Tier-code: An XOR-based RAID-6 Code with Improved Write and Degraded-mode Read Performance

Bingzhe Li[†], Meng Yang[*], Soheil Mohajer[†], Weikang Qian[*], David J. Lilja[†]
[†]Department of Electrical and Computer Engineering, University of Minnesota, U.S.A.
[*]University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, China
Email: [†]{lixx1743, soheil, lilja}@umn.edu, [*]{yangm.meng, qianwk}@sjtu.edu.cn

***Abstract*—The RAID-6 configuration is more tolerant of disk failures than other RAID levels because of its ability to tolerate two disk failures. However, previous RAID-6 codes suffer from two major overheads - the time of encoding or decoding processes plus the need to access multiple blocks when updating parities or recovering failed blocks. For example, the PS and Reed-Solomon codes do not have optimal computation complexity, while P-code, X-code and RDP-code must access multiple blocks to update parities during write operations. This work proposes a new XOR-based RAID-6 code, called Tier-code, which not only achieves the optimal parity computation complexity, but also increases the write and degraded-mode read performance compared to previous codes. It uses two tiers of coding, one at the block level and the other at the chunk level. Experimental results of software testing, simulation and ASIC synthesis for this new hierarchical code demonstrate that Tier-code can outperform the previous RAID-6 codes in both write performance and degraded-mode read performance while maintaining the optimal computation complexity in both hardware and software implementations.**

I. Introduction

The new storage systems often have to deal with multiple disk failures. With steadily increasing the storage capacity of disks [1][2], disk failures have remained a significant concern [3]. Therefore, as demand for storage continues to grow, large storage systems often have to deal with multiple simultaneous disk failures. In order to tolerate failures, people use the Redundant Array of Independent Disks [4] (RAID) configuration to build fast and reliable disk systems. However, by evaluating the degrade-mode RAID systems [5][6], the performance of those RAID system are degraded significantly and thus the performance of those RAID systems becomes a critical issue with increasing the requirement of the high-speed systems.

Previously, the most popular RAID-6 uses the Reed-Solomon code [7]. However, it encounters very high overhead from the parity computation. Later, an new code called Cauchy Reed-Solomon (CRS) [8] improved the parity computation by converting multiplication to XOR operations. However, there still has been significant opportunity to reach the optimal computation complexity that has the minimum number of XORs for parity updates. Regarding the reduction of the overhead from the parity computation, several previous codes achieve the optimal computation complexity including X-code [9], P-code [10], B-code [11], EVENODD code [12], and RDP-code [13]. All of these codes use the minimum number of XORs in their encoding processes. However, those codes need to update more parities to achieve the optimal computation complexity because they use scattered layouts.

In addition, a degraded mode read operation can be regarded as the inverse operation of a write operation, whose performance is related to the decoding complexity and the number of read operations for recovering failed data blocks. Similar to the write operation, none of the previous codes can keep both the optimal decoding computation complexity and the minimum number of extra reads. Therefore, there is an opportunity to improve the write and degraded mode read performance while maintaining the optimal computation complexity and the minimum number of operations.

In this paper, we propose a new XOR-based RAID-6 code called Tier-code, which has optimal computation complexity and the minimum number of operations compared to the previous codes. The main characteristic of this code is to use two tiers of labeling, block-level labeling and chunk-level labeling, to compute parities. By using tier-labeling, the code produces better write performance and degraded mode read performance than previous codes while still maintaining fault-free read performance [9][10][14][13][15][16][17]. To evaluate the write and degraded mode read performance, we compare those codes from two aspects, parity computation complexity and I/O performance. Real system testing and VLSI implementation are employed to indicate the parity computation complexity. The DiskSim simulator is used to demonstrate the I/O performance of systems implemented with different codes.

This paper is organized as follows: Section II discusses the background and related work. Section III introduces Tier-code's construction and reconstruction algorithms. The write and degraded mode read performance analysis is discussed in Section IV. Section V presents the experimental methodology and results. Finally, the conclusion is presented in Section VI.

II. Background and Related Work

A. *Terms and Notations*

To give a better understanding of the whole paper, we first summarize the terms and notations that will be used frequently throughout this paper.

