].

*2) **Optimal Construction/Reconstruction/Update Computational Complexity**:* For a code with m-row-by-n-column and x data elements, P-Code [8] has deduced the optimal XOR operations to each data element in construction is $\frac{3x - m \cdot n}{x}$ and the minimal XOR operations to each lost element in reconstruction is $\frac{3x - m \cdot n}{m \cdot n - x}$ [8].

Therefore, for a stripe of a code with $(p-1)$-row-by-$(p-1)$-column and $(p-3)(p-1)$ data elements, the optimal complexity of construction should be $\frac{2(p-4)}{(p-3)}$ XOR operations per data element and the optimal complexity of reconstruction should be $(p-4)$ XOR operations per lost element in theorem.

For HV Code, there are $2(p-1)$ parity elements needed to build in the construction. Each of them is calculated by performing $(p-4)$ XOR operations among the participated $(p-3)$ data elements. Therefore, the total XOR operations to generate the parity elements in a stripe is $2(p-4)(p-1)$, and the averaged XOR operations per data element is $\frac{2(p-4)}{(p-3)}$, being consistent with the above deduced result. Moreover, to reconstruct the two corrupted disks, every invalid element is recovered by a chain consisting of $(p-3)$ elements, among which $(p-4)$ XOR operations are performed. The complexity equals the optimal reconstruction complexity deduced above.

For update efficiency, every data element joins the computation of only two parity elements, indicating the update of any data element will only renew related two parity elements.

*3) **Achievement of Perfect Load Balancing**:* Rather than concentrating the parity elements on dedicated disks [10] [4], HV Code distributes the parity elements evenly among all the disks, which is effective to disperse the load to disks.

*4) **Fast Recovery for Disk Failure**:* The length of each parity chain in HV Code is $p-2$, which is shorter than those of many typical MDS codes [3] [10] [4] [5] [7] in RAID-6. The shortened parity chain decreases the number of needed elements when repairing a lost element. In addition, every disk holds two parity elements in HV Code, enabling the execution of four recovery chains in parallel in double disk repairs.

*5) **Optimized Partial Stripe Write Performance**:* We first dissect the performance for the writes to two continuous data elements in HV Code. The first case is when the two data elements reside in the same row, then update them will incur only one write I/O to the horizontal parity element and two separate renewals to their vertical parity elements. The second is when the renewed two data elements are in different rows, i.e., the last data element in the i-th row and the first in the $(i+1)$-th row. As described above, $E_{i,j}$ will participate in the generation of the vertical parity element residing on the $\langle j-2i\rangle_p$-th disk. This rule makes $E_{i,p-1}$ and $E_{i+1,1}$, if both of them are data elements, belong to the same vertical parity chain. Therefore, the second case only needs to update a shared vertical parity element and the two corresponding horizontal parity elements. Since the HV Code's layout defines that a column will include two parity elements, there will be at least $(p-6)$ pairs [2] of two continuous data elements which locate in different rows but share the same vertical parity elements.

[2] There are altogether $(p-2)$ pairs of continuous two data elements that are in the different rows. So the rate is $\frac{p-6}{p-2}$, which approaches to 1 when p grows.

The proof in [10] has shown any two data element updates should renew at least three parity elements in a stripe of a lowest density MDS code. Thus, HV Code achieves *near optimal performance* of partial stripe writes to two continuous data elements.

V. Performance Evaluation

In this section, we will evaluate the performance of HV Code in terms of recovery, partial stripe writes, degraded read, and load balancing. We also select RDP Code (over $p+1$ disks), HDP Code (over $p-1$ disks), H-Code (over $p+1$ disks) and X-Code (over p disks) to serve as the references.
Evaluation Environment: The performance evaluation is run on a Linux server with a X5472 processor and 12GB memory. The operating system is SUSE Linux Enterprise Server and the filesystem is EXT3. The deployed disk array consists of 16 Seagate/Savvio 10K.3 SAS disks, each of which owns 300GB capability and 100,000rmp. The machine and disk array are connected with a Fiber cable with the bandwidth of 800MB. The five MDS RAID-6 codes are realized based on Jerasure 1.2 [29] that is an open source library and widely used in this literature.
Evaluation Preparation: We first create a file, partition it into many data elements, and encode them by using every evaluated code. Like previous works [27] [28], the element size is set as 16MB. The data elements and the encoded parity elements will then be dispersed over the disk arrays by following the layout of each code (like Fig. 1 for RDP Code and Fig. 2 for X-Code). The encoded files will be used in the next tests, such as partial stripe writes efficiency and degraded read efficiency.

