Short Code: An Efficient RAID-6 MDS Code for Optimizing Degraded Reads and Partial Stripe Writes

Yingxun Fu, Jiwu Shu, *Senior Member, IEEE*, Xianghong Luo, Zhirong Shen, and Qingda Hu

Abstract—As reliability requirements are increasingly important in both clusters and data centers, RAID-6, which can tolerate any two concurrent disk failures, has been widely used in modern storage systems. However, most existing RAID-6 codes cannot provide satisfied performance on both degraded reads and partial stripe writes, which are important performance metrics in storage systems. To address these problems, in this paper we propose a new RAID-6 MDS erasure code called Short Code, in order to optimize the degraded reads and partial stripe writes. In Short Code, we propose a novel short horizontal parity chain, which assures that all disks contribute to degraded reads while the continuous data elements are more likely to share the same horizontal chain for optimizing degraded reads. On the other hand, Short Code distributes all diagonal parities among disks for optimizing partial stripe writes. The proposed Short Code not only owns the optimal storage efficiency, but also keeps the optimal complexity for both encoding/decoding computations and update operations. The experiments show that Short Code achieves much higher speed on degraded reads and partial stripe writes than other popular RAID-6 codes, and provide acceptable performance on single disk failure recoveries and normal reads. Specifically, compared to RDP code, Short Code provides 6.1 to 26.3 percent higher speed on degraded reads and 36.2 to 80.3 percent higher speed on partial stripe writes with the same number of disks.

Index Terms—RAID-6, MDS erasure codes, degraded reads, partial stripe writes

1 Introduction

Modern storage systems, such as GFS [1], Windows Azure [2], and OceanStore [3], usually distribute data across hundreds of thousands of storage devices. For the concerns on both reliability and availability, the tolerance of disk failures is usually required. In recent years, Redundant Arrays of Independent Disks (RAID) have been widely used to provide high reliability and high performance. As a variant of RAID, RAID-6 that can tolerate the concurrent failures of any two disk, receives a lot of attentions [4] and [5].

There exist many RAID-6 implementations based on various erasure codes. One of the popular realizations is to use the Maximum Distance Separable (MDS) [6] codes, which are a class of erasure codes that achieve the optimal storage efficiency. According to the parity distribution, RAID-6 MDS codes can be divided into horizontal codes and vertical codes. Horizontal codes, such as RDP code [7] and EVENODD code [8], are usually constructed over $k+2$ logical disks, where the first k disks store user's data, and the last *two* disks (usually labeled P disk and Q disk, respectively) keep the parities. Vertical codes, such as X-Code [9], usually distribute their parities across all disks.

With the number of equipped disks increasing, disk failures become normal things rather than exceptions. Among different failure patterns, single disk failure is the most frequent failure and takes up to 99.75 percent of disk failure recoveries [10]. For single disk failures, two critical operations that influence the performance: *recovering the lost information from the failed disk* and *degraded reads to respond users' read requests* [11], [12], because the slow recoveries degrade the system's performance and reliability as well as the slow degraded reads degrade users' experiences when they request to access their data. In some level, degraded reads are important than single failure recoveries, because previous studies [13], [14] indicate that more than 90 percent of failures are triggered by disks temporarily unavailable (such as system upgrades and network connection snaps) and with no data lost. On the other hand, *partial stripe writes*, where writes occur on a portion of a stripe, is another critical performance metric in today's storage systems, particularly in write-intensive applications. In this paper, we assume that the contiguous data chunks are stored on different disks in storage systems to exploit parallel I/O. This assumption also exists in some previous studies [11], [15] and follows the original RAID layout [16]. However, existing RAID-6 codes cannot provide satisfied performance on both degraded reads and partial stripe writes.

For degraded reads, horizontal codes usually provide better performance than vertical codes, because the continuous data elements that need to be read in horizontal codes are usually share horizontal parity chains. On the other hand, horizontal codes also have a limitation that the diagonal parity disk (the Q disk) cannot contribute to degraded reads,

• *The authors are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China.*
E-mail: mooncape1986@126.com, shujw@tsinghua.edu.cn, {luo09, hqd13}@mails.tsinghua.edu.cn, zhirong.shen2601@gmail.com.

Manuscript received 11 Sept. 2015; revised 12 May 2016; accepted 23 May 2016. Date of publication 6 June 2016; date of current version 19 Dec. 2016.
Recommended for acceptance by C. Metra.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2016.2576461

while all disks of vertical codes can help degraded reads. Since the RAID-based storage systems usually read data in parallel, one more contributed disk effectively speeds up the degraded read performance. Therefore, if we could take advantage of both horizontal codes and vertical codes, the degraded read performance can be further improved.

For partial stripe writes, horizontal codes usually behave poorly due to the fact that the parity elements are stored in parity disks. Though most vertical codes, such as X-Code, provide good I/O balancing on partial stripe writes, they usually suffer from more write accesses than horizontal codes. In addition, some vertical codes like H-Code [15] contain horizontal parity chains and provide good performance on partial stripe writes, but they still cannot provide satisfied performance on degraded reads, because the diagonal parities contained in each row cannot contribute to degraded reads.

To address these problems, in this paper we propose a novel RAID-6 MDS erasure code named Short Code, in order to improve both degraded read performance and partial stripe write performance while provide acceptable single disk failure recovery performance and normal read (read without disk failure) performance. Different from previous MDS codes, Short Code uses short horizontal parity chains, in order to assure that all disks contribute to degraded reads and continuous data elements are more likely to share the same parity chain. Though deploying all horizontal parity elements in the same disk may cause the unbalanced I/O accesses, rotating the maps from logic disks to physical disks like RAID-5 will effectively alleviate this problem [17]. For improving the performance on partial stripe writes, Short Code distributes the diagonal parity elements across all disks. Our contributions can be summarized as follows:

- We propose a novel RAID-6 MDS erasure code termed Short Code with new parity chains and new parity distributions, to improve the performance on degraded reads and partial stripe writes.
- We conduct a series of analysis on our proposed Short Code. The results illustrate Short Code provides the optimal storage efficiency, encoding/decoding computational complexity, and update complexity.
- We implement Short Code based on Jerasure 1.2 [18], and evaluate its performance in a real storage system. The experiment results show that Short Code achieves good performance on degraded reads and partial stripe writes, and provides acceptable performance on single disk failure recoveries and normal reads. Specifically, compared to RDP code, Short Code gains 6.1 to 26.3 percent higher speed on degraded reads, 36.2 to 80.3 percent higher higher speed on partial stripe writes, 3.1 to 13.7 percent higher speed on single disk failure recoveries, and 5.2 to 27.8 percent higher speed on normal reads with the same number of disks.

The rest of this paper continues as follows: The next section states the background and the problem in existing RAID-6 codes. Section 3 presents the details and the optimal properties of Short Code. We focus on the experimental evaluations in Section 4. Finally, we conclude this paper in the last section.

2 Background and Problem Statement

Though various kinds of RAID-6 MDS codes have been proposed in literature, most of them still don't provide satisfied performance on degraded reads and partial stripe writes. In this section, we present these problems in detail and describe our motivations. Firstly, we introduce some terms and notations that are frequently used in this paper.

2.1 Terms and Notations

Element. A chunk of data or parity information, which is the fundamental unit in RAID-6 architecture. The data elements store original data information, while the parity elements keep redundant information.

Stripe. A maximal set of elements that are dependent on each other in terms of redundancy relations [19]. A stripe is usually described as a two dimensions array, where each column represents a logical disk. In this paper, we use $C_{i,j}$ to denote the element that locates on the $i\text{th}$ row and $j\text{th}$ column of the two dimensions array. Figs. 1a and 1b illustrate the layout and the parity chains of a stripe in RDP code.

Row. A maximum set of elements which belong to the same row of the above two dimensions array. In Fig. 1a, $\{C_{0,0}, C_{0,1}, \ldots, C_{0,7}\}$ compose a specific row.

