N-Code: An Optimal RAID-6 MDS Array Code for Load Balancing and High I/O Performance

Ping Xie,Zhu Yuan
xieping@qhnu.edu.cn
Qinghai Normal University
Xining, Qinghai, China

Jianzhong Huang
hjzh@hust.edu.cn
Huazhong University of Sci.& Tech.
Wuhan, Hubei, China

Xiao Qin
xqin@auburn.edu
Auburn University
Auburn, AL 36849, USA

ABSTRACT

Existing RAID-6 codes are developed to optimize either reads or writes for storage systems. To improve both read and write operations, this paper proposes a novel RAID-6 MDS array code called N-Code. N-Code exhibits three aspects of salient features: (i) *read performance*. N-Code assigns both horizontal parity chains and horizontal parities across disks, without generating a dedicated parity disk. Such a parity layout not only makes all the disks service normal reads, but also allows continuous data elements to share the same horizontal chain to optimize degraded reads; (ii) *write performance*. Diagonal parities are distributed across disks in a decentralized manner to optimize partial stripe writes, and horizontal parity chains enable N-Code to reduce I/O costs of partial stripe writes by merging I/O operations; and (iii) *balancing performance*. Decentralized horizontal/diagonal parities potentially support the I/O balancing optimization for single writes.

A theoretical analysis indicates that apart from the optimal storage efficiency, N-Code is featured with the optimal complexity for both encoding/decoding computations and update operations. The results of empirical experiments shows that N-Code demonstrates higher normal-read, degraded-read, and partial-stripe-write performance than the seven baseline popular RAID-6 codes. In particular, in the partial-stripe-write case, N-Code accelerates partial stripe writes by 32%-66% relative to horizontal codes; when it comes to degraded reads, N-Code improves degraded reads by 32%-53% compared to vertical codes. Furthermore, compared to the baseline codes, N-Code enhances load balancing by a factor anywhere between 1.19 to 9.09 for single-write workload, and between 1.3 to 6.92 for read-write mixed workload.

CCS CONCEPTS

• **Information systems** → *Distributed storage*; • **Computer systems organization** → *Redundancy*.

KEYWORDS

RAID-6; MDS Array Codes; Normal/Degraded Reads; Single/Partial-Stripe Writes

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICPP 2019, August 5–8, 2019, Kyoto, Japan
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6295-5/19/08…$15.00
https://doi.org/10.1145/3337821.3337829

ACM Reference Format:
Ping Xie,Zhu Yuan, Jianzhong Huang, and Xiao Qin. 2019. N-Code: An Optimal RAID-6 MDS Array Code for Load Balancing and High I/O Performance. In *48th International Conference on Parallel Processing (ICPP 2019), August 5–8, 2019, Kyoto, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3337821.3337829

1 INTRODUCTION

Modern storage systems, such as GFS [11], WAS [4] and OceanStore [20], usually distribute data across hundreds and thousands of storage devices. Disk failure is a normal phenomena in large-scale paralleled and distributed storage systems [42]. Redundant Arrays of Independent Disks or RAID [25] was proposed to provide high reliability and high I/O performance with acceptable spatial and monetary cost. Evidence shows that the possibility of disk failure occurrence grows with the increasing scale of storage systems [34][26]. In the view of high storage reliability and availability, RAID-6 is widely deployed in modern storage clusters and data centers to tolerate any two concurrent disk failures [38][8].

There exist a handful of RAID-6 implementations powered by various erasure codes. For example, the Maximum Distance Separable codes or *MDS* [24] achieve excellent storage efficiency. From the perspective of parity layout, RAID-6 MDS codes can be categorized into horizontal MDS codes and vertical MDS codes. Horizontal MDS codes (e.g., EVENODD code [1] and RDP code [6]) are constructed over $k+2$ logical disks, where the first k disks hold user data, and the last two disks keep parities. Vertical MDS codes (e.g., X-Code [44] and P-Code [18]) distribute parities across all the disks, in which data elements and parity elements are mixed.

Most existing RAID-6 MDS codes fall short in delivering satisfying read performance (i.e., normal and degraded reads), which is critical for modern storage systems. Horizontal MDS codes (e.g., EVENODD and RDP) own double special parity disks (i.e., horizontal parity disk 'P' and diagonal parity disk 'Q') in the normal mode. As such, both P and Q parity disks don't service normal reads, fending off read performance from paralleled I/O accesses in RAID-coded storage systems [25]. Even worse, Q parity disk can't carry out degraded reads in the case of a single disk failure. In normal/degraded reads, all the disks among a RAID system offer parallel read I/Os. Intuitively, the vertical MDS codes like X-Code and P-Code achieve good normal read performance; however, these MDS codes fail in offering good degraded-read performance. This is because continuous data elements to be accessed should share common horizontal parity chains rather than the same diagonal parity chains in X-Code, which triggers extra I/O cost. Other MDS codes (e.g., H-Code [39] and Short Code [10]) achieve poor normal-reads performance from special parity disk, whereas H-Code [39]

can't maintain good degraded-read performance due to the lack of contributions from diagonal parities to degraded reads.

In addition to reads, write performance (e.g., single writes [39], partial stripe writes and full stripe writes) is another key metric for RAID-coded storage systems. MDS code achieves optimal performance on full stripe writes. Horizontal MDS codes (e.g., EVENODD and RDP) have poor I/O balancing and high I/O cost on partial stripe writes due to own special parity disks and non-lowest density property [2]. Vertical MDS codes, like X-Code and Balanced P-Code [41], improve I/O balancing by assigning parities across disks under partial stripe writes; however, such MDS codes suffer from high I/O cost in the case of partial stripe writes. Moreover, H-Code and Short Code gain subpar I/O balancing due to special parity disk on single writes. Some dynamic load balancing strategies [32][33] and static rotating the mappings from logic disks to physical disks stripe by stripe like RAID-5 [29] may alleviate the unbalanced I/O problem to some extent. These schemes are inadequate to balance I/Os accessing the same stripe, due to the fact that each stripe has different access frequencies [38].

To address the aforementioned issues, we propose a novel RAID-6 MDS array code called N-Code. To improve normal/degraded read I/Os performance, N-Code assigns horizontal parity chains across all disks, whereas horizontal parities are distributed across disks to optimize I/O balancing in the single-write case. To boost partial stripe write performance, diagonal parities are dispersed over disks instead of residing in an independent disk. Different from the previous MDS codes, N-Code takes full advantage of decentralized horizontal/diagonal parities while keeping the merit of horizontal parity chains. Without independent parity disks, N-Code guarantees that all disks process normal reads and degraded reads, realizing I/Os optimization on both single writes and partial stripe writes. The main contributions of this study are summarized as below:

- We design N-Code - a novel RAID-6 MDS array code powered by both decentralized horizontal/diagonal parities and horizontal parity chain. N-Code aims to optimize I/O performance in terms of normal and degraded reads, single writes, and partial stripe writes.
- We conduct a thorough property analysis on N-Code. The analytic results illustrate that N-Code provides the high storage efficiency, low encoding/decoding computational complexity, and reduced update complexity.
- We evaluate the read/write performance and load balancing of N-Code in real experiments. The results show that N-Code achieves better I/O performance over the other popular RAID-6 codes in terms of load balancing, normal reads, degraded reads and partial stripe writes. For example, N-Code speeds up by up to 9.09x for load balancing ratios, improves by up to 53% for degraded read speed and provides up to 66% higher partial stripe write speed over popular RAID-6 codes under P=5, 7 and 11.

The remaining of the paper is organized as follows: Section 2 states the background and motivation. The layout and reconstruction rule of N-Code are presented in Section 3. Section 4 and Section 5 shed light on load balancing and read/write performance evaluations. Finally, we conclude the paper in Section 6.