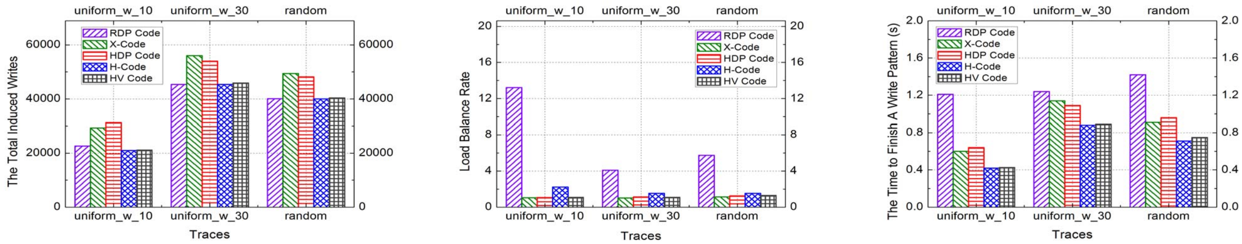
A. Partial Stripe Writes Efficiency

In this test, we mainly evaluate the efficiency when performing partial stripe writes. For a write operation with the length L, both the L continuous data elements whose size is $(16 \times L)$ MB and the associated parity elements will be totally written.

To evaluate the patterns of partial stripe writes, the following two traces are mainly considered.

- *Uniform write trace*: Every access pattern simulates the operation to write a pre-defined number of data elements starting from a uniformly chosen data element.
- *Random write trace*: Every access pattern (S, L, F) contains three random values, i.e., the start element S for the writes, the random length L, and the random write frequency F. For the pattern (S, L, F), it means the write operation that starts from the S-th data element and terminates at the $(S+L-1)$-th data element will be executed for F times.

In this test, we select two uniform write traces named ”uniform_w_10” and ”uniform_w_30”. For uniform write



(a) Total Induced Writes to A Stripe. (b) The Load Balancing Rate. (c) The Time to Complete A Write Pattern

Figure 6. The Partial Stripe Write Efficiency.($p = 13$)

trace named "uniform_w_L" (L:=10,30), we specify 1,000 write requests with the constant length L and the uniformly selected beginning. These two traces ensure the same number of data elements in a stripe is written for each code and the write frequency of each data element will be almost the same with large probability. With respect to the random write trace, we generate the patterns of (S, L, F) by employing the random integer generator [25] and the generated trace is shown in Table II. For example, (28,34,66) means the write operation will start from the 28th data element and the 34 continuous data elements will be written for 66 times.

Table II
THE RANDOM WRITE PATTERN OF (S, L, F).

(28,34,66)	(34,22,69)	(4,45,3)	(30,18,64)	(24,32,70)
(29,26,48)	(6,3,51)	(34,42,50)	(37,9,1)	(34,38,93)
(6,44,75)	(10,44,2)	(34,15,43)	(2,6,49)	(28,17,57)
(20,33,39)	(48,28,27)	(48,13,30)	(40,2,32)	(16,24,7)
(19,4,77)	(22,14,31)	(49,31,82)	(35,26,1)	(31,1,48)

Evaluation Method: For the file encoded by each code, we replay the three write traces named "uniform_w_10", "uniform_w_30", and the generated random trace respectively, just by performing every write pattern in them to the file. During the evaluation, the following three metrics are measured (as shown in Fig. 6).