Continuous Data Elements. A series of data elements that can be handled in a read/write operation. In order to take the maximum advantage of parallel I/O, we follows the assumption in [15] that the contiguous data chunks are stored on different disks. This assumption follows the definition of RAID [16] and be usually used in recent works [11]. Based on this assumption, in Fig. 1a $\{C_{0,4}$, $C_{0,5}$, $C_{1,0}$, and $C_{1,1}\}$ are continuous data elements.

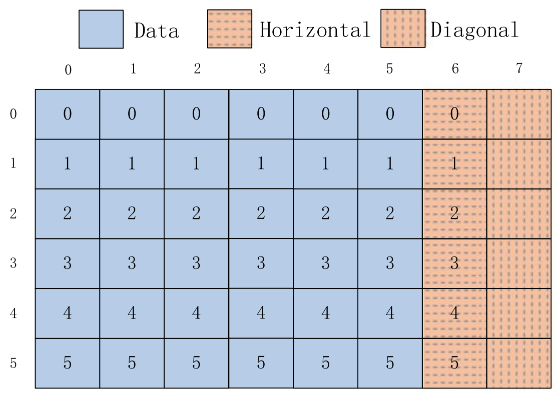
Parity Chain. A parity chain is composed of a parity element and other corresponding elements that are used to compute this parity element. In Fig. 1a $\{C_{0,0}, C_{0,1}, \ldots, C_{0,6}\}$ is a specific parity chain.

Horizontal Parity Chain. A kind of parity chain that contains a parity element and some continuous data elements. The parity element of horizontal parity chain is called horizontal parity element. Since the horizontal parities in existing RAID-6 codes are simply calculated by all data elements of each row, in some papers the "horizontal parity" has been called "row parity" as well.

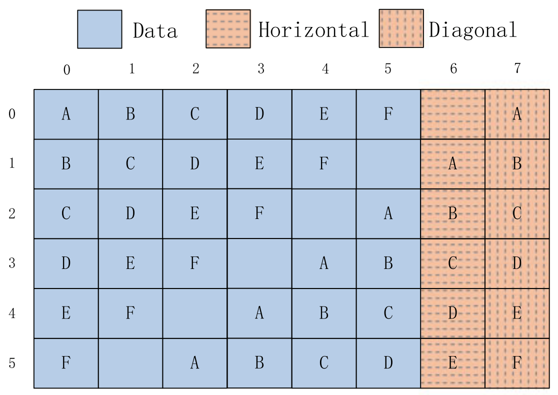
Diagonal Parity Chain. Another kind of parity chain where the data elements are dispersed. Each diagonal parity chain contains a diagonal parity element and its related elements, where the diagonal parity element is computed by those related elements.

2.2 Related Work

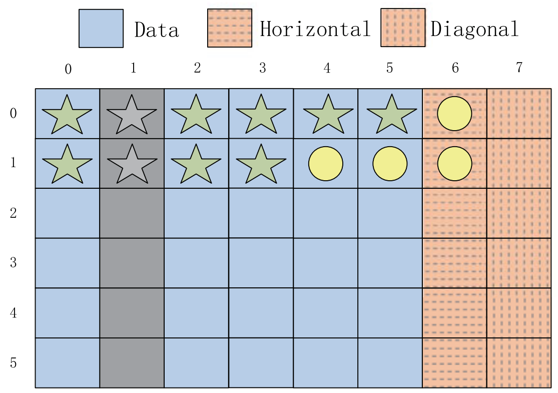
There exist a lot of RAID-6 implementations, of which a typical class of implementations is to use various MDS erasure codes, including Reed-Solomon code [20], Cauchy Reed-Solomon code [21], RDP code [7], EVENODD code [8], Liberation code [17], Liber8tion code [22], Blaum-Roth code [23], X-Code [9], HDP code[4], B-Code [24], P-Code [37], Balanced P-Code [38], H-Code [15], Cycle code [25], and C-Code [35]. Some non-MDS erasure codes and some new kinds of erasure codes such as Hover code [26], WEAVER code [27], LRC code [13], Flat XOR-Code [28], Generalized X-Code [29], SD code [33] and STAIR code [34] are other



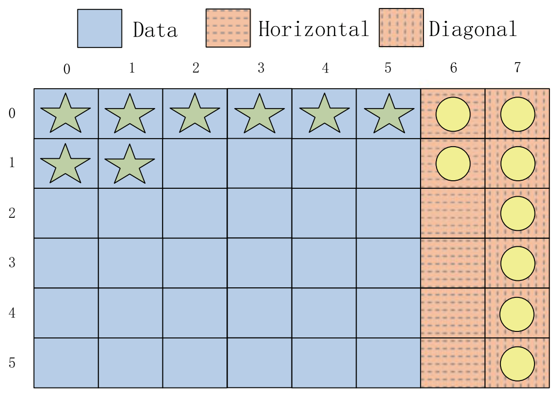
(a) Layout and horizontal chains (The elements with the same number mean that their belong to the same horizontal chain. E.g., $C_{1,6} = C_{1,0} \oplus C_{1,1} \oplus C_{1,2} \oplus C_{1,3} \oplus C_{1,4} \oplus C_{1,5}$).



(b) Layout and diagonal chains (The elements with the same letter indicate that their belong to the same diagonal chain. E.g., $C_{0,7} = C_{0,0} \oplus C_{1,6} \oplus C_{2,5} \oplus C_{3,4} \oplus C_{4,3} \oplus C_{5,2}$).



(c) A degraded read example when the second disk is temporarily unavailable (Star icons indicate the data elements that user requests, while round icons represent the elements that need to be extra read).



(d) A partial stripe write example (Star icons indicate the data elements that need to be written, while round icons represent the parity elements that need to be extra written).

Fig. 1. The problems in an RDP code with 8-disks (The last disk don't contribute to degraded reads, and encounters a high burden in partial stripe writes).

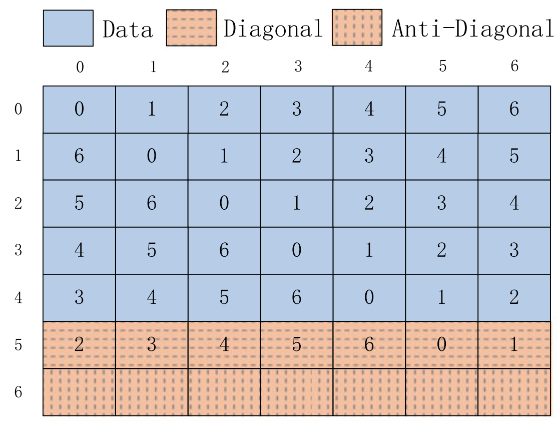
RAID-6 candidates. Furthermore, some network codes such as Zigzag code [31] and NCCloud [30] can be used for RAID-6 implementations, and achieve good performance on single disk failure recoveries. Particularly, Zigzag code is promising to achieve theoretical optimal performance for single failure recoveries, but doesn't address the problems on degraded reads and partial stripe writes. In this paper, we just focus on RAID-6 MDS array codes, which can be further categorized as horizontal codes and vertical codes.

Horizontal MDS Array Codes for RAID-6. Traditional RAID-6 horizontal array codes have k logical data disks and two parity disks, where the first parity disk usually stores the horizontal parity elements. EVENODD code is a typical horizontal code. In EVENODD code, horizontal parity elements are computed by the XOR sum of each row's data elements, and the diagonal parity elements are calculated by the data elements on diagonals. RDP code is another typical horizontal code for providing the optimal encoding/decoding computation complexity. In RDP code, the generation of horizontal parity elements is the same as EVENODD code (as Fig. 1a shows), while diagonal parity elements are generated by both data elements and horizontal parity elements on diagonals (as shown in Fig. 1b). Liberation, Liber8ation, and Blaum-Roth are lowest-density codes and achieve the near-optimal update complexity.