2 BACKGROUND AND PROBLEM STATEMENT

Though various RAID-6 MDS codes have been reported in the literature, most of these codes still deliver unsatisfied read and write performance. In this section, we pinpoint these performance issues, followed by the motivation of our proposed RAID-6 code (i.e., N-Code).

2.1 Existing RAID-6 Codes

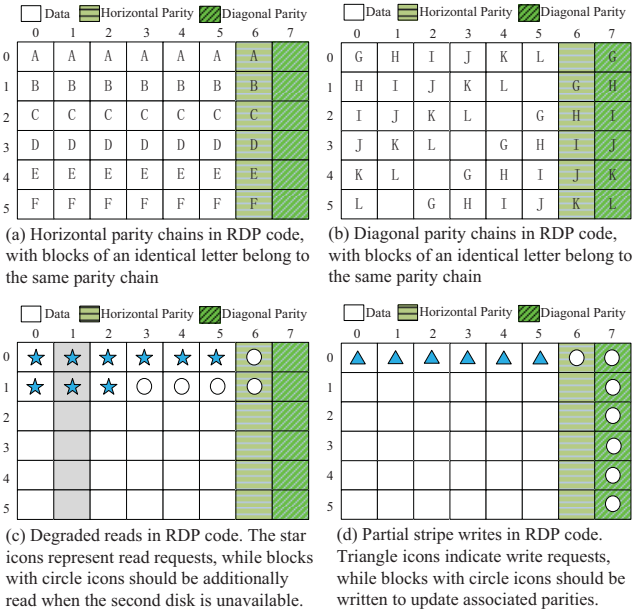
In recent years, there have been an enormous amount of research in the areas of RAID-6 implementations, among which RAID-6 MDS erasure codes attract much attention thanks to the optimal storage efficiency. For instance, Reed-Solomon code [31], Cauchy Reed-Solomon code [23], EVENODD code [1], RDP code [6], Blaum-Roth code [3], B-Code [43], Liberation code [27], Liber8tion code [28], Cyclic code [5], X-Code [44], P-Code [18], H-Code [39], HDP code [38], C-Codes [22], Short code [10], D-Code [9], and HV code [35]are categorized into RAID-6 MDS codes. On the other hand, RAID-6 non-MDS codes have been proposed to achieve good reconstruction performance, such as WEAVER [13], HOVER [14], Pyramid code [16], Flat XOR-Code [12], LRC [17], SD code [30], STAIR codes [21], Code-M [37], V^2-Code [40], and the like. Furthermore, some network codes (e.g., Zigzag code [36], NCCloud [15]) can also be used to enhance single-failure-recovery performance for RAID-6 storage. In addition, lowest density code is a special class of popular erasure codes of the optimal update complexity [45][3][27][28].

RAID-6 array codes exhibit low coding computational complexity, since such codes adopt simple XOR operations; MDS of the RAID-6 array codes achieve optimal storage efficiency. Therefore, in this paper we choose to focus on RAID-6 MDS array codes, which can be further classified into horizontal RAID-6 MDS array codes and vertical ones.

2.2 Read Problems in RAID-6 MDS Array Codes

Data reads constitute the majority I/O operations in various applications accessing today's storage systems. There are two kinds of read operations, namely normal reads and degraded reads. Recall the fact that if all disks are involved to handle read requests, then normal or degraded read performance is improved thanks to high disk access parallelisms. However, most existing RAID-6 codes fail in providing satisfied performance for read operations.

Normal Read: Normal reads are issued to continuous data block on operational disks. Many existing popular RAID-6 codes do not allow dedicated parity disks to be accessed in the normal mode, thus bringing forth insufficient parallel I/O accesses and degrading normal-read performance. For example, two dedicated parity disks in EVENODD and RDP make no contribute to normal reads (see for details in Fig. 1). Additionally, the dedicated horizontal parity disk in H-Code and Short-code can't be also accessed in normal mode. On the contrary, the array codes (e.g., X-Code and P-Code) distributing data elements across all disks can accomplish high I/O access parallelism, thereby improving normal-read performance.



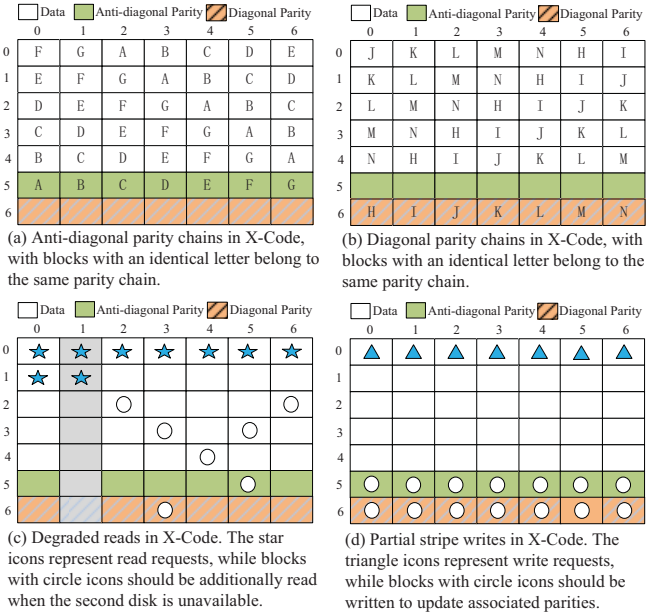
(a) Horizontal parity chains in RDP code, with blocks of an identical letter belong to the same parity chain

(b) Diagonal parity chains in RDP code, with blocks of an identical letter belong to the same parity chain

(c) Degraded reads in RDP code. The star icons represent read requests, while blocks with circle icons should be additionally read when the second disk is unavailable.

(d) Partial stripe writes in RDP code. Triangle icons indicate write requests, while blocks with circle icons should be written to update associated parities.

Fig. 1: Read and write problems in the 8-disk RDP codes, where dedicated parity disks don't service normal/degraded reads, and unbalanced I/Os occur in partial stripe writes.



(a) Anti-diagonal parity chains in X-Code, with blocks with an identical letter belong to the same parity chain.

(b) Diagonal parity chains in X-Code, with blocks with an identical letter belong to the same parity chain.

(c) Degraded reads in X-Code. The star icons represent read requests, while blocks with circle icons should be additionally read when the second disk is unavailable.

(d) Partial stripe writes in X-Code. The triangle icons represent write requests, while blocks with circle icons should be written to update associated parities.

Fig. 2: Read and write problems in 7-disk X-Code, where extra blocks are retrieved for degraded reads and partial stripe writes.

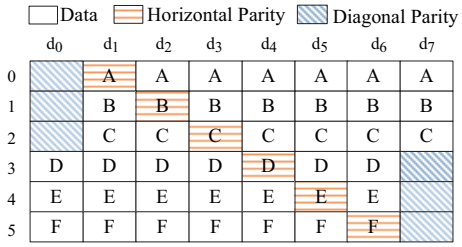
Degraded Read: Degraded reads occur when requested data blocks are located on failed disks. The horizontal-parity-chain layout enables requested data blocks to be more likely to share the same horizontal parity chain, thus reducing I/O cost and improving I/O performance on degraded reads. However, the diagonal parities in RAID-6 codes (e.g., EVENODD, RDP and H-Code) can not contribute to degraded reads under single-disk failures (as shown in Fig. 1). Some RAID-6 codes bring forth extra I/O costs in the case of degraded reads when the diagonal parity chain generating rule is employed to rebuild unavailable blocks. For example, as illustrated in Figs. 1(c) and 2(c), additional five blocks and seven blocks (i.e., circle icons) are required for degraded reads on continuous nine blocks (i.e., star icons) in RDP and X-Code, respectively. In a word, a RAID-6 code adopting only diagonal chains (e.g., X-Code) encounters higher I/O costs than that is characterized by horizontal chains (e.g., RDP) in the degraded-read case.