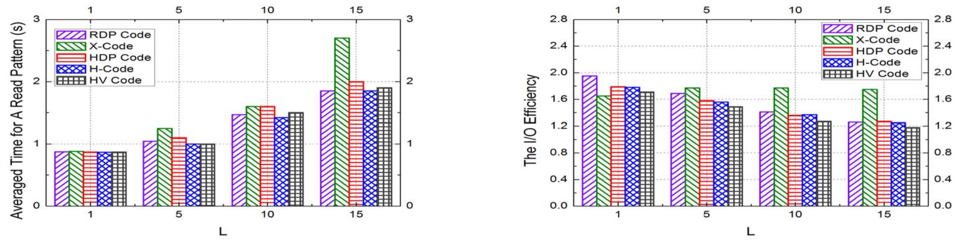
1) The total induced writes of every trace. As referred above, a write operation to a data element will trigger the associated updates to its related parity elements. For the file encoded by each code, we perform the three traces and record all the I/O requests induced by every trace (as shown in Fig. 6(a)).
2) The I/O balancing capability of each code. For the file encoded by each code, we run every trace and collect the incurred I/O requests loaded on each disk. To reflect the balancing degree of the load to a stripe, we then define the load balancing rate, which is the same with the "metric of load balancing" in [3]. Suppose the number of write requests arriving at the i-th disk is R_i and the number of disks in a stripe is N, we can calculate the **load balancing rate** λ as:

$$\lambda := \frac{Max\{R_i | 1 \leq i \leq N\}}{Min\{R_i | 1 \leq i \leq N\}} \tag{7}$$

The smaller of λ usually indicates the better behavior of balancing the load to a stripe. We finally calculate the load balancing rate of every trace for each code(as shown in Fig. 6(b)).
3) The averaged time of a write pattern. For the file encoded by each code, we measure the averaged time to execute a write pattern in every trace, i.e., from the time to start the write pattern to the time when the data elements in the pattern and corresponding parity elements are completely written (as shown in Fig. 6(c)).

Figure 6(a) indicates the total write operations of each code scale up with the increase of L for the uniform write traces "uniform_w_10" and "uniform_w_30". Though X-Code retains the optimal update complexity, its construction based on diagonal parity and anti-diagonal parity engenders more write requests to the associated parity elements. Because of the high update complexity, HDP Code triggers more write operations when compared to both of HV Code and H-Code. Since both HV Code and H-Code optimize the write operation to two continuous data elements, both of them outperform other contrastive codes when conducting the continuous writes. For uniform write trace "uniform_w_10", HV Code reduces up to 27.6% and 32.4% write I/O requests when compared to X-Code and HDP Code, respectively. When conducting random write trace, HV Code also eliminates about 18.4% and 16.2% write I/O requests when compared with X-Code and HDP Code, respectively. Even compared with H-Code, which has optimized partial stripe writes, HV Code only increases marginal extra overhead, about 0.9% under random write trace.

Figure 6(b) illustrates the load balancing rate of various codes under different access traces. Being evaluated by the three kinds of traces, RDP Code easily concentrates the writes to the parity disks and thus holds the largest load balancing rate. The load balance rates of RDP under the valuations of "uniform_w_10" and "random write trace" are 13.2 and 5.75, respectively. Though H-Code disperses the diagonal parity to the data disks, its uneven distribution of diagonal parity elements and the dedicated disk for horizontal parity storage still make it hard to achieve perfect



(a) The Averaged Time to Complete A Read Pattern (b) I/O Efficiency of A Read Pattern

Figure 7. The Degraded Read Efficiency.($p = 13$)

load balance. Its load balancing rates are 2.22 and 1.54 under the evaluations of "uniform_w_10" and "random write trace", respectively. Owing to the even distribution of parity elements, HV Code, HDP Code and X-Code approach the perfect load balancing rate (i.e., 1) under the three write traces.

The averaged time to complete a write pattern in these three traces is also recoded in Figure 6(c). In the figure, RDP Code needs the most time to complete the absorption of the write requests to the diagonal parity elements. The incompatible layout of X-Code and the expensive update complexity of HDP Code also easily extend the completion time. When performing the uniform trace "uniform_w_10", the operation time in HV Code decreases about 28.8%∼64.7% when compared to those of RDP Code, HDP Code, and X-Code. Being evaluated by the random write trace, the averaged time to complete a write pattern in HV Code is about 17.6%∼47.2% less than those of RDP Code, X-Code, and HDP Code. However, H-Code outperforms HV Code by reducing 4.1% write time on this metric. This is because the number of participating disks in H-Code is larger than that of HV Code, making H-Code better at shunting the write I/O requests. This comparison also reveals the "tradeoff" brought by the shorter parity chain, which keeps down the recovery I/O for a single disk repair but is weaker at reducing the amount of the average requests on each disk.