978-1-5386-8367-5/18/$31.00 ©2018 IEEE

- **Update Complexity [18]** is the number of parity blocks that need to be accessed when one data block is updated.
- **Computational Complexity [18]** corresponds to the operations that calculate the parities in a RAID systems. If a RAID code uses minimum number of XORs to compute parities, it is regarded as having the optimal computational complexity. Compared to XOR operations, the multiplication in the Galois field (GF) is much more expensive.
- **Storage Efficiency** is the ratio between the total useful data size and the raw size. In RAID-6 systems, the optimal storage efficiency is (M-2)/M, where M is the number of disks.
- **Multi-block Access Complexity [15]** indicates the average number of parity updates and read operations when a request accesses multiple data blocks.
- **Degraded Mode Read.** A RAID system enters degraded mode when a failure occurs in one or more disks. If one request wants to read data blocks including failed blocks, the system not only needs to read non-failed data blocks, but also needs to reconstruct the failed data blocks by reading other uncorrupted data and parity blocks. Thus, the system can provide the host with the requested data blocks correctly even with failed disks. However, the performance of degraded mode reads may be much lower than the performance of normal read operations due to the time required to access the additional blocks and compute the desired block.
- **Degraded Read Complexity** is defined in terms of the average number of extra read operations for recovering the failed data blocks in the RAID system during degraded mode.
- **Data Block Labeling and Parity Block Labeling.** The labeling shows the dependency of parity and data blocks. Parity is computed from those data blocks that share the same label as the parity's label. As seen in Figure 1, the two parity blocks labeled "1" are computed from the four data blocks labeled "1". **Block-level labeling** and **chunk-level labeling** are two-tier labellings. Each block consists of several chunks.
- A **Stripe** is a set of data and parity blocks that have the same label value.
- **Read-modify-write [19].** If a write request does not cover a whole stripe array, RAID systems cannot simply write the small portion of data to disks. The systems first must read old data blocks in this stripe, then obtain new data from the host, and then compute new parities using those data. After computing the parities, it can write the new data to target locations and update the new parities.

B. Background and Related Work

Erasure coding is a popular technique which has been used in many fields such as network and storage systems. Recently, new erasure codes are designed for storage systems to tolerate up to two disk failures. However, those erasure codes designed for RAID-6 storage systems suffered parity write and degraded read overhead. Some of RAID-6 codes are introduced in the following:

Reed-Solomon code [7] is a popular erasure coding technique using Galois Field arithmetic ($GF(2^w)$) (where w is the number of columns for the encoding matrix) during coding and decoding. However, the computation overhead of Galois Field arithmetic is very expensive.
PS-code [15] is a vertical code that attains optimal multi-block access complexity based on its labeling algorithm. It improves write performance when the write operations access multiple contiguous blocks. However, even though it uses the Cauchy Reed-Solomon code to compute its parities, PS-code still suffers low performance on the parity computation.
RDP-code [13] is constructed using $prime + 1$ columns and $prime - 1$ rows, which is similar to EVENODD code in terms of using the diagonal data blocks to calculate the second parities. The difference lies in the RDP-code's ability to obtain the optimal computational complexity because it directly XORs all diagonal data blocks instead of introducing an extra intermediate parameter as used in EVENODD code.
X-code [9] has the $p * p$ structure, where p is a prime number. Data blocks are stored in the first $p - 2$ rows and parities are stored at last two rows. It uses two diagonals with slope 1 and -1 to compute the first and second row parity. X-code has the optimal update complexity and computational complexity.
P-code [10] is constructed with a $(p-1)/2 * (p-1)$ matrix. The parity blocks are located at the first row and the data blocks are located at the remaining $(p-3)/2$ rows. A pair of tuple values (m, n) and integer i are assigned to each data block and parity, respectively. By following the labeling rules in P-code, it can achieve the optimal computational complexity and update complexity.
HV-code [16] is a type of hybrid vertical and horizontal code that has optimal computational and update complexity. By using the horizontal chain to compute first parties and the diagonal chain to compute second parties, it improves I/O balancing and optimizes the operation of partial stripe writes to contiguous data elements.

The above codes share a common problem: they are not able to achieve optimal computation complexity, low multi-block access complexity, and low degraded read complexity at the same time. As shown in the following, our new Tier-code achieves all of these goals.

III. Tier-code

A. Tier-code Description

In RAID-6 systems, for each data block write, at least two parity block updates have to take place [9][10][20][12][18][16][21]. For such case, the update complexity [18] indicates the number of parities when one data block is updated. Additionally, in real systems, multiple blocks are updated together for each write I/O request. This scenario is investigated by the multi-block access complexity [15], which demonstrates one request that requires multiple block writes. Therefore, two above complexities can reflect the overhead for write requests in RAID systems.