2.3 Write Problems in RAID-6 MDS Array Codes

Data writes are another essential operations in erasure-coded storage systems. Write operations bear additional I/Os, because writes need to update parities in a stripe. Unbalanced I/O loads and high I/O costs are usually posed by write operations in RAID-6 codes due to the parity-chain-constructing-rules. There exist usually three types of write operations - single writes, partial-stripe writes, and full-stripe writes. Among the three writes in MDS RAID-6 codes, full-stripe writes achieve the optimal write performance, whereas single writes and partial stripe writes are the dominant factors of write performance [19].

Single Write: Single write is a one-data-block-write request. Single writes in lowest-density RAID-6 codes (e.g., X-Code, P-Code, H-Code, Short code and D-Code) achieve optimal updating I/O complexity (a.k.a., updating I/O cost). That is, single writes only incur updating operations onto two associated parities. On the contrary, non-lowest-density RAID-6 codes (e.g., EVENODD and RDP) suffer from high updating I/O costs on single writes. Compared to non-lowest-density RAID-6 codes, lowest-density ones gain lower bound of updating I/O costs on single writes. As for lowest-density RAID-6 codes, if a large number of single writes occur, updating I/O operations will be gathered to the dedicated parity disks, thus leading to unbalanced I/O loads.

Partial Stripe Write: Partial stripe write refers to a write request to a part of continuous data blocks in one stripe. Partial stripe writes usually cause high updating I/O costs and unbalanced I/O loads. Although horizontal parity chains can reduce I/O costs of partial stripe writes by merging I/O operations, the dedicated diagonal parity disk induce unbalanced I/O loads (see RDP for details). To address such an I/O-unbalancing issue, a common practice is to distribute diagonal parities across disks (see H-Code and Short code for details). On the other hand, lowest-density RAID-6 codes (e.g., X-Code) may induce high I/O costs since they do not merge I/Os in diagonal parity chains. Take RDP and X-Code as examples. Fig. 1(d) shows that RDP triggers additional updating I/O operations on seven parity blocks (i.e., round icons) under partial stripe writes onto six continuous blocks (i.e., triangle icons), as such, diagonal parity disk holds heaviest I/O loads. Fig. 2(d) shows that X-code causes high updating I/O costs of fourteen parity blocks (i.e., circle icons) induced by partial stripe write onto seven continuous blocks (i.e., triangle icons).



(a) Horizontal parities layout and construction rules, with $C_{1,2} = C_{1,1}+C_{1,3}+C_{1,4}+C_{1,5}+C_{1,6}+C_{1,7}$.

	d0	d1	d2	d3	d4	d5	d6	d7
0	G		L	K	J	I	H	G
1	H	G		L	K	J	I	H
2	I	H	G		L	K	J	I
3	J	I	H	G		L	K	J
4	K	J	I	H	G		L	K
5	L	K	J	I	H	G		L

(b) Diagonal parities layout and construction rules, with $C_{0,0} = C_{1,1}+C_{2,2}+C_{3,3}+C_{4,4}+C_{5,5}+C_{0,7}$.

Fig. 3: The layout and construction rules of N-Code, with p = 7. Blocks with an identical letter belong to the same horizontal/diagonal parity chain.

3 N-CODE ARCHITECTURE

To address the above-mentioned read/write issues in the existing RAID-6 MDS codes, we propose a novel RAID-6 MDS array code - **N-Code** - to optimize read/write and I/O-balancing performance. N-Code is characterized by distributing horizontal parity blocks and diagonal parity blocks across all logic disks rather than dedicated logic disks.

3.1 Layout and Construction Rules

A stripe in N-Code consists of $(p\text{-}1)\times(p\text{-}1)$ data blocks and $2\times(p\text{-}1)$ parity blocks (i.e., $(p\text{-}1)\times(p+1)$ blocks in total), which are distributed across $p+1$ logic disks. We use $C_{i,j}$ to denote a block located on the ith row and j^{th} column in a stripe of N-Code, with $0 \le i \le (p\text{-}2)$ and $0 \le j \le p$. To meet the RAID-6 specification, the value of p should be a prime number. The storage efficiency ratio of N-Code is gauged as $(p\text{-}1)/(p+1)$, where the total number of blocks in one stripe is $(p\text{-}1)\times(p+1)$ and the number of data blocks is $(p\text{-}1)\times(p\text{-}1)$. For instance, the storage efficiency ratio of N-Code is 2/3, 3/4, 5/6, and 6/7 when p is set to 5, 7, 11, and 13, respectively. Fig. 3 depicts the layout of 'p=7' N-Code, where each disk holds six blocks (i.e., data blocks and parity blocks). There are two parity- block types: (i) horizontal parity blocks are placed across the medium p-1 disks (e.g., $C_{0,1}, C_{1,2}, C_{2,3}, C_{3,4}, C_{4,5}$ and $C_{5,6}$), and (ii) diagonal parity blocks are uniformly distributed across the first disk and the last disk (e.g., $C_{0,0}, C_{1,0}, C_{2,0}, C_{3,7}, C_{4,7}$ and $C_{5,7}$). Data blocks are dispersed across all the logic disks. Blocks tagged with the same letter belong to the same horizontal/diagonal parity chain.

N-Code constructs horizontal parity blocks and diagonal parity blocks using encoding equations (1) and (2), respectively. Here, symbol '+' or '$\sum$' indicates an XOR operation.

Horizontal Parities Construction Rules:

$$C_{i,i+1} = \begin{cases} \sum_{t=1}^{p-1} C_{i,(t+i)mod p+1}, & 0 \le i < (p\text{-}1)/2 \\ \sum_{t=1}^{p-1} C_{i,(t+i+1)mod p}, & (p\text{-}1)/2 \le i < (p\text{-}1) \end{cases} \quad (1)$$

Diagonal Parities Construction Rules:

$$\begin{cases} C_{i,0} = \sum_{t=1}^{p-2-i} C_{i+t,t} + \sum_{t=p-1-i}^{p-1} C_{(i+t)mod(p-1),t+1}, & 0 \le i < (p\text{-}1)/2 \\ C_{i,p} = \sum_{t=1}^{p-1-i} C_{i+t-1,t-1} + \sum_{t=p-i}^{p-1} C_{t-p-i,t}, & (p\text{-}1)/2 \le i < (p\text{-}1) \end{cases} \quad (2)$$

Fig. 3 shows the layout and construction rule of 'p=7' N-Code, with a $(p\text{-}1)$-row-by-$(p+1)$-column two-dimension matrix. Both horizontal and diagonal parity chain layouts are given in Figs. 3(a) and 3(b), respectively. Each parity chain holds one parity block and p-1 data blocks in one stripe in accordance with Eq. (1) or Eq. (2).

Fig. 3(a) depicts the horizontal parity chains of 'p=7' N-Code. We use letters to indicate different horizontal parity chains; blocks sharing an identical letter indicate that these blocks belong to the same horizontal parity chain. For instance, the blocks (e.g., $\{C_{1,1}, C_{1,2}, C_{1,3}, C_{1,4}, C_{1,5}, C_{1,6}$ and $C_{1,7}\}$) with the same letter 'B' mean they belong to the same horizontal parity chain. A horizontal parity chain include one parity block and p-1 data blocks, where the horizontal parity block is generated by using equation (1). For example, horizontal parity block $C_{1,2}$ is calculated from data blocks $\{C_{1,1}, C_{1,3}, C_{1,4}, C_{1,5}, C_{1,6}$ and $C_{1,7}\}$ according to expression $C_{1,2} = C_{1,1}+C_{1,3}+C_{1,4}+C_{1,5}+C_{1,6}+C_{1,7}$.