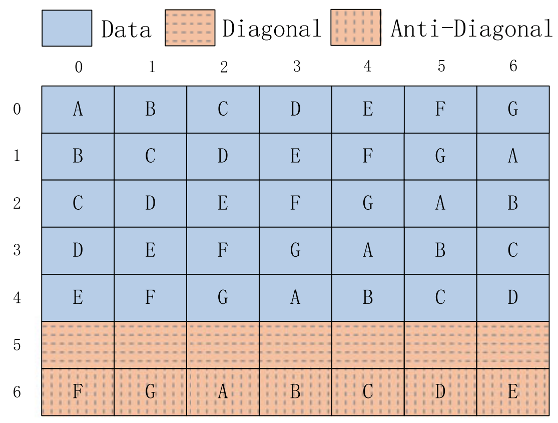
Vertical MDS Array Codes for RAID-6. Vertical array codes often distribute parity elements across disks, such as X-Code, Cycle code, C-Code, and HDP Code. A stripe of X-Code can be represented as a $p \times p$ two dimensions array (p needs to be a prime number), where the first $(p-2)$ rows store data elements, and the last two rows keep parity elements. There are two kinds of parity elements in X-Code: diagonal parity elements which are generated by the XOR sum of diagonal data elements and stored in the $(p-1)$th row, and anti-diagonal parity elements which are generated by anti-diagonal data elements and stored in the pth row. Cycle code is another vertical code that supports the number of disks with $(p-1)$ or $2 \times (p-1)$. Unlike typical vertical codes, HDP code uses both horizontal parity chains and diagonal parity chains, in order to optimize I/O load balancing and the disk failure recoveries. In the following of this paper, we treat both typical diagonal parity chains and anti-diagonal parity chains as diagonal parity chains.

2.3 The Degraded Read Problem in Existing RAID-6 MDS Array Codes

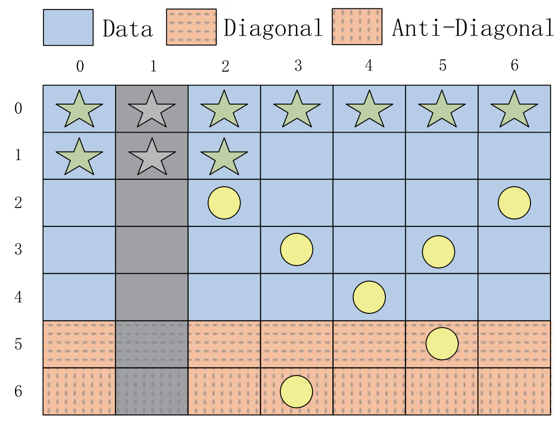
When a disk fails, since the failure is not always detected as soon as it occurred and the recovery process is often triggered by the act of man, the storage system usually recovers the lost information after a period of degraded running,



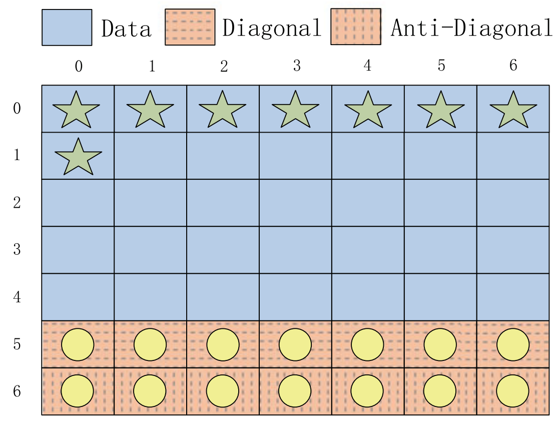
(a) Layout and diagonal chains (The elements with the same number mean that their belong to the same diagonal chain. E.g., $C_{5,5} = C_{0,0} \oplus C_{1,1} \oplus C_{2,2} \oplus C_{3,3} \oplus C_{4,4}$).



(b) Layout and anti-diagonal chains (The elements with the same letter indicate that their belong to the same anti-diagonal chain. E.g., $C_{6,2} = C_{0,0} \oplus C_{1,6} \oplus C_{2,5} \oplus C_{3,4} \oplus C_{4,3}$).



(c) A degraded read example when the second disk is temporarily unavailable (Star icons indicate the data elements that user requests, while round icons represent the elements that need to be extra read).



(d) A partial stripe write example (Star icons indicate the data elements that need to be written, while round icons represent the parity elements that need to be extra written).

Fig. 2. The problems in an X-Code with 7-disks (The system need to extra read/write more elements than horizontal codes for degraded reads and partial stripe writes).

while degraded reads are used to respond users' read requests during this time. Since the users' experience is a significant performance metric for storage systems, the performance on degraded reads is an important metric as well. However, most of existing RAID-6 codes cannot provide satisfied performance on degraded reads.

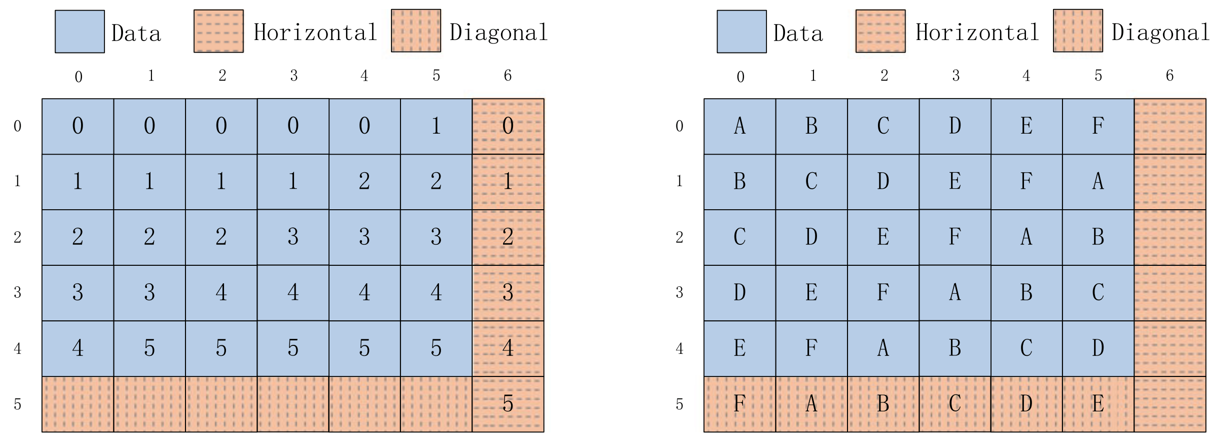
Figs. 1c and 2c show two examples of degraded reads in RDP-Code and X-Code when the second disk becomes temporarily unavailable. In these cases, when a user requests to read 10 data elements (starting with $C_{0,0}$), RDP code just needs to read 12 elements in total, while X-Code requires to retrieve 15 elements. What's more, it is easy to observe that all disks in X-Code contribute to degraded reads, while the diagonal parity disk in RDP code cannot help degraded reads. Since the RAID-based storage systems usually read data in parallel, one more contributed disk will effectively speed up the performance.

In Summary, horizontal codes like RDP code usually need to read less elements, while vertical codes such as X-Code have more contributed disks. Some vertical codes such as H-Code code have horizontal chains, but they also lay two parities in each row and one of them cannot contribute to degraded read, thus degrade the performance. Though parity disk failures for horizontal codes don't degrade the read speed, they won't affect the overall performance due to the fact that the possibility of parity disk failures is very low. Therefore, if there existing an erasure code that just lays one horizontal parity in a row, the degraded read performance will be further improved.

2.4 The Partial Stripe Write Problem in Existing RAID-6 Array MDS Codes

Partial stripe writes to continuous data elements are common in storage system, particularly in write-intensive applications like online storage systems. Since these systems usually upload the local data with any size into erasure coded systems in real time, which usually trigger one to a half stripe of data element's update and causes partial stripe writes frequently occurring. However, most of RAID-6 erasure codes suffer from high I/O cost or low write speed on partial stripe writes. We now illustrate these problems as follows.

Figs. 1d and 2d give two examples of partial stripe writes in RDP-Code and X-Code with eight continuous data elements (starting with $C_{0,0}$). It is easy to see that though RDP code only requires to update 8 extra parity elements, there



(a) Horizontal parities layout and construction rules (The elements with the same number mean that their belong to the same horizontal parity chain. E.g. $C_{1,6} = C_{0,5} \oplus C_{1,0} \oplus C_{1,1} \oplus C_{1,2} \oplus C_{1,3}$).