B. Degraded Read Comparison

Evaluation Method: Given a encoded file (encoded by RDP Code, X-Code, HDP Code, H-Code, and HV Code respectively) and the degraded read pattern with the length of L ($L := 1, 5, 10, 15$, respectively), we let the elements hosted on a disk corrupted (by either injecting faults or erasing the data on that disk) and issue 100 degraded read patterns with the length of L that start at uniformly selected points. Two metrics are concerned in this failure case, i.e., the averaged time and the I/O efficiency for a degraded read pattern.

Specifically, suppose L' denotes the number of elements returned for a degraded read pattern. When the L requested data elements happen on the surviving disks, then $L' = L$. Otherwise, if the L requested elements include the lost

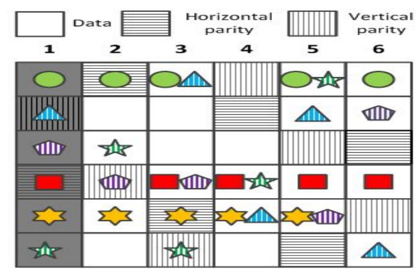


Figure 8. The Reconstruction of Disk #1 in HV Code when $p = 7$. The filled pattern elements participate in horizontal parity chain while the stripy elements involve in the vertical parity chain.

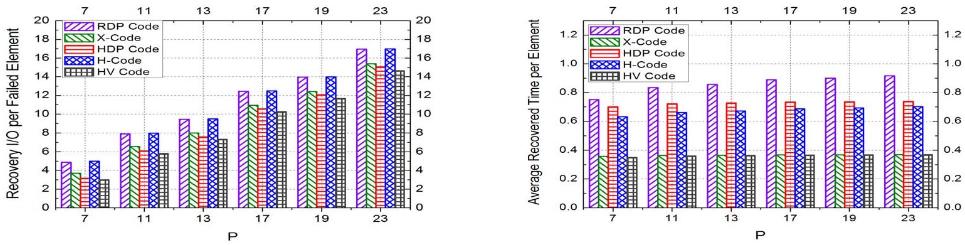
elements, then the recovery of the lost elements will be triggered by fetching the associated elements and finally $L' >= L$. The needed time for a degraded read pattern is recorded from the time of issuing the degraded read pattern to the time when the L' elements are taken from the disk array to the main memory. The I/O efficiency per degraded read pattern is evaluated by the rate $\frac{L'}{L}$. We then evaluate these two metrics under the data corruption on every disk, and calculate the expectation results in Fig. 7.

For the averaged time of a degraded read pattern (as shown in Figure 7(a)), X-Code requires the maximum time. This observation also proves the advantage of horizontal parity when handling degraded read operations. Meanwhile, this figure also reals that it usually needs more time when the number of elements to read increases. HV Code significantly outperforms X-Code and retains similar performance when compared with RDP Code, HDP Code, and H-Code.

With respect to the I/O efficiency, HV Code offers competitive performance by significantly reducing the number of read elements. When $L = 10$, HV Code eliminates about 10.0%, 28.3%, 6.6%, and 7.3% degraded read I/O requests compared with RDP Code, X-Code, HDP Code, and H-Code, respectively.

C. The Recovery I/O for Single Disk Failure

In this test, we mainly compare the required I/O to reconstruct a failed element. An example of single disk repair in HV Code is shown in Figure 8 when $p = 7$, in which at least 18 elements have to retrieve for the recovery of lost elements and thus it needs 3 elements on average to



(a) Recovery for Single Disk Failure

(b) Recovery for Double Disk Failure

Figure 9. Comparison on Disk Failure Recovery.

repair each lost element on the failed disk.