Fig. 3(b) plots the diagonal parity chains of 'p=7' N-Code. Similarly, blocks with an identical letter belong to the same diagonal parity chain. A diagonal parity chain consists of one parity block and p-1 data blocks, and the parity block is yielded by applying Eq. (2). For example, diagonal parity block $C_{0,0}$ is generated from data blocks $\{C_{1,1}, C_{2,2}, C_{3,3}, C_{4,4}, C_{5,5}, C_{0,7}\}$ via XOR operations (i.e., $C_{0,0} = C_{1,1}+C_{2,2}+C_{3,3}+C_{4,4}+C_{5,5}+C_{0,7}$).

3.2 Proofs of Correctness

To validate that N-Code has the ability to tolerate arbitrary two concurrent failures, we investigate the one-stripe-reconstruction case. This case is essential, because all stripes are independent of one another during encoding/decoding processes. We have the following lemma and theorem from the perspective of one stripe construction.

Lemma 1: There exists a sequence of a two-integer tuple (W_k, W'_k) that meets the following two conditions:

$$W_k = (p\text{-}2+(\frac{k+1}{2}+\frac{1+(-1)^k}{4}))mod(p\text{-}1)$$

$$W'_k = \frac{1+(-1)^k}{2}f_1+\frac{1+(-1)^{k+1}}{2}f_2,\ k \in \{0, 1, \cdots, 2p\text{-}3\}$$

With constraint '$0 < (f_2\text{-}f_1) < p+1$', all two-integer tuples $\{(0, f_1), (0, f_2), \cdots, (p\text{-}2, f_1), (p\text{-}2, f_2)\}$ occur only once in the sequence. The proof of this lemma can be found in the literature on the RAID-6 codes [1][44][18].

Theorem 1: One (p-1)-row-by-(p+1)-column stripe can be reconstructed from any two concurrent disk failures in N-Code.

Proof: Two failed disks are denoted as f_1 and f_2, with $0 \le f_1 < f_2 \le p$. According to the parity layout and construction rules of N-Code, we deduce that any two or more blocks in one parity chain do not reside in the same stripe. Meanwhile, anyone data block only takes part in the construction of exact two parity chains. For any two concurrent failures, two kinds of failed blocks exist, namely, independent failed blocks and dependent failed blocks. The reconstruction process of the two failed-block types are demonstrated as below.

Case 1: **Independent-failed-block Reconstruction.** Independent failed blocks mean that only one block failure occurs in one parity chain when two disks (e.g., f_1 and f_2) concurrently fail. For instance, independent failed block C_{i,f_1} or C_{i,f_2} at the ith row can be rebuilt according to N-Code's construction rules, because the other blocks (except C_{i,f_1} or C_{i,f_2}) in the corresponding parity chain survive in the case of two concurrent disk failures.

Case 2: **Dependent-failed-block Reconstruction.** Dependent failed blocks refer to a scenario where only two block failures happen in one parity chain in the double-disk failure case. Let us consider an example in which C_{i,f_1} and C_{i',f_2} in one parity chain are dependent failed blocks. There exist two situations: (i) failed blocks C_{i,f_1} and C_{i',f_2} belong to the same row, and (ii) failed blocks C_{i,f_1} and C_{i',f_2} are located in different rows in one stripe.

When dependent failed blocks C_{i,f_1} and C_{i',f_2} are not in the same row, they lie in the same diagonal parity chain. Once failed block C_{i,f_1} can be recovered, we can reconstruct the dependent failed block C_{i',f_2} via $C_{(i+f_2-f_1),f_2}$. Because the two blocks belong to the same diagonal parity chain, and the other blocks of the corresponding diagonal parity chain survive when two disks f_1 and f_2 concurrently fail. Similarly, if faulty block C_{i',f_2} can be recovered, we can reconstruct the dependent failed block C_{i,f_1} using $C_{(i'-f_2+f_1),f_1}$. The reason is that these blocks are residing in the same diagonal parity chain, and the remaining blocks in the corresponding parity chain still survive.

When dependent faulty blocks C_{i,f_1} and C_{i',f_2} are in the same row, they are located in the same horizontal parity chain (i.e., i=i'). If defective block C_{i,f_1} can be recovered, its dependent failed block C_{i,f_2} can be easily reconstructed because they belong to the same horizontal parity chain and the remaining blocks in the corresponding horizontal parity chain survive when two disks f_1 and f_2 are concurrently malfunctioning. Similarly, once failed block C_{i,f_2} can be recovered, then we can reconstruct the dependent failed block C_{i,f_1} from surviving blocks in a single horizontal parity chain.

From the above theoretical analysis, we conclude that all the failed blocks in one stripe of N-Code can be reconstructed when two disks concurrently become malfunctioning, and the reconstruction order is in accordance with the sequence of the two-integer tuples in *Lemma 1*. In summary, N-Code can tolerate any two concurrent disk failures.

3.3 Reconstruction Procedures

Both single-disk and double-disk reconstruction processes of N-Code are illustrated as below.

Recovery of Single-disk Failure: According to the parity layout and construction rules of N-Code, each horizontal parity chain or diagonal parity chain assigns at most one block to each disk. Therefore, when an arbitrary disk fails, all p-1 blocks in the faulty disk become independent failed blocks. Then, we can recover each failed block according to its associated horizontal parity chain construction rules (i.e., Equation (1)) or diagonal parity chain construction rules (i.e., Equation (2)).

Recovery of Double-disk Failures: We denote two concurrent failed disks as f_1 and f_2, with $0 \le f_1 < f_2 \le p$. According to N-Code's parity construction rules, each data block participates in the construction of two parity chains where all parity blocks are independent of each other. Therefore, when two disks f_1 and f_2 are out of order concurrently, there exist independent defective parity blocks, independent failed data blocks and dependent failed data blocks on failed disks.

As to malfunctioning disks f_1 and f_2, there exist 2p-2 failed blocks. We assume that there are x independent failed blocks, which correspond to x recovery chains. Arbitrary two or more independent faulty blocks don't exist in an identical recovery chain. For a recovery chain, the start-point is an independent failed data block or parity block. If the start-point C_{i,f_1} or C_{i,f_2} is an independent failed data block in a parity chain, then it must be a dependent failed data block joining in another parity chain. The reason is that anyone data block takes part in the construction of exact two parity chains. The end-point for each recovery chain is certainly a parity block.

Algorithm 1: The double-disk-failure recovery algorithm of N-Code.

```
: Identify two failed disks f1 and f2 (0 ≤ f1 < f2 ≤ p).
: Traverse the (p-1)×(p+1) matrix of N-Code to search out all
  independent failed blocks Bx, such as C_{i,f1} or C_{i,f2}.
: For each Bx do
: If Bx is a parity block then
: Generate Bx via Equation (1) or (2).
: Else
: Generate Bx(E.g., C_{i,f1}) via Equation (1) or (2).
: Repeat
: If C_{i,f1} and C_{i',f2} are at the same row then
: Generate C_{i,f2} via Equation (1);
: Generate C_{i,f1} via Equation (1);
: Else
: Generate C_{(i+f2-f1),f2} via Equation (2);
: Generate C_{(i'-f2+f1),f1} via Equation (2);
: End If
: Until the end-point of a recovery chain is a parity block.
: End if
: End For
```

Let us assume that two disks f_1 and f_2 concurrently become defective. Firstly, Algorithm 1 traverses the (p-1)×(p+1) two-dimensional matrix of N-Code to find out all independent failed blocks. Each faulty block corresponds to a recovery chain. Secondly, Algorithm 1 launches reconstruction process according to each recovery chain, and select the independent failed block (E.g., C_{i,f_1}) of each recovery chain as the start-point of the recovery process. If C_{i,f_1} is an independent failed parity block, compute it based on Equation (1) or (2); otherwise, it is an independent failed data block

regenerated from Equation (1) or (2). Afterwards, assure the corresponding to dependent failed block C_{i',f_2} of the failed block C_{i,f_1}. Thirdly, if C_{i,f_1} and C_{i',f_2} are at the same row, then Algorithm 1 recovers the C_{i,f_1} or C_{i,f_2} via Equation (1); otherwise, $C_{(i+f_2-f_1),f_2}$ or $C_{(i'-f_2+f_1),f_1}$ is recovered by using Equation (2). The reconstruction procedure of dependent failed blocks is repeated until the end-point of the recovery chain is a parity block, in this way, the reconstruction process of a recovery chain is completed. Following the above procedures in Algorithm 1, we can reconstruct all faulty blocks.