(b) Diagonal parities layout and construction rules (The elements with the same letter indicate that their belong to the same diagonal parity chain. E.g. $C_{5,1} = C_{0,0} \oplus C_{1,5} \oplus C_{2,4} \oplus C_{3,3} \oplus C_{4,2}$).

Fig. 3. The layout and construction rules of Short Code.

are six elements gathering on the diagonal parity disk (disk-7) causing a bottleneck and significantly degrading the whole system's performance, because the write speed is due to the slowest disk. On the other hand, X-Code provides good balancing on partial stripe writes, but it needs to update more parity elements (in this case is 14) due to the fact that continuous data elements are hard to share the common diagonal parity chain.

In summary, most vertical codes such as X-Code usually suffer from high I/O cost, while horizontal codes like RDP code provide unbalanced I/O on partial stripe writes. Though some dynamic balancing methods (such as rotating the maps from logical disks to physical disks like RAID-5) are able to reform the I/O balancing under intensive write operations for horizontal codes, these methods are not applicable for improving the partial stripe write speed due to the fact that they are designed among different stripes but the problem is occurred inter-stripe. Therefore, for each stripe, if we can keep the horizontal parities in a specific disk and distribute all diagonal parities among disks, the partial stripe write performance will be significantly improved.

3 Short Code

In this section, we present the detailed designs of our proposed Short Code. Unlike other MDS codes, the horizontal parity elements of Short Code are computed by the XOR sum of some continuous data elements rather than all data elements of each row. In Short Code, all horizontal parity elements are stored in a specific parity disk, while diagonal parity elements are distributed across disks.

3.1 Layout and Construction Rules

A stripe of Short Code consist of $(n-2) \times (n-1)$ data elements and $2 \times (n-1)$ parity elements (total is $n \times (n-1)$ elements) distributed on n logic disks. Like most RAID-6 vertical codes, the amount of disks n needs to be a prime number. The storage efficiency ratio of Short Code is $(n-2)/n$, because the total number of elements in one stripe is $n \times (n-1)$ and the number of data elements in one stripe is $(n-2) \times (n-1)$. For example, when $n = 7, 11, 13$, the storage efficiency ratio of Short Code is $5/7$, $9/11$, $11/13$, respectively. Fig. 3 shows an example of Short Code with 7 disks. In this case, each disk contains 6 elements. There are two kinds of parity elements: horizontal parity elements which are stored in the last disk (such as $C_{0,6}, C_{1,6}, \cdots, C_{5,6}$), and diagonal parity elements which are distributed across the first $n-1$ disks (such as $C_{5,0}, C_{5,1}, \cdots, C_{5,5}$).

Horizontal Construction Rules. Different from other MDS codes, the horizontal parities of Short Code are generated by some continuous data elements rather than all data elements of each row. The horizontal construction rules can be represented in mathematics as follows (notation $\oplus$ means XOR operation, and $<>$ means modular arithmetic operation).

$$C_{i,n-1} = \bigoplus_{j=0}^{n-3} C_{\lfloor \frac{i\times(n-2)+j}{n-1} \rfloor, \langle i\times(n-2)+j \rangle_{n-1}}, \quad (0 \leqslant i \leqslant n-2). \tag{1}$$

For easy to understanding, the horizontal construction rules can be represented in the following five steps:

- define *the next element of* $\boldsymbol{C_{i,j}}$ as: if $j \leqslant n-3$, the next element is $C_{i,j+1}$; otherwise, the next element is $C_{i+1,0}$ (i.e., the first data element of the next row).
- denote $C_{0,0}$ as the 0th element, the next element ($C_{0,1}$) as the 1th element, then the next as 2th element, etc., until the last data element ($C_{n-3,n-2}$) is denoted.
- label the 0th to $(n-3)$th element as number '0', the $(n-2)$th to $((n-2)+(n-3))$th element as number '1', and so on, until all data elements are labeled.
- sequentially label each element of the last disk as '0', '1', ..., 'n−2'.
- For each labeled number, calculate the XOR sum of its corresponding data elements, and then store the sum in its related parity element.

Fig. 3a shows an example of the horizontal construction rules in a 7-disks Short Code. It is easy to see that the $(n-2)$th to $((n-2)+(n-3))$th element are $C_{0,5}$, $C_{1,0}$, $C_{1,1}$, $C_{1,2}$, and $C_{1,3}$ (with the same number '1'), while their corresponding parity element is $C_{1,6}$. Therefore, $C_{1,6}$ can be calculated by the XOR sum of these data elements, i.e., $C_{1,6} = C_{0,5} \oplus C_{1,0} \oplus C_{1,1} \oplus C_{1,2} \oplus C_{1,3}$. Furthermore, We use H_i to denote each specific horizontal parity chain and give the following definition:

$$H_i = \Big\{C_{i,n-1}, C_{\lfloor\frac{i\times(n-2)}{n-1}\rfloor,\langle i\times(n-2)\rangle_{n-1}}, \ldots, C_{\lfloor\frac{i\times(n-2)+(n-3)}{n-1}\rfloor,\langle i\times(n-2)+(n-3)\rangle_{n-1}}\Big\}, \quad (0\leqslant i\leqslant n-2).$$

Obviously, this definition is another representation form of Equation (1), because each H_i contains and just contains all elements of each equation in Equation (1). We consider a specific ***H_i* is irrelevant with disk *j*** if and only if H_i doesn't contain any element of disk j, and give the following lemma to help understand horizontal parity chains.

Lemma 1. *H_i $(0\leqslant i\leqslant n-2)$ is irrelevant with disk $n-2-i$, and the data elements of H_i are the last i elements of row $i-1$ and the first $n-2-i$ elements of row i.*

Proof. As shown in Equation (1), each horizontal parity chain contains $n-2$ continuous data elements that start with $C_{\lfloor\frac{i\times(n-2)}{n-1}\rfloor,\langle i\times(n-2)\rangle_{n-1}}$ and locate on different disks. Since Short Code contains $n-1$ data disks, each horizontal parity chain is irrelevant with exact one data disk. We now discuss this lemma in two cases:

Case 1: $i=0$.

In this case, $C_{\lfloor\frac{i\times(n-2)}{n-1}\rfloor,\langle i\times(n-2)\rangle_{n-1}}$ is actual $C_{0,0}$, thus the data elements of H_0 are actual the first $n-2$ elements of row 0. Since the first $n-2$ elements of row 0 belong to disk 0 to $n-3$, H_0 is irrelevant with disk $n-2$, which equals $n-2-i$.

Case 2: $1\leqslant i\leqslant n-2$.

In this case, due to $i\times(n-2) = (i-1)\times(n-1) + (n-1-i)$ and $0 < n-1-i < n-1$, $C_{\lfloor\frac{i\times(n-2)}{n-1}\rfloor,\langle i\times(n-2)\rangle_{n-1}}$ can be simplified to $C_{i-1,n-1-i}$. Therefore, the data elements of H_i are $C_{i-1,n-1-i}, C_{i-1,n-i}, \ldots, C_{i-1,n-2}$ (the last i elements of row $i-1$), and $C_{i,0}, C_{i,1}, \ldots, C_{i,n-3-i}$ (the first $n-2-i$ elements of row i). Since these data elements are taken out from disk $n-i-1, n-i-2, \ldots, n-2$ and $0, 1, \ldots, n-3-i$, H_i does not contain disk $n-2-i$. □

We can easily observe that the number of data elements of each horizontal parity chain is less than those of each row, which is different from other RAID-6 MDS codes. In this paper we call this kind of parity chains *Short Horizontal Parity Chain*.