Evaluation Method: Given an encoded file, for each case of data corruption, we let the elements on that disk corrupted, evaluate the minimal averaged elements that are retrieved from the surviving disks to recover a lost element, and calculate the expectation result. Following this evaluation method, we consider the file encoded by RDP Code, X-Code, HDP Code, H-Code, and HV Code respectively when p varies, and show the results in Fig 9(a).

According to this comparison, HV Code requires the minimal number of the needed elements to repair an invalid element among the five candidates. The I/O reduction ranges from 2.7% (compare with HDP Code) to 13.8% (compare with H-Code) when $p = 23$. The saving will expand to the range from 5.4% (compare with HDP Code) to 39.8% (compare with H-Code) when $p = 7$. This superiority should own to the shorter parity chain in HV Code compared to those in other codes. We list the length of the parity chains of some representative codes in Table III, which indicates that the length of parity chain in both H-Code and RDP Code is $(p-1)$ while HV Code only keeps $(p-2)$ elements in a parity chain. Notice that the length of diagonal parity chain and the length of horizontal-diagonal parity in HDP Code are $(p-2)$ and $(p-1)$ respectively.

D. The Recovery for Double Disk Failures

As referred above, double disk failures require to fetch all the elements in the survived disks, and the time to recover the two failed disks can be simulated by the latency to accomplish the recovery in the longest recovery chain.

Evaluation Method: Suppose the average time to recover an element, either data element or parity element, is R_e and the longest length among all the recovery chains is L_c, then the time needed to finish the recovery can be evaluated by $L_c \cdot R_e$. A subsequent problem is the recovery time will be not consistent because L_c may be changeable once the failed disks vary. In this case, we consider the occurrence of any double disk failure, test every possible recovery time, and calculate the expectation.

The comparison results are shown in Figure 9 (b), which shows a big difference among the compared codes. When $p = 7$, both of X-Code and HV Code respectively reduce nearly 47.4%, 47.4%, and 43.2% of the reconstruction time when compared with RDP Code, HDP Code, and H-Code. This saving will increase to 59.7%, 50.0%, and 47.4% when $p = 23$. This huge retrenchment should owe to the placement of parity elements in X-Code and HV Code, both of which distribute two parity elements over every single disk. Every parity element will lead a recovery chain, which begins at a start point (the start point can be obtained by the chain intersecting either one of the two failed disks only) and ends at a parity element. Therefore, four recovery chains can be parallel executed without disturbing each other. RDP Code gathers the parity elements on the specified disks and the repair process should be serially processed, making it the most time-consuming code to reconstruct two corrupted disks. Both of H-Code and HDP Code though place two parity elements over every disk, the dependent relationship between the two kinds of parities in HDP Code results in the worse performance.

E. A Comparison Between HV Code and Other Codes.

Based on the above comparisons, Table III summarizes a detailed comparison between HV Code and other popular MDS array codes in RAID-6, i.e., RDP Code (over $p+1$ disks), HDP Code (over $p-1$ disks), X-Code (over p disks), and H-Code (over $p+1$ disks). The comparison indicates that HV Code achieves low cost for partial stripe writes, retains the optimal update complexity, keeps high parallelism for double disk recovery, and shortens the length of recovery chain.

VI. CONCLUSION

In this paper, we propose HV Code, which can be deployed over $p-1$ disks (p is a prime number). HV Code evenly places parities over the disks to achieve the optimization of I/O balancing. By utilizing the horizontal parity and designing a delicate construction of vertical parity, HV Code significantly decreases the I/O requests induced by the partial stripe writes to continuous data elements. Meanwhile, the shortened length of parity chain also grants HV Code a more efficient recovery for single disk failure and good performance on degraded read operation compared to other typical MDS codes in RAID-6. HV Code also

Table III
A BRIEF COMPARISON BETWEEN HV CODE AND OTHER TYPICAL MDS ARRAY CODES.