Tier-code is a RAID-6 code constructed using $M = p + 1$ disks, where p is prime number. The main purpose of Tier-code is to improve write and degraded mode read performance.

To achieve the best update complexity and multi-block access complexity, Tier-code uses the contiguous block-level labeling layout. Moreover, a revised code labeling algorithm is applied into Tier-code's chunk-level labeling so that Tier-code has the optimal computation complexity by using the minimum number of XORs to compute the parities. The chunk-level labeling may use the codes such as RDP-code, P-code and other and then most of the array codes, especially for the strong systematic ones, will have similar performance. In this paper, we use revised RDP-code as an example for chunk-level labeling. Compared to XORs, some previous works like RS-code using multiplication over GF to compute parities faces significant overhead on the parity computation time. Therefore, by using the two-tiered labeling algorithms, Tier-code can achieve better write and degraded mode read performance. In addition, the storage efficiency of Tier-code is the same as other codes, which is $(M-2)/M$. The labeling and encoding algorithms are described in the following paragraphs.

Tier-code's encoding algorithm can be divided into three primary steps – block-level labeling, chunk-level labeling and construction. Each block consists of $(M-2)$ chunks. Figure 1 and Figure 2 give two examples of block-level labeling and chunk-level labeling.

1) Block-level Labeling: In the block-level labeling, we assume that the block matrix has the size of $M*M/2$ including both data blocks and parity blocks as seen in Figure 1. i $(0 < i \leq M/2)$ and j $(0 < j \leq M)$ are the row number and the disk number, respectively. D_{ij} is the value of the data block located at the i^{th} row and the j^{th} disk, and C_{ij} is its corresponding data block's label value. First, we assign integer labels to each data block where the label C_{ij} can be obtained from Eq. 1.

$$C_{ij} = \lfloor \frac{i*M+j-3}{M-2} \rfloor \tag{1}$$

To compute the parity in the block level, the parity row is labeled using Eq. 2. P_j is the parity's label value $(0 < P_j \leq M/2)$, j $(0 < j \leq M)$ is the disk number (or column index).

$$P_j = \lfloor \frac{M-j+2}{2} \rfloor \tag{2}$$

As shown in Figure 1, with an example using $M = 6$ disks, label values of the data and parity blocks are distributed across all disks, which are computed from Eq. 2.

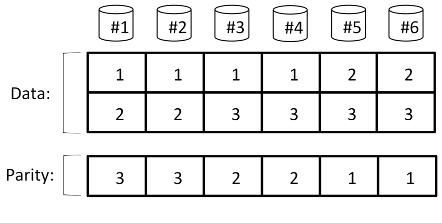


Fig. 1. An example of the Tier-code with number of disks M=6. The labels of the parity row are calculated in Eq. 2. As an illustration, the two parity blocks P_5 and P_6 labeled "1" are computed from four data blocks labeled "1".

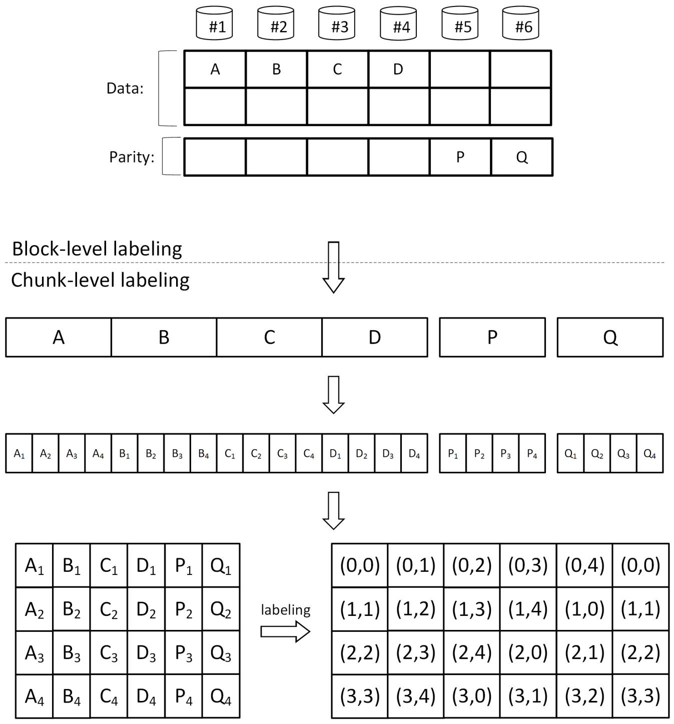


Fig. 2. An example of Tier-code with the number of disks $M = 6$. Suppose the blocks 'A', 'B', 'C', 'D', 'P' and 'Q' are from the same stripe with the label value '1'. The first four blocks are data blocks. The blocks 'P' and 'Q' are parities. The labeling tuples are assigned to the blocks. The first element of the tuple is used for the parity block 'P' computation. The second element of the tuple is used for the parity block 'Q' computation. The other block-level stripes need to follow the same process of the tuple labeling to compute the parity values.