Diagonal Construction Rules. the diagonal construction rules can be presented as

$$C_{n-2,i} = \bigoplus_{j=0}^{n-3} C_{j,\langle n-2+i-j\rangle_{n-1}}, \quad (0\leqslant i\leqslant n-2). \tag{2}$$

Each diagonal parity chain contains a parity element and $n-2$ data elements that are located on a diagonal line with slope of -1. For example, as Fig. 3b shows, we can easily observe that the parity element $C_{5,1}$ and the data element $C_{4,2}$, $C_{3,3}$, $C_{2,4}$, $C_{1,5}$, and $C_{0,0}$ are located on the same diagonal line, thus $C_{5,1}$ can be computed by the XOR sum of these data elements.

3.2 Reconstruction Process

In this section, we discuss how does Short Code recovery from single disk failures and double disk failures.

Recovery from Single Failures. According to the construction rules, each horizontal parity chain or diagonal parity chain at most contains one element in each disk. Therefore, when a single disk fails, we can recover each lost element based on its corresponding horizontal parity chain or diagonal parity chain, either.

Algorithm 1. The Algorithm for Recovering Double Failures in Short Code

```
 1: Identify the two failed disks $f_1$ and $f_2$ $(0\leqslant f_1 < f_2\leqslant n-1)$.
 2: if $f_2 == n-1$ then
 3:     Recover all failed elements in $f_1$ based on Equation (2);
 4:     Recover all failed elements in $f_2$ based on Equation (1);
 5: else
 6:     Recovery $C_{n-2-f_2,f_1}$ and $C_{n-3-f_1,f_2}$ by Equation (1);
 7:     Two cases run synchronously:
 8:     Case 1: starting with $C_{n-2-f_2,f_1}$ repeat
 9:         Recover a missing element based on Equation (2);
10:         Recover another lost element based on Equation (1);
11:     until The parity element $C_{n-1,f_2}$ has been recovered.
12:     Case 2: starting with $C_{n-3-f_1,f_2}$ repeat
13:         Recover a missing element based on Equation (2);
14:         Recover another lost element based on Equation (1);
15:     until The parity element $C_{n-1,f_1}$ has been recovered.
16: end if
```

Recovery from Double Failures. We denote the two failed disks as f_1 and f_2, where $0\leqslant f_1 < f_2\leqslant n-1$. If $f_2 = n-1$, we can recover all the lost elements of f_1 based on the diagonal parity chains, because each diagonal chain only contains exact one element in disk 0 to disk $n-2$. Afterward, we can recover all the missing elements of f_2 by recalculating all the horizontal parity information.

We then discuss the case of $0\leqslant f_1 < f_2\leqslant n-1$. According to Lemma 1, two data elements C_{n-2-f_2,f_1} and C_{n-3-f_1,f_2} can be directly recovered by horizontal parity chains, because their related horizontal parity chains are only relevant with exact one failed disk. Afterwards, based on C_{n-3-f_1,f_2} and its related diagonal parity chain, we can recover a lost element ($C_{\langle n-3-f_1+(f_2-f_1)\rangle_{n-1},f_1}$), and then reconstruct the next element based on horizontal parity chains. Following this method and starting with C_{n-2-f_2,f_1} and C_{n-3-f_1,f_2}, we can reconstruct all failed elements based on Algorithm 1.

Fig. 4 shows an example for recovery from disk 2 and 3 concurrent failures in 7-disks Short Code. As it shows, we first recover the two starting elements $C_{2,2}$ and $C_{2,3}$ (Identifying by the star icons) based on horizontal parity chains (the same number '2' and '3') respectively, and then reconstruct $C_{1,3}$ based on $C_{2,2}$ and its related diagonal parity chain (the same letter 'E'). Following this method, we can reconstruct all failed elements following $C_{2,2} \to C_{1,3} \to C_{1,2} \to C_{0,3} \to C_{0,2} \to C_{5,3}$ and $C_{2,3} \to C_{3,2} \to C_{3,3} \to C_{4,2} \to C_{4,3} \to C_{5,2}$.

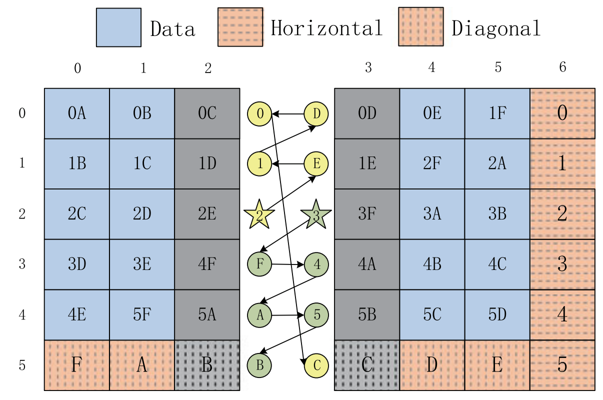


Fig. 4. Recovery from disk 2 and disk 3 concurrent failures.

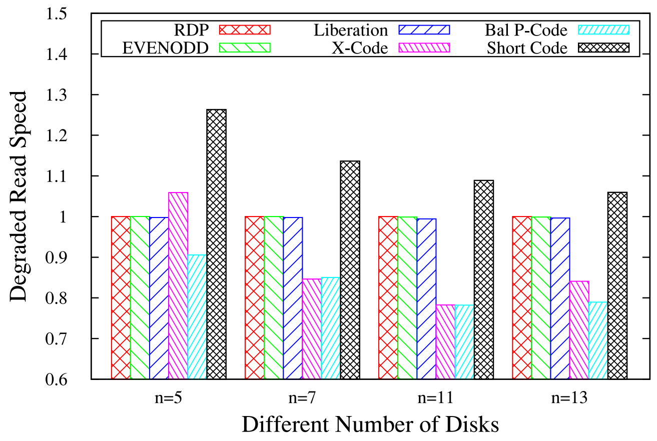


Fig. 5. The degraded read speed for different RAID-6 codes.

3.3 The Optimal Properties

The Optimal Storage Efficiency. As introduced in Section 3.1, a stripe of Short Code have $(n-2) \times (n-1)$ data elements and $2 \times (n-1)$ parity elements. Therefore, the storage efficiency of Short Code is $(n-2)(n-1) \ / \ [(n-2)(n-1)+2(n-1)] = (n-2)/n$, which is the optimal storage efficiency for RAID-6 codes [6].

The Optimal Encoding Computational Complexity. From the construction rules, Short Code needs $n-2$ data elements to compute each parity element, thus the total number of XOR operations for encoding all parity elements is $2(n-1)(n-3)$, because there exist $2(n-1)$ parity elements and each parity element should be calculated by $n-3$ XOR operations from $n-2$ data elements. On the other hand, since a stripe of Short Code contains $(n-1)(n-2)$ data elements, the amount of XOR operations per data element is $2(n-1)(n-3)/(n-1)(n-2) = 2-2/(n-2)$, which is the optimal encoding computational complexity in RAID-6 MDS codes [7].

The Optimal Decoding Computational Complexity. According to Algorithm 1, Short Code needs to use all $2(n-1)$ parity chains to recover all failed elements from double disk failures, where each parity chain contains $n-1$ elements. In order to recover one lost element, we should to calculate the XOR sum of all $n-2$ survived elements caused $n-3$ XOR operations. Since one column of Short code contains $n-1$ elements, when two concurrent disk failure occurs, the total number of failed elements is $2(n-1)$. Therefore, the decoding computation complexity of Short Code is $2(n-1) \ (n-3)/[2(n-1)] = (n-3)$ per failed element, which is the optimal computation decoding complexity in RAID-6 MDS code [15].

The Optimal Update Complexity. In Short Code, each data element only belongs to one horizontal parity chain and one diagonal parity chain, while the parity elements are independent on each other. Therefore, for each data element's update, Short Code just needs to update exact two parity elements, which is the optimal update complexity in RAID-6 [9].

4 Experiment Evaluations

In this section, we conduct a number of experiments to evaluate the performance in different RAID-6 MDS array codes under the metrics of degraded reads, partial stripe writes, single disk failure recoveries, and normal reads. We implement each code based on Jerasure-1.2 library [18], which is an open source library and commonly used in erasure code community [32]. We select typical horizontal codes (RDP code, EVENODD code, and Liberation code) and typical vertical codes (X-Code and Balanced P-Code) for comparison. We evaluate the real performance when RAID-6 codes are deployed on the number of disks $n = 5, 7, 11, 13$ (limited by the construction rules, Balanced P-Code is deployed on $n-1$ disks). Furthermore, in order to clarify the gap between Short Code and other codes, most of our evaluation results normalize the value of RDP code as *one*.