Codes	Load Balancing	Update Complexity	Partial Stripe Writes	Double Disk Reconstruction	Parity Chain Length
RDP Code [4]	unbalanced	more than 2 extra updates	**low cost**	2 recovery chains	p
HDP Code [3]	**balanced**	3 extra updates	high cost	2 recovery chains	$p-2$,$p-1$
X-Code [7]	**balanced**	**2 extra updates**	high cost	**4 recovery chains**	$p-1$
H-Code [10]	unbalanced	**2 extra updates**	**low cost**	2 recovery chains	p
HV Code	**balanced**	**2 extra updates**	**low cost**	**4 recovery chains**	$\mathbf{p-2}$

accelerates the repair of two disabled disks by deriving four independent recovery chains. The performance evaluation demonstrates the efficiency brought by HV Code.

VII. ACKNOWLEDGMENT

We would like to thank our shepherd, Arun Somani, and the anonymous reviewers for their constructive comments and suggestions. This work is supported by the National Natural Science Foundation of China (Grant No. 61232003, 60925006), the National High Technology Research and Development Program of China (Grant No. 2013AA013201), Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and Tsinghua University Initiative Scientific Research Program.

REFERENCES

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

[2] 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, 2007.

[3] C. Wu, X. He, G. Wu, S. Wan, X. Liu, Q. Cao, and C. Xie. HDP Code: A Horizontal-Diagonal Parity Code to Optimize I/O Load Balancing in RAID-6. In Proc. of DSN'11, 2011.

[4] 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, 2004.

[5] 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, 1995.

[6] S. Ghemawat, H. Gobioff, and S. Leung. The Google File System. In Proc. of ACM SOSP, 2003.

[7] L. Xu and J. Bruck. X-Code: MDS array codes with optimal encoding. IEEE Transactions on Information Theory, 1999.

[8] 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, 2009.

[9] J. Plank. The RAID-6 liberation codes. In Proc. of the USENIX FAST'08, 2008.

[10] C. Wu, S. Wan, X. He, Q. Cao, and C. Xie. H-Code: A Hybrid MDS Array Code to Optimize Partial Stripe Writes in RAID-6. In Proc. of IPDPS'11, 2011.

[11] M. Holland and G. Gibson. Parity declustering for continuous operation in redundant disk arrays. In Proc. of the ASPLOS'92, 1992.

[12] L. Xu, V. Bohossian, J. Bruck, and D. Wagner. Low-density MDS codes and factors of complete graphs. IEEE Transactions on Information Theory, 1999.

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

[14] 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, 2007.

[15] 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, 2010.

[16] 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, 2010.

[17] J. Hafner. HoVer erasure codes for disk arrays. In Proc. of the IEEE/IFIP DSN'06, 2006.

[18] J. Hafner. WEAVER codes: Highly fault tolerant erasure codes for storage systems. In Proc. of the USENIX FAST'05, 2005.

[19] I. Reed and G.Solomon. Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics, 1960.

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

[21] S. Xu, R. Li, P. Lee, Y. Zhu, L. Xiang, Y. Xu. J. Lui. Single Disk Failure Recovery for X-code-based Parallel Storage Systems. IEEE Tranasaction on Computer, 2013.

[22] L. Xiang, Y. Xu, J. C. S. Lui, and Q. Chang. Optimal recovery of single disk failure in RDP code storage systems. In Proc. of ACM SIGMETRICS'10, 2010.

[23] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure Coding in Windows Azure Storage. In Proc. of USENIX ATC'12, 2012.

[24] M. Sathiamoorthy, M. Asteris, D.S. Papailiopoulos, A.G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. XORing Elephants: Novel Erasure Codes for Big Data. In Proc. of the VLDB Endowment, 2013.

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

[26] Windows Azure: Microsoft's Cloud Platform. www.windowsazure.com/

[27] O. Khan, R. Burns, J. Plank, and W. Pierce. Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads. In Proc. of USENIX FAST'12, 2012.

[28] J. Schindler, S. Schlosser, M. Shao, A. Ailamaki, and G. Ganger. Atropos: A Disk Array Volume Manager for Orchestrated Use of Disks. In Proc. of USENIX FAST'04, 2004.

[29] J. Plank, S. Simmerman, and C. Schuman. Jerasure: A library in C/C++ facilitating erasure coding for storage applications-Version 1.2. Technical Report CS-08-627, University of Tennessee, 2008.