2) Chunk-level labeling: After the block-level labeling, each block-level stripe needs to be addressed by the chunk-level labeling. First, we focus on one block-level stripe because all block-level stripes share the same chunk-level labeling algorithm.

The chunk-level labeling has two steps. The first step of the chunk-level labeling is to split each block of the selected block-level stripe into $(M-2)$ chunks. Then, the second step is to assign a label tuple to the $(M-2)$ chunks. The labeling algorithm is shown in Eq. 3.

$$T_{ij} = (i, j\%(M-1)+i)) \tag{3}$$

where, T_{ij} is the labeling tuple of the i^{th} chunk of the j^{th} block and M is the number of disks. The first element of T_{ij} is the labeling for computing parity P $(i, j \geq 0)$. The second element is the labeling for computing parity Q.

Figure 2 shows a tuple labeling example with $M = 6$. The blocks from the same block-level stripe with the label value '1' in Figure 1 are split into smaller size of chunks and then they are given chunk-level tuple labels. Finally, the rest of the block-level stripes in one matrix need to be labeled using the chunk-level labeling by following the above steps.

3) Construction: After the chunk labeling process, the algorithm starts to compute parities. For the parity block P, each chunk of the parity P is computed by XORing all the data chunks whose first values of the tuple are equal to the first element of the parity P's chunk (We call the stripe as P stripe). After computing the parity Ps of all chunks, the parity P computation is done for this chunk-level stripe. The parity Q that are computed from second elements of tuples follows a similar process (We call the stripe as Q stripe). Two computation examples for the parities P and Q are given in Eq. 4 and Eq. 5 based on the layout shown in Figure 2.

$$\begin{cases} P_1 = A_1 \oplus B_1 \oplus C_1 \oplus D_1 \\ P_2 = A_2 \oplus B_2 \oplus C_2 \oplus D_2 \\ P_3 = A_3 \oplus B_3 \oplus C_3 \oplus D_3 \\ P_4 = A_4 \oplus B_4 \oplus C_4 \oplus D_4 \end{cases} \tag{4}$$

$$\begin{cases} Q_1 = A_1 \oplus C_4 \oplus D_3 \oplus P_2 \\ Q_2 = A_2 \oplus B_1 \oplus D_4 \oplus P_3 \\ Q_3 = A_3 \oplus B_2 \oplus C_1 \oplus P_4 \\ Q_4 = A_4 \oplus B_3 \oplus C_2 \oplus D_1 \end{cases} \tag{5}$$

The construction for the selected block-level stripe is done after finishing computation of the parities. Then, another block-level stripe is selected for its construction following exactly the same process above until all of the block-level stripes have been constructed.

B. Reconstruction of Tier-code

In Figure 1, we can see that, in the block-level labeling, the parity P and parity Q are computed from the same data blocks because both of them share the same labeling algorithm. However, in the chunk-level labeling, the parity P and parity Q are calculated from different chunks based on the first element and second element of the chunk-level labeling tuples, respectively.

In the block-level labeling, each block label value is unique in one disk. Also, each block label value is distributed across all the disks. Consequently, every stripe has M blocks covers M disks. In other words, each disk contains only one block from each stripe. Moreover, each stripe has the same construction process. Therefore, we only need to consider one block-level stripe reconstruction process and the rest of the stripes use exactly same process for the reconstruction process.

For a block-level stripe, according to the chunk labeling algorithm shown in Figure 2, the first elements of the chunks' tuples are the same for the chunks located in the same row. For the second element of tuples, each block contains $M-2$ chunks with $M-2$ different values. Additionally, any two blocks that keep $M-2$ chunks with $M-2$ different second labeling values have at least two different second labeling values. In other words, there are two different second labeling values in two blocks. The two unique second labeling values are the starting points for recovering failed blocks. An example of two failed blocks is given in Figure 3. Block B contains the second labeling '1', '2', '3' and '4' in its chunks' tuples. Block C contains the second labeling '0', '2', '3' and '4' in its chunks' tuples. Therefore, the labeling '0' and '1' are two unique values in the block B and C labellings. So, the two chunks holding the unique second values are the starting points of the reconstruction process. As seen in Figure 3, the two chunks are marked with ① as the first reconstruction step.