4.1 Environment

All our experiments are run on a machine with Intel Xeon X5472 processor and 8 GB of RAM. All the 16 SAS disks are setting in a disk array. The disk type is Seagate/Savvio 10 K.3, and the model number is ST9300603SS. Each disk has 300 GB capability and 10,000 rpm. The operation system of the machine is Red Hat with Linux fs91 3.2.16.

4.2 Degraded Read Evaluation

We evaluate the degraded read performance by two aspects: the degraded read speed and the influence of different read size, in order to illustrate that Short Code achieves good performance on degraded reads.

Degraded Read Speed. We evaluate the degraded read speed based on a series of experiments: since each RAID-6 code contains n (n denote the number of data disks) different data disk failure cases, we build 200 experiments for every possible failure case. In each experiment, we generate two random numbers as the start point and the read size, where the start point may be an arbitrary data element in the stripe, and the value of read size is from 1 to 20 elements [11].

Fig. 5 illustrates the average degraded read speed in different RAID-6 codes. As the figure shows, Short Code gains 6.1 to 26.3 percent higher degraded read speed than horizontal codes. This is because each row of Short Code contains $n-1$ data elements and one horizontal parity elements, but each row of horizontal codes contains $n-2$ data elements, one horizontal parity element, and one diagonal parity element. Since only data elements and horizontal parity elements contribute to degraded reads and all disks in RAID architecture are read in parallel, one more contributed disk makes Short Code provide higher speed on degraded reads than horizontal codes. On the other

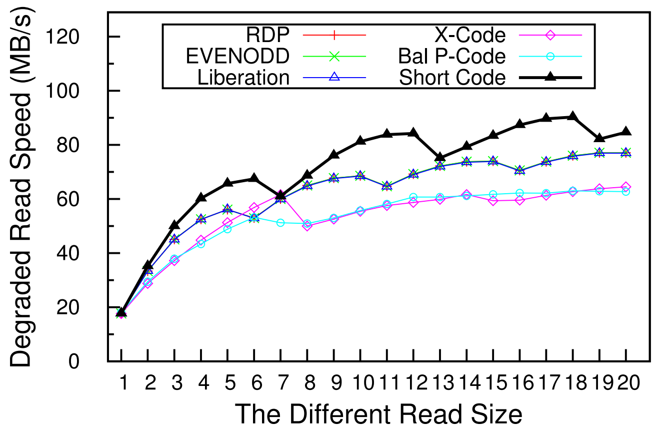


Fig. 6. The degraded read speed with different read size when n=7.

hand, Short Code achieve 19.3 to 39.1 percent higher degraded read speed than X-Code and 33.7 to 39.5 percent higher read speed than Balanced P-Code, because the continuous data elements in X-Code and Balanced P-Code are hard to share the common parity chains.

X-Code provides 15.4 to 21.7 percent lower speed than RDP code in all cases except $n = 5$, because all survived disks in X-Code contribute to degraded reads while the Q disk in horizontal codes cannot help to degraded reads. When n equals to 5, one more contributed disk is more significant than the less elements that need to be read. In other cases, the degraded read speed of X-Code is much lower than horizontal codes.

The Influence of Read Size. In order to analyze the influence of read size for degraded read speed, we evaluate the read size from 1 to 20 data element for each erasure pattern and each potential start element when $n = 7$, and show the average degraded read speed for each read size in Fig. 6. Compared to horizontal codes, Short Code provides 0.1 to 0.4 percent higher degraded read speed when the read size equals to 1, 6, 13, and achieves 5.7 to 29.8 percent higher speed for other cases of read size. Compared to X-Code and Balanced P-Code, Short Code provides up to 46.5 and 45.8 percent higher degraded read speed for all tested cases. The results matches Fig. 5 closely and illustrates that Short Code performs well on degraded reads.

4.3 Partial Stripe Write Evaluation

We run 2,000 experiments to evaluate the partial stripe write performance. For each experiment, we randomly generate two numbers as start point and write size, where the start point is an arbitrary data element in the stripe. Since some erasure codes besides our Short Code achieve the optimal update complexity and all MDS codes have the optimal full stripe write performance, the evaluated write size is from two to a half stripe of data elements (e.g., in 7-disks X-Code, the write size is from 2 to 17). We use both write speed and write cost to evaluate the performance, because large write cost will easily accelerate the burn-in of many storage devices, for example, SSD.

Partial Stripe Write Speed. We evaluate the real partial stripe write speed for different RAID-6 codes, and show the results in Fig. 7. As the figure shows, the partial write speed in Short Code is 36.2 to 80.3 percent higher than RDP code, 40.1 to 86.9 percent higher than EVENODD code, 32.8 to

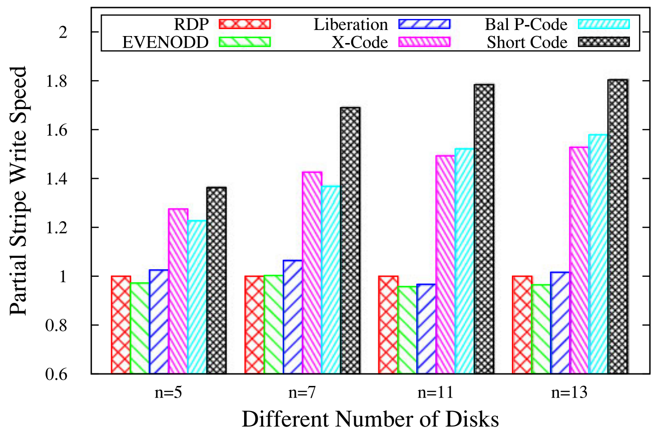


Fig. 7. The partial stripe write speed for different RAID-6 codes.

84.4 percent higher than Liberation code, 6.8 to 19.5 percent higher than X-Code, and 11.0 to 23.6 percent higher than Balanced P-Code.

Partial Stripe Write Cost. We define the *partial stripe write cost* as the ratio between the number of total elements (that need to be written) and the number of data elements (that need to be written). For example, as Fig. 2d shows, in order to handle the 8 elements' partial stripe write, the system totally needs to write 22 elements, thus the partial stripe write cost is $22/8 = 2.75$. Fig. 8 shows the average partial stripe write cost in different RAID-6 codes, where Short Code is a little lower write cost than the tested horizontal codes and Balanced P-Code. X-Code suffers from high write cost, which may accelerate the burn-in of some crisp devices, like SSD.

Fig. 7 doesn't match with Fig. 8, because the write speed is determined by the slowest disk rather than the total write cost. We show the average write cost on the heaviest loaded disk in Fig. 9, which shows that Short Code, X-Code and Balanced P-Code provide less write cost on the heaviest loaded disk than other codes, and thus supply the higher write speed. In addition, with the number of disks growing, the limited bandwidth becomes another important factor for write speed. E.g., though Short Code provides much lower cost on the heaviest loaded disk, it only provides 80.3 percent higher speed than RDP code when $n = 13$, because the network bandwidth is not enough to transfer the information retrieved by disks in real time. In summary, combined with

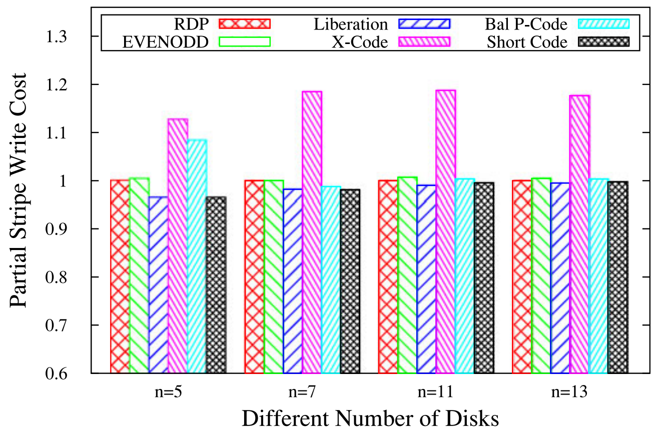


Fig. 8. The partial stripe write cost for different RAID-6 codes.