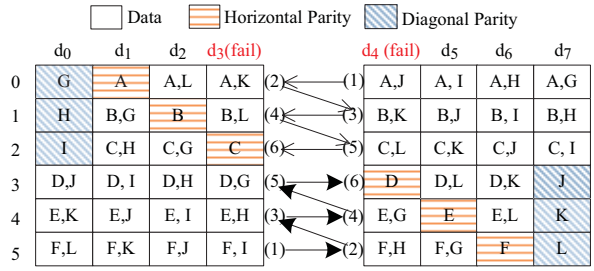


Fig. 4: Reconstruction process of N-Code under double concurrent failures, with disks d_3 and d_4 fail.

Fig. 4 shows a double-disk-recovery scenario in a 8-disk N-Code, with disks d_3 and d_4 concurrently fail. The recovery procedure is articulated as follows. Firstly, we identify two starting points of recovery chains (e.g., failed blocks $C_{5,3}$ and $C_{0,4}$). Secondly, we reconstruct failed blocks according to the corresponding recovery chains until the endpoints are reached (e.g., failed blocks $C_{3,4}$ and $C_{2,3}$). One recovery chain for failed blocks is '$C_{5,3} \rightarrow C_{5,4} \rightarrow C_{4,3} \rightarrow C_{4,4} \rightarrow C_{3,3} \rightarrow C_{3,4}$'; another one is '$C_{0,4} \rightarrow C_{0,3} \rightarrow C_{1,4} \rightarrow C_{1,3} \rightarrow C_{2,4} \rightarrow C_{2,3}$'.

3.4 The Features of N-Code

Optimal Storage Efficiency: Storage efficiency is a ratio between the number of data elements and that of total elements in a stripe. According to the constructing rule of N-Code, a stripe has (p-1)×(p-1) data elements and 2×(p-1) parity elements. Therefore, the storage efficiency of N-Code is (p-1)×(p-1)/[(p-1)×(p+1)] = (p-1)/(p+1), which is the optimal storage efficiency for RAID6-coded storage systems [24].

Optimal Encoding Computational Complexity: Encoding computational complexity refers to the average number of XOR operations for each data element during its encoding process. The N-Code's layout shows that there are p-1 horizontal parity chains and p-1 diagonal parity chains. Each parity chain holds just p-1 data elements and one parity element. According to the constructing rule of parity chains, the number of XOR operations in a parity chain is p-2 and; thus, there are 2(p-1)×(p-2) XOR operations during the encoding process. In addition, the number of data elements is (p-1)×(p-1) in a stripe of N-Code. Therefore, encoding computational complexity in N-Code is 2×(p-1)×(p-2)/[(p-1)×(p-1)] = 2(p-2)/(p-1), which is the optimal encoding computational complexity in MDS RAID-6 codes [6].

Optimal Decoding Computational Complexity: Decoding computational complexity is a ratio between the number of XOR operations and parity elements in a stripe during decoding process. A decoding operation is an inverse process of an encoding one in RAID-6 codes and; hence, the number of XOR operations is 2(p-1)×(p-2) during the decoding process of N-Code. Additionally, the number of parity elements is 2(p-1) in a stripe of N-Code. Therefore, the decoding computational complexity is 2(p-1)×(p-2)/[2(p-1)] = (p-2), which is the optimal decoding computational complexity in MDS RAID-6 codes [39].

Optimal Updating Complexity: In N-Code, each data element only belongs to one horizontal parity chain and one diagonal parity chain, whereas all parity elements are independent of each other. Hence, when one data element is modified, it is required for N-Code to update exact two parity elements - the optimal update complexity in RAID-6 codes [44].

4 LOAD BALANCING ANALYSIS

In this section, we conduct a group of experiments to comparatively evaluate the load balancing of N-Code and the popular RAID-6 codes under a wide range of workloads.

4.1 Evaluation Methodology

We configure an typical disk array size for RAID-6 codes, with the parameter of prime P is set to 5, 7, and 11, respectively. We adopt a classical trace to mimic I/O behaviors of storage systems. The size of most write requests is about 4KB in Microsoft Exchange Server [19]; the rate of ~4KB write requests is 75.6% over 15-minute intervals trace. Meanwhile, the size of one typical stripe is 256KB in production storage systems [7]; the size of a data block is 4KB; and single writes are dominant among various access patterns. Therefore, it is essential and important to examine the impact of single-write patterns on the performance of disk arrays. We evaluate the load balancing of a handful of RAID-6 codes by selecting several representative types of read/write requests as follows.

- **Read(R)**: a read operation occurring on data blocks.
- **Single Write(SW)**: a write operations occurring on a data block once, i.e., a single-data-block-write request. Simultaneous write operations do not happen on continuous data blocks in the case of single writes.
- **Equational Read-Write(ERW)**: both read and write requests evenly occur on data blocks. In particular, the case of 50% read requests and 50% single-write requests on data blocks is referred to as **ERSW**.

The above three typical access patterns are observed in real-world storage systems. The three access patterns are in part reflected by read-intensive workloads, write-intensive workloads and read-write mixed workloads. To reveal I/O distribution of the various RAID-6 codes under a wide range of workloads, we define an ideal access sequence in a stripe (i.e., each data block is treated as a read/write block only once as for a stripe). To further characterize the diversified access patterns of a wide spectrum of traces, we investigate read/write requests by taking both uniform and random I/O access patterns into accounts.

- **Uniform Access(UA)**: each request to a data block occurs only once. Specifically, **UA-R** refers to a read UA request, **UA-SW** denotes a single-write UA request, and **UA-ERSW**

means that the access frequency of each data block is only one in the **ERSW** case.

- **Random Access(RA)**: the access frequency of read/write requests to each data block occurs randomly between 1 to 1000. Specifically, **RA-R** represents a read RA request, **RA-SW** refers to a write RA request, **RA-ERSW** means that the access frequency of each data block ranges randomly from 1 to 1000 in the **ERSW** case.

The load-balancing ratio (see also the definition in [38]) is adopted to evaluate the load-balancing degree of erasure-coded storage systems. Similarly, we define the load-balancing ratio (**L**) as a rate between the I/O access numbers (i.e., Num_{max}) of a heaviest-load disk and the I/O access numbers (i.e., Num_{min}) of a lightest-load disk in a disk array. Eq. (3) specifies the expression of load-balancing ratio L.

$$L = \frac{Num_{max}}{Num_{min}} \tag{3}$$

A small value of load-balancing ratio (**L**) suggests that an erasure-coded storage system exhibits prominent I/O-balancing feature. The lower bound of load-balancing ratio **L** is 1 (i.e., N_{max}=N_{min}), indicating that the erasure-coded storage system gains the best load balancing.

4.2 Load Balancing Comparison

The I/O-balancing metric of N-Code and the five popular codes (i.e., P-Code, Short code, H-Code, RDP code and EVENODD code) is assessed under various access patterns. The evaluation results of load-balancing ratio are depicted in Figs. 5 and 6.

Figs. 5 and 6 show that I/O balancing ratio of N-Code is overwhelmingly lower than those of the other five popular codes under a wide spectrum of workloads with various access patterns, with parameter P is set to 5, 7, and 11, respectively.