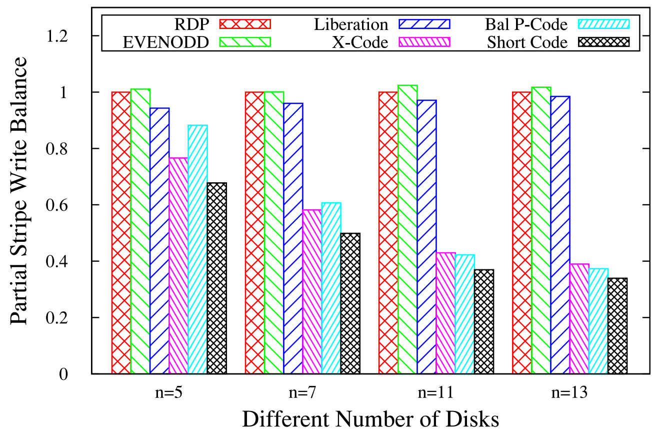


Fig. 9. The partial stripe write cost on the heaviest loaded disk for different RAID-6 codes.

low write cost, Short Code provides much better performance on partial stripe writes compared to other popular codes.

4.4 Further Discussion

We also evaluate the single disk failure recovery performance and normal read performance by the metric of real recovery speed and read speed, respectively. The aim of these evaluations is to illustrate that Short Code also provides acceptable performance for these operations.

Single Disk Failure Recovery Speed. We build 20 stacks to evaluate the single disk failure recovery speed, where each stack contains n stripes. For each code, we test all n potential disk failure patterns (including both data disk failures and parity disk failures) by U-Scheme in [36], because U-Scheme is fit for any XOR-Based erasure codes and provides very good recovery performance over disk array. Fig. 10 shows the average recovery speed for all n tested failure patterns, which illustrates that Short Code achieves a little higher recovery speed compared to the tested horizontal codes and X-Code.

Balanced P-Code achieves very high recovery speed when $n = 5$ and $n = 7$, but only provides a little lower recovery speed than Short Code when $n = 11$ and $n = 13$, because a stripe of Balanced P-Code only contains 2 rows and 3 rows when $n = 5$ and $n = 7$, and the heaviest loaded disk just need to afford 1 and 2 elements for recovery the lost information of one stripe, respectively. With the number of n growing, the ratio between the number of elements afforded by the heaviest loaded disk and the number of elements in each row is growing as well, and thus the recovery speed of Balanced P-Code is decreased, relatively.

On the other hand, the worse case of the test horizontal codes and Short Code is parity disk erased, which need to retrieve all survived data elements to reconstruct the lost information, but the worse case of X-Code and Balanced P-Code is data disk failure, which recovers more faster than parity disk failure cases. However, since data elements usually more important than parity elements and the possibility of parity disk failures is low, the recovery performance is mainly due to the average speed over all erasure cases. In summary, combine with the evaluation results above analysis, we can conclude that Short Code achieves acceptable single disk failure recovery performance.

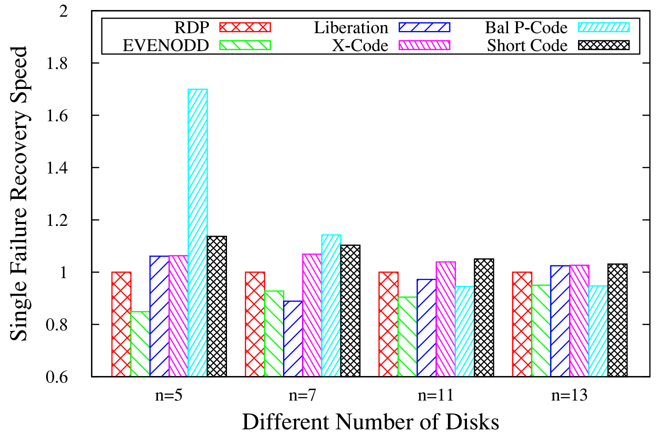


Fig. 10. The single disk failure recovery speed for different RAID-6 codes.

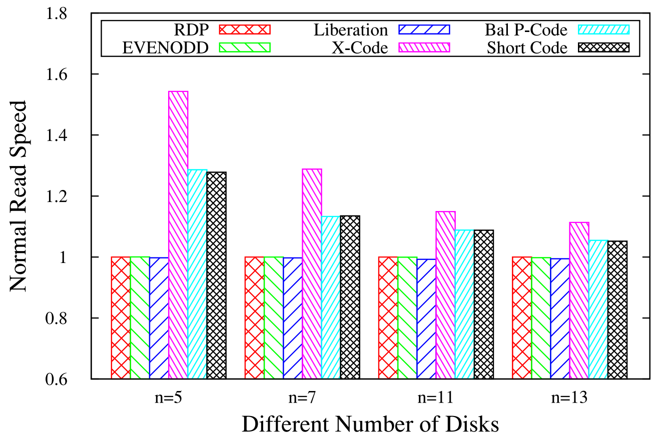


Fig. 11. The normal read speed for different RAID-6 codes.

Normal Read Speed. We conduct $2,000$ experiments for each RAID-6 code to evaluate the normal read speed without any erasure. For each experiment, we randomly generate the start point and read size, where the start maybe an arbitrary data element and the range of read size is from 1 to 20 elements. Fig. 11 shows average read speed of all tested codes. As shown, the read speed of Short Code is a little higher than the tested horizontal codes, very similar as Balanced P-Code, and a little lower than X-Code, which illustrates Short Code provides acceptable normal read performance as well.

5 Conclusion

In this paper, we propose a novel RAID-6 MDS erasure code, called Short Code. A stripe of Short Code consist of $n \times (n-1)$ elements evenly distributed on n logic disks, where n is a prime number. Different from previous RAID-6 codes, in Short Code, the number of data elements in each horizontal parity chain is less than the number of data elements in each row, while the continuous data elements still belong to adjacent horizontal chains. The diagonal parity elements are generated by anti-diagonal data elements, and distributed across disks. The properties analyses show that Short Code achieves the optimal storage efficiency, the optimal encoding/decoding computational complexity, and the optimal update complexity. The experiment results show that Short Code achieves good performance on degraded reads, partial stripe writes, single disk failure recoveries and normal reads, which illustrates Short Code is a good candidate for RAID-6 implementation.

Acknowledgments

We would like to greatly appreciate the anonymous reviewers for their insightful comments. This work is supported by the National Natural Science Foundation of China (Grant Nos. 61232003 and 61327902) and the State Key Laboratory of High-end Server and Storage Technology (Grant No. 2014HSSA02). Jiwu Shu is the corresponding author.

References

[1] S. Ghemawat, H. Gobioff, and S. Leung, “The Google file system,” in *Proc. 19th ACM Symp. Oper. Syst. Princ.*, 2003, pp. 29–43.

[2] B. Calder, et al., “Windows Azure system: A highly available cloud storage service with strong consistency,” in *Proc. 23rd ACM Symp. Oper. Syst. Princ.*, 2011, pp. 143–157.

[3] J. Kubiatowicz, “Oceanstore: An architecture for global-scale persistent storage,” in *Proc. 9th Int. Conf. Archit. Support Program. Languages Oper. Syst.*, 2000, pp. 190–201.

[4] C. Wu, et al., “HDP Code: A horizontal-diagonal parity code to optimize I/O load balance in RAID-6,” in *Proc. IEEE/IFTP 41st Int. Conf. Dependable Syst. Netw.*, 2011, pp. 209–220.

[5] B. Fan, W. Tanisiriroj, L. Xiao, and G. Gibson, “DiskReduce: RAID for data intensive scalable computing,” in *Proc. 4th Annu.Parallel Data Storage Workshop*, 2009, pp. 6–10.