- In the case of read-only workload (i.e., UA-R and RA-R), N-Code and P-Code achieve good load balancing, whereas other four popular codes (i.e., Short code, H-Code, RDP code and EVENODD code) exhibit bad load balancing.
- For the single-write workload (i.e., UA-SW and RA-SW), N-Code enhances load balancing between 1.4x to 6.92x for UA-SW workload and between 1.19x to 9.09x for RA-SW workload relative to the five popular codes.
- As to the read-write mixed workload (i.e., UA-ERSW and RA-ERSW), the I/O balancing ratio in N-Code improves load balancing between 1.27x to 5.4x for UA-ERSW workload and between 1.3x to 6.92x for RA-ERSW workload.

In summary, N-Code achieves better load balancing than the five competitive codes. The essential rationale is that N-Code assigns the parity elements across all the disks, whereas the popular codes (e.g., Short code, H-Code, RDP code and EVENODD code) distribute parity elements to the dedicated disk, thereby leading to unbalanced I/O layout across disks. Furthermore, EVENODD and RDP codes are non-lowest density codes, which bring forth high updating I/O cost. Both unbalanced I/O load and high I/O cost would degrade access performance of erasure-Coded storage systems.

5 READ/WRITE PERFORMANCE EVALUATION

5.1 Experimental Platform

In this section, we conduct a group of experiments to evaluate both read and write performance for the various RAID-6 codes in a real-world RAID system (i.e., a DELL PowerEdge-R920 server). The server is equipped with a CPU of type Intel(R) Xeon(R) E7-4809 v2 @ 1.90GHz, 24 SAS disks, and the main memory size is 24GB. All the SAS disks are organized into a disk array. The disks are Seagate AL15SEB030NY. Each disk has 300GB capacity and 10K RPM. The operation system of the server is Ubuntu with Linux kernel 14.04.

5.2 Evaluation Methodology

We compare our N-Code with the seven popular RAID-6 codes (i.e., EVENODD, RDP, H-Code, HDP, Short, D-Code, and X-Code. We evaluate read/write performance for the RAID-6 codes, where the number P of disks is set to 5, 7, and 11, respectively. Furthermore, all the experimental results are normalized to clarify the gap between N-Code and the alternative codes, with the performance value of the RDP code is normalized to 1.

Read performance is evaluated in terms of normal read speed and degraded read speed. Write performance is evaluated via partial stripe write cost and partial stripe write speed. We run five experiments for each test configuration, in which we randomly generate the start point and read/write size in each stripe. The start point is an arbitrary data element in a stripe, and the read/write size is varied from 1 to the number of data elements in a half stripe. For example, as to 8-disks RDP code, the read size is varied from 1 to 18 (see Fig. 1).

5.3 Read Performance Experiment

5.3.1 Normal Read Speed. We evaluate the normal read performance of all the RAID-6 codes, the normal-read-speed performances of which are ploted in Fig. 7. The P value of the RAID-6 codes is set to 5, 7, and 11, respectively.

Fig. 7 reveals that N-Code, X-Code and D-Code outperform the other popular RAID-6 codes in terms of normal read speed. In particular, N-Code's normal read speed is 6% to 15% higher than those of the horizontal codes EVENODD and RDP; N-Code enhances the normal read speed of H-Code and Short Codes by up to 9% and 7%, respectively. The driving force behind N-Code's superb performance is that all the disks in N-Code respond to normal reads; in other words, one more accessed disk improves parallel reads in the normal mode. On the contrary, the dedicated parity disks in the competitors (e.g., H-Code, Short, RDP and EVENODD) can not facilitate normal reads with sufficient parallel I/O accesses, thereby deteriorating read performance.

5.3.2 Degraded Read Speed. We evaluate the degraded read performance of the RAID-6 codes in the case of single-disk failure. For each RAID-6 code, there are n single-disk-failure cases, where n denotes the number of data disks, and each single-disk failure case is viewed as one individual test. In this way, degraded read speed is the average value of degraded read speed under all the single-disk failure cases. Fig. 8 plots degraded read speeds of various RAID-6 codes.

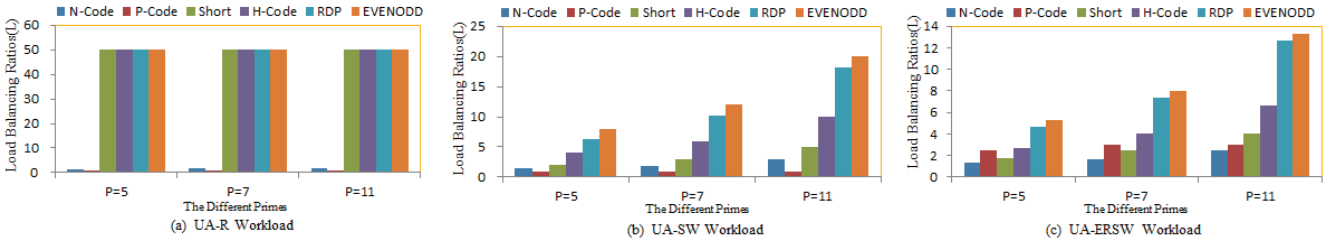


Fig. 5: The load-balancing ratios under different workloads with the UA pattern. Due to the limitation of the range in y-axis, we use '50' to represent the infinity

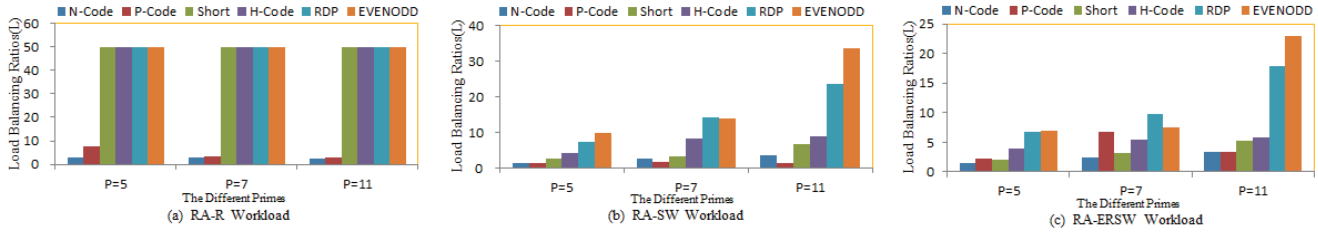


Fig. 6: The load balancing ratios under different workloads with the RA Pattern. Due to the limitation of the range in y-axis, we use '50' to represent the infinity

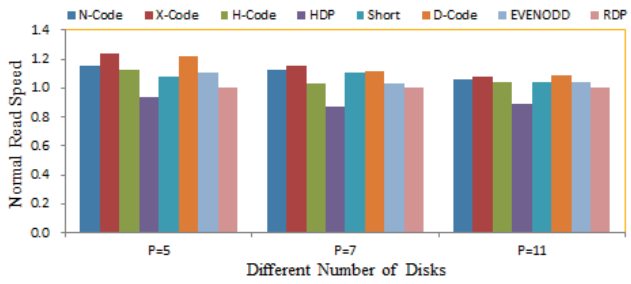


Fig. 7: Normalized normal-read speeds

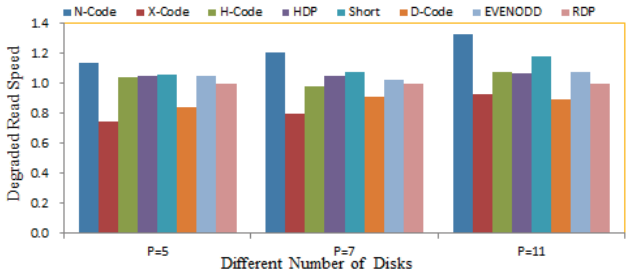


Fig. 8: Normalized degraded-read speeds

Fig. 8 unveils that N-Code provides a higher degraded-read speed than the seven baseline codes. For example, compared to the vertical codes (e.g., X-Code and D-Code), N-Code improves the degraded-read speed by 32% to 53% when parameter P is configured to 5, 7, and 11. The reason for such a productive improvement is that it is difficult for continuous data elements to share common parity chains in vertical codes and; thus, additional I/O cost is incurred by degraded reads. Compared to the horizontal codes (e.g., EVENODD and RDP), N-Code demonstrates a higher degraded read speed by 14% to 33%, because that one more accessed disk helps to boost the degraded read performance, whereas diagonal parity disks in the horizontal codes don't service degraded reads.