[6] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. New York, NY, USA: North-Holland, 1977.

[7] P. Corbett, et al., “Row-Diagonal Parity for double disk failure correction,” in. *Proc. 3rd USENIX Conf. File Storage Technol.*, Mar. 2004, pp. 1–1.

[8] M. Blaum, J. Bruck, and J. Nebib, “EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures,” *IEEE Trans. Inf. Theory*, vol. 45, no. 1, pp. 46–59, Jun. 1999.

[9] L. Xu, and J. Bruck, “X-Code: MDS array codes with optimal encoding,” *IEEE Trans. Inf. Theory*, vol. 45, no. 1, pp. 272–276, Jan. 1999.

[10] 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. 5th USENIX conf. File Storage Technol.*, Feb. 2007, Article No. 1.

[11] 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. 10th USENIX conf. File Storage Technol.*, Feb. 2012, pp. 20–20.

[12] L. Xiang, Y. Xu, J. Lui, and Q. Chang, “Optimal recovery of single disk failures in RDP code storage systems,” in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2010, pp. 119–130.

[13] C. Huang, et al., “Erasure coding in Windows Azure storage,” in *Proc. USENIX Conf. Annu. Tech. Conf., Jun.*, 2012, pp. 2–2.

[14] D. Ford, “Availability in globally distributed storage systems,” in *Proc. 9th USENIX Conf. Oper. Syst. Des. Implementation*, 2010, pp. 61–74.

[15] C. Wu, S. Wan, X. He, Q. Cao and C. Xie, “H-Code: A hybrid MDS array code to optimize large stripe writes in RAID-6,” in *Proc. IEEE Parallel Distrib. Process. Symp.*, May 2011, pp. 782–793.

[16] D. Patterson, G. Gibson, and R. Katz, “A case for redunant arrays of inexpensive disks (RAID),” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1988, pp. 109–116.

[17] J. Plank, “The RAID-6 liberation codes,” in *Proc. 6th USENIX Conf. File Storage Technol.*, Feb. 2008, Art. no. 7.

[18] J. Plank, S. Simmerman, and C. Schuman, “Jerasure: A library in C/C++ facilitating erasure coding for storage applications-Version 1.2,” Univ. Tennessee, Knoxville, TN, USA, Tech. Rep. CS-08-627, Aug., 2008.

[19] J. L. Hafner, V. Deenadhayalan, T. Kanungo, and K. Rao, “Performance metrics for erasure codes in storage systems,” IBM Research, San Jose, CA, USA, Tech. Rep. RJ 10321, 2004.

[20] I. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *J. Soc. Ind. Appl. Math.*, vol. 8, pp. 300–304, 1960.

[21] J. Blomer, M. Kalfane, R. Krap, M. Karpinski, M. Luby, and D. Zuckerman, “An XOR-based erasure-resilient coding scheme,” *Int. Comput. Sci. Inst., Berkeley, CA, USA, Tech. Rep. TR-95-048*, Aug. 1995.

[22] J. Plank, “A new minimun density RAID-6 code with a word size of eight,” in *Proc. 7th IEEE Int. Symp.Netw. Comput. Appl.*, Jul. 2008, pp. 85–92.

[23] M. Blaum, and R. Roth, “On lowest density MDS codes,” *IEEE Trans. Inf. Theory*, vol. 45, no. 1, pp. 46–59, Jan. 1999.

[24] L. Xu, V. Bohossian, J. Bruck, and D. Wagner, “Low-density MDS codes and factors of complete graphs,” *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 1817–1826, Sep. 1999.

[25] Y. Cassuto and J. Bruck, “Cyclic lowest density MDS array codes,” *IEEE Trans. Inf. Theory*, vol. 55, vol. 4, pp. 1721–1729, Apr. 2009.

[26] J. Hafner, “HoVer erasure codes for disk arrays,” in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun., 2006, pp. 217–226.

[27] J. Hafner, “WEAVER codes: Highly fault tolerant erasure codes for storage systems,” in *Proc. 4th Conf. USENIX Conf. File Storage Technol.*, Dec. 2005, pp. 16–16.

[28] K. Greenan, X. Li, and J. Wylie, “Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs,” in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, 2010, pp. 1–14.

[29] X. Luo and J. Shu, “Generalized X-Code: An efficient RAID-6 code for arbitrary size of disk array,” *ACM Trans. Storage*, vol. 8, no. 3, Sep.,2012, Art. No. 10.

[30] Y. Hu, H. Chen, P. Lee, and Y. Tang, “NCCloud: Applying network coding for the storage repair in a cloud-of-clouds,” in *Proc. 10th USENIX Conf. File Storage Technol.*, 2012, pp. 21–21.

[31] I. Tamo, Z. Wang, and J. Bruck, “Zigzag codes: MDS array codes with optimal rebuilding,” *IEEE Trans. Inf. Theory*, vol. 59, no. 3, pp. 1597–1616, Mar., 2013.

[32] J. Plank, J. Luo, C. Schuman, L. Xu and Z. W. O'Hearn, “A performance evaluation and examination of open-source erasure coding libraries for storage,” in *Proc. USENIX 7th Conf. File Storage Technol.*, Feb.,2009, pp. 253–265.

[33] J. Plank, M. Blaum, and J. Hafner, “SD codes: Erasure codes designed for how storage systems really fail,” in *Proc. 11th USENIX Conf. File Storage Technol.*, Feb. 2013, pp. 95–104.

[34] M. Li and P. Li, “STAIR Codes: A general family of erasure codes for tolerating device and sector failures in practical storage systems,” in *Proc. 12th USENIX Conf. File Storage Technol.*, Feb. 2014, pp. 147–162.

[35] M. Li and J. Shu, “C-Codes: Cyclic lowest-density MDS array codes constructed using starters for RAID 6,” in *Proc. Comput. Res. Repository*, 2011.

[36] X. Luo and J. Shu, “Load-balanced recovery schemes for single-disk failure in storage systems with any erasure code,” in *Proc. 42nd IEEE Int. Conf. Parallel Process.*, Oct.,2013, pp. 552–561.

[37] C. Jin, H. Jiang, D, Feng, L, Tian, “P-Code: A new RAID-6 code with optimal properties,” in *Proc. 23rd Int. Conf. Supercomput.*, 2009, pp. 360–369.

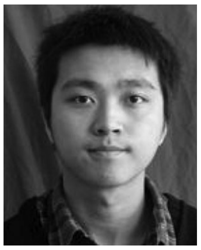
[38] P. Xie, J. Huang, Q. Cao, and C. Xie, “Balanced P-Code: A RAID-6 code to support highly balanced I/Os for disk arrays,” in *Proc. 9th IEEE Int. Conf. Netw. Archit. Storage*, 2014, pp. 133–137.

Yingxun Fu received the bachelor’s degree from North China Electric Power University in 2007, the master’s degree from Beijing University of Posts and Telecommunications in 2010, and the doctor’s degree from Tsinghua University in 2015. He is currently a lecturer in the Department of Computer Science and Technology, North China University of Technology. His current research interests include storage reliability and cloud/distributed storage systems.

Jiwu Shu received the PhD degree in computer science from Nanjing University in 1998, and finished the postdoctoral position research at Tsinghua University in 2000. Since then, he has been teaching at Tsinghua University. His current research interests include storage security and reliability, non-volatile memory based storage systems, parallel and distributed computing, and big data storage. He is a senior member of the IEEE.

Xianghong Luo received the bachelor's degree from the University of Electronic Science and Technology of China in 2009, the doctor's degree from Tsinghua University in 2014. He is currently an engineer in Google Company. His current research interests include erasure codes and distributed systems.

Zhirong Shen received the bachelor's degree from University of Electronic Science and Technology of China in 2010, and the doctor's degree from Tsinghua University in 2015. He is currently a lecturer in Fuzhou University. His current research interests include storage reliability and storage security.



Qingda Hu received the bachelors degree from Jilin University in 2013. He is currently a PhD candidate in Tsinghua University. His current research interests include memory and file system, GPU, and graph computing.