In a nutshell, after the comparative evaluations with respect to normal and degraded read speeds, we confirm that N-Code delivers better read performance than the other popular RAID-6 codes.

5.4 Write Performance Experiment

We evaluate the write performance of N-Code and all the baseline RAID-6 codes using partial stripe write cost and partial stripe write speed as the metrics.

5.4.1 Partial Stripe Write Cost. We calculate partial stripe write cost of each test item via the ratio between the total I/O access numbers (onto both data elements and corresponding parity elements that needs to be written/read) and the numbers of data elements (that needs to be written). For instance, Fig. 2(d) shows that in order to handle a 7-elements write, a total of 42 I/Os on 21 elements (i.e., 7 data elements and 14 corresponding parity elements) need to be accessed, thus the partial stripe write cost is $42/7 = 6$. The partial stripe write costs for all the evaluated RAID-6 codes are plotted in Fig. 9.

Fig. 9 reveals that N-Code achieves a lower partial-stripe-write cost than those of the baseline codes. For instance, compared to the vertical codes (e.g., X-Code and D-Code), N-Code saves partial stripe write cost by anywhere between 15.8% to 28.1%. Because continuous data elements in the vertical codes can not share the common parity chains, which pose high I/O cost for parity-element updating incurred by partial stripe writes. In addition, the horizontal codes (e.g., EVENODD, RDP, Short and H-Code) achieves lower partial-stripe-write cost than those of the vertical codes. For instance, H-Code provides 17.4% to 25.4% lower partial-stripe-write

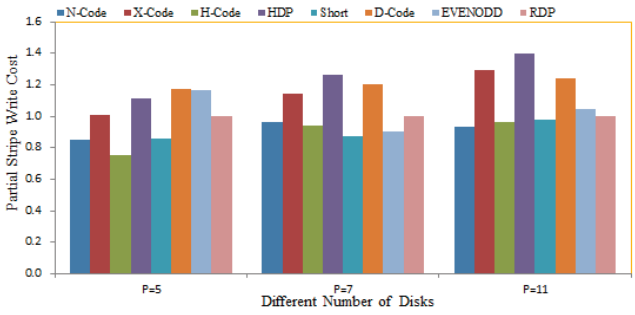


Fig. 9: Normalized partial-stripe-write costs

I/O cost than X-Code. The comparative results are reasonable, because horizontal parity chains help to accomplish I/O merging, which reduces I/O access cost. However, high write cost may accelerate the burn-in of such crisp devices like SSDs.

5.4.2 Partial Stripe Write Speed. Fig. 10 shows partial-stripe-write speed of the tested RAID-6 codes in the real storage system. Because the write performance is limited by the disk access speed rather than write cost, the results in Fig. 9 are inconsistent with those plotted in Fig. 10.

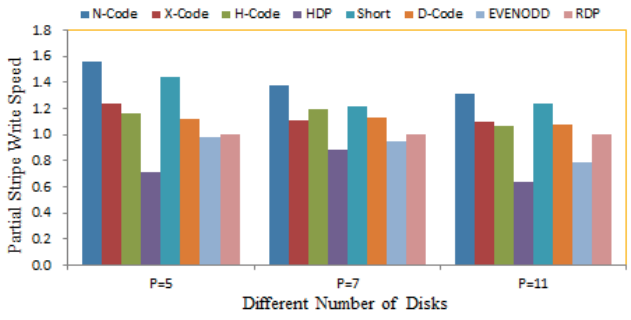


Fig. 10: Normalized partial-stripe-write speeds

Fig. 10 unveils that N-Code achieves a salient write-performance improvement against the existing RAID-6 codes. For instance, N-Code accelerates the partial-stripe-write speed of the vertical codes X-Code and D-Code by 38% and 20%, because there are only diagonal parity chains in the vertical codes leading to high I/O cost under partial stripe writes. The partial-stripe-write speed of N-Code is 32% and 66% higher than those of horizontal codes EVENODD and RDP, since both EVENODD and RDP are categorized as non-lowest density codes, which cause the heaviest load on the dedicated diagonal parity disk. N-Code offers 34% and 6% higher partial-stripe-write speed than those of horizontal-diagonal mixed codes Short and H-Code; the reason is that there are dedicated diagonal parity disks in horizontal-diagonal mixed codes (see the details in Figs. 5 and 6), which induce unbalanced load and reduce write speed when partial stripe writes occur.

The comparative evaluations on partial-stripe-write cost and partial-stripe-write speed conclude that N-Code is adroit at offering higher partial-stripe-write performance than the conventional RAID-6 codes.

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed a novel RAID-6 MDS array code, which is termed as N-Code. A stripe of N-Code consists of (p-1)×(p+1) elements evenly distributed on (p+1) logic disks, where p is a prime number. Different from the previous RAID-6 codes, N-Code assigns horizontal parity elements across disks rather than relying on an independent disk, and diagonal parity elements in N-Code are uniformly distributed between the first and last disks.

Apart from the optimized storage efficiency, the mitigated encoding/decoding computational complexity, and the alleviated update complexity, N-Code delivers superb read performance (i.e., normal and degraded reads) and write performance (i.e., single and partial stripe writes). N-Code exhibits improved performance in terms of I/O balancing compared to the other popular RAID-6 codes. The evaluation results show that N-Code achieves good performance on I/O balancing, normal reads, degraded reads, single writes, and partial stripe writes. The evidence suggests that N-Code is a sturdy candidate for implementing RAID-6 storage systems.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their insightful comments. This work is supported by the National Nature Science Foundation of China(NSFC) under Grant No. 61762075 and 61862055, Next Generation Internet Technology Innovation Project of CERNET under Grant No. NGII20180116. Meanwhile, it is also supported in part by Qinghai Key Laboratory of Internet of Things Project under Grant No. 2017-ZJ-Y21. Jianzhong Huang's work is supported by the National Science Foundation of China under Grants No. 61572209 and Creative Research Group Project of NSFC No.61821003. Xiao Qin's work is supported by the 111 Project under Grant B07038.

REFERENCES

[1] Mario Blaum, Jim Brady, Jehoshua Bruck, and Jai Menon. 1995. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. *Computers, IEEE Transactions on* 44, 2 (1995), 192–202.
[2] Mario Blaum and Ron M Roth. 1999. On lowest density MDS codes. *Information Theory IEEE Transactions on* 45, 1 (1999), 46–59.
[3] Mario Blaum and Ron M. Roth. 1999. On lowest density MDS codes. *Information Theory, IEEE Transactions on* 45, 1 (1999), 46–59.
[4] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam Mckelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, and Huseyin Simitci. 2011. Windows Azure Storage:a highly available cloud storage service with strong consistency. In *ACM Symposium on Operating Systems Principles*. 143–157.
[5] Yuval Cassuto and Jehoshua Bruck. 2009. Cyclic lowest density MDS array codes. *Information Theory, IEEE Transactions on* 55, 4 (2009), 1721–1729.
[6] Peter Corbett, Bob English, Atul Goel, Tomislav Grcanac, Steven Kleiman, James Leong, and Sunitha Sankar. 2004. Row-diagonal parity for double disk failure correction. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*. 1–14.
[7] Corporation EMC. July 2007. EMC CLARiiON RAID 6 Technology: A Detailed Review. White Paper. http://tahoe-lafs.org/trac/tahoe-http://www.emc.com/collateral/hardware/white-papers/h2891-clariion-raid-6.pdf
[8] Bin Fan, Wittawat Tantisiriroj, Lin Xiao, and Garth Gibson. 2009. DiskReduce:RAID for data-intensive scalable computing. In *The Workshop on Petascale Data Storage*. 6–10.

[9] Yingxun Fu and Jiwu Shu. 2015. D-Code: An Efficient RAID-6 Code to Optimize I/O Loads and Read Performance. In *Parallel and Distributed Processing Symposium*. 603–612.

[10] Yingxun Fu, Jiwu Shu, Xianghong Luo, Zhirong Shen, and Qingda Hu. 2017. Short Code: An Efficient RAID-6 MDS Code for Optimizing Degraded Reads and Partial Stripe Writes. *IEEE Trans. Comput.* 66, 1 (2017), 127–137.

[11] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In *ACM SIGOPS Operating Systems Review*, Vol. 37. ACM, 29–43.

[12] Kevin M Greenan, Xiaozhou Li, and Jay J Wylie. 2010. Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 1–14.

[13] James Lee Hafner. 2005. Weaver codes: Highly fault tolerant erasure codes for storage systems. In *Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies*, Vol. 4. 16–16.

[14] James Lee hafner. 2006. HoVer erasure codes for disk arrays. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*. IEEE, 217–226.

[15] Yuchong Hu, Henry C. H. Chen, Patrick P. C. Lee, and Yang Tang. 2012. NCCloud: applying network coding for the storage repair in a cloud-of-clouds. (2012), 21–21.

[16] Cheng Huang, Minghua Chen, and Jin Li. 2007. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. In *Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on*. IEEE, 79–86.

[17] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. 2012. Erasure coding in windows azure storage. In *Usenix Conference on Technical Conference*. 2–2.

[18] Chao Jin, Hong Jiang, Dan Feng, and Lei Tian. 2009. P-Code: A new RAID-6 code with optimal properties. In *Proceedings of the 23rd international conference on Supercomputing*. ACM, 360–369.

[19] Swaroop Kavalanekar, Bruce Worthington, Qi Zhang, and Vishal Sharda. 2008. Characterization of storage workload traces from production windows servers. In *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*. IEEE, 119–128.

[20] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, and Chris Wells. 2002. *OceanStore: an architecture for global-scale persistent storage*. ACM. 190–201 pages.

[21] Mingqiang Li and Patrick P. C. Lee. 2014. STAIR codes: a general family of erasure codes for tolerating device and sector failures in practical storage systems. In *Usenix Conference on File and Storage Technologies*. 147–162.

[22] Mingqiang Li and Jiwu Shu. 2011. C-Codes: Cyclic Lowest-Density MDS Array Codes Constructed Using Starters for RAID 6. *Ibm Corporation* (2011).

[23] Michael Luby and David Zuckermank. 1995. *An XOR-Based Erasure-Resilient Coding Scheme*. Technical Report. Tech Report.

[24] F. J Macwilliams and N. J. A Sloane. 1977. *The theory of error-correcting codes*. North-Holland Pub. Co. 185–186 pages.

[25] David A Patterson, Garth Gibson, and Randy H Katz. 1988. *A case for redundant arrays of inexpensive disks (RAID)*. Vol. 17. ACM.

[26] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso. 2007. Failure trends in a large disk drive population. In *Proceedings of the 5th USENIX conference on File and Storage Technologies*. 2–2.

[27] James S plank. 2007. A new MDS erasure code for RAID-6. In *Technical Report CS-07-602, University of Tennessee*.

[28] James S Plank. 2008. A new minimum density RAID-6 code with a word size of eight. In *Network Computing and Applications, 2008. NCA'08. Seventh IEEE International Symposium on*. IEEE, 85–92.

[29] James S. Plank. 2008. The RAID-6 Liberation Codes. In *6th USENIX Conference on File and Storage Technologies, FAST 2008, February 26-29, 2008, San Jose, CA, USA*. 97–110. http://www.usenix.org/events/fast08/tech/plank.html

[30] James S. Plank, Mario Blaum, and James L. Hafner. 2013. SD codes: erasure codes designed for how storage systems really fail. In *Usenix Conference on File and Storage Technologies*. 95–104.

[31] Irving S Reed and Gustave Solomon. 1960. Polynomial codes over certain finite fields. *Journal of the Society for Industrial & Applied Mathematics* 8, 2 (1960), 300–304.

[32] Peter Scheuermann, Gerhard Weikum, and Peter Zabback. 1993. Adaptive load balancing in disk arrays. In *Foundations of Data Organization and Algorithms*. Springer, 345–360.

[33] Peter scheuermann, Gerhard Weikum, and Peter Zabback. 1998. Data partitioning and load balancing in parallel disk systems. *the VLDB Journal* 7, 1 (1998), 48–66.

[34] Bianca Schroeder and Garth A Gibson. 2007. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you. In *Proceedings of the 5th USENIX conference on File and Storage Technologies*, Vol. 6. 2.

[35] Zhirong Shen and Jiwu Shu. 2014. HV Code: An All-Around MDS Code to Improve Efficiency and Reliability of RAID-6 Systems. In *Ieee/ifip International Conference on Dependable Systems and Networks*. 550–561.

[36] Itzhak Tamo, Zhiying Wang, and Jehoshua Bruck. 2011. Zigzag Codes: MDS Array Codes With Optimal Rebuilding. *IEEE Transactions on Information Theory* 59, 3 (2011), 1597–1616.

[37] Shenggang Wan, Qiang Cao, Changsheng Xie, Ben Eckart, and Xubin He. 2010. Code-M: A non-mds erasure code scheme to support fast recovery from up to two-disk failures in storage systems. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*. IEEE, 51–60.

[38] Chentao Wu, Xubin He, Guanying Wu, Shenggang Wan, Xiaohua Liu, Qiang Cao, and Changsheng Xie. 2011. HDP code: A Horizontal-Diagonal Parity Code to Optimize I/O load balancing in RAID-6. In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*. IEEE, 209–220.

[39] Chentao Wu, Shenggang Wan, Xubin He, and Qiang Cao. 2011. H-Code: A Hybrid MDS Array Code to Optimize Partial Stripe Writes in RAID-6. In *Parallel & Distributed Processing Symposium*. 782–793.

[40] Ping Xie, Jianzhong Huang, Qiang Cao, Xiao Qin, and Changsheng Xie. 2013. V2-Code: A new non-MDS array code with optimal reconstruction performance for RAID-6. In *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*. IEEE, 1–8.

[41] Ping Xie, Jianzhong Huang, Qiang Cao, and Changsheng Xie. 2014. Balanced P-Code: A RAID-6 Code to Support Highly Balanced I/Os for Disk Arrays. In *IEEE International Conference on Networking, Architecture, and Storage*. 133–137.

[42] Qin Xin, Ethan L Miller, Thomas Schwarz, Darrell DE Long, Scott A Brandt, and Witold Litwin. 2003. Reliability mechanisms for very large storage systems. In *Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on*. IEEE, 146–156.

[43] Lihao Xu, Vasken Bohossian, Jehoshua Bruck, and David G Wagner. 1999. Low-density MDS codes and factors of complete graphs. *Information Theory, IEEE Transactions on* 45, 6 (1999), 1817–1826.

[44] Lihao Xu and Jehoshua Bruck. 1999. X-code: MDS array codes with optimal encoding. *Information Theory, IEEE Transactions on* 45, 1 (1999), 272–276.

[45] GV Zaitsev, Victor Alexandrovich Zinov'ev, and NV Semakov. 1983. Minimum-check-density codes for correcting bytes of errors, erasures, or defects. *Problemy Peredachi Informatsii* 19, 3 (1983), 29–37.