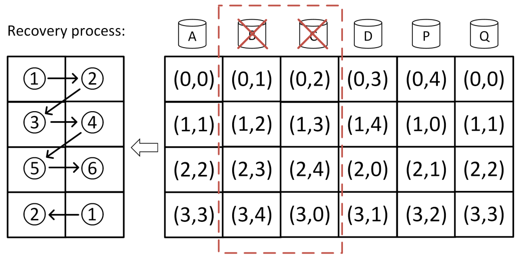


Fig. 3. An example with block B and C failed for $M = 6$. The left part of the figure shows the recovery steps following the ordinal number. The first step is to recover the left top chunk and right bottom chunk because both chunks are the only failed chunks in their Q stripes. Once they are recovered, the second step is to recover the failed chunks from their P stripe. Then, the next step is to recover the Q stripe again and then P stripe until all chunks are recovered.

TABLE I. ONE POSSIBLE BLOCK SIZE LIST FOR THE DIFFERENT NUMBERS OF DISKS

Disk#	Block size(KB)	Chunk size(KB)
6	64	16
8	48	8
12	40	4
14	72	6
18	64	4
20	72	4
24	88	4
30	112	4
32	120	4
38	144	4
42	160	4
...	...	...

Then, the chunks locating at the same row with the starting point chunks can be recovered from the first labeling values. After that, we always can find one or two failed chunks that have the unique second labeling value among the rest of the failed chunks. The reconstruction follows the above process until all of the chunks are recovered. Figure 3 gives an example of the recovery process with $M = 6$ for one block-level stripe. Once one block-level stripe is recovered, the failed data blocks in other block-level stripes can be recovered using the same steps.

The chunk-level labeling algorithm is similar to the RDP-code and the correctness of RDP-code has previously been proven [13]. Thus, the Tier-code can recover at most two failed blocks in each block-level stripe. The block-level labeling ensures that each block-level stripe gets distributed across all disks. According to Eq. 1 and Eq. 2, one label value is evenly distributed across all disks for data blocks and parity blocks. Therefore, each block-level stripe can have at most two failed blocks when two disks fail. So, the Tier-code algorithm can tolerate up to two disk failures.

However, there are two limitations in Tier-code. The first one is that the number of disks must be one plus a prime number, greater than three. Second, since the chunk size is computed by Eq. 6, the block size must be divisible by the number of disks. Table I provides one solution for the block and chunk sizes for different numbers of disks.

$$Size_{block} = (M-2) * Size_{chunk} \tag{6}$$

IV. Write and Degraded Mode Read Performance Analysis

In this section, we analyze the performance of the RAID system to determine major factors which have influence on a complete RAID write operation and a degraded mode read operation. According to the major factors, we explain the reason why Tier-code has better performance on those factors than the previous codes.

A. *Performance Analysis for RAID Systems*

The write operation in a RAID-6 system consists of the following four major operations:

1) Reading the existing blocks (data blocks or parity blocks).
2) Computing parities with new data.
3) Writing data blocks.
4) Writing parity blocks.

First, the cost of operation #3 above is the same for all RAID-6 systems because the number of the data blocks written is the same for all systems employing RAID-6 codes with a same write request. Second, the overheads of operations #1 and #4 are determined by the number of parities written and the number of existing block reads, respectively. Thus, RAID-6 codes have different overheads of operations #1 and #4 caused by their labeling algorithms. Third, the overhead of #2 indicates the parity computation complexity. The computation complexity is varied in RAID-6 codes based on the number and types of operations for computing parties. Therefore, it is important to note that the performance of a complete write is primarily dependent on operations #1, #2 and #4. The effect of these operations is discussed in depth below.

For the operation #2, there is no doubt that the optimal operation for computing parities is the XOR operation and the minimum number of XOR operations for one parity computation with M disks is $M-3$ for RAID-6 systems. Therefore, to pursue the optimal computation complexity, some previous codes [9][10][12][18][21] use different data layouts.

For the operations #1 and #4, two metrics can be used to evaluate their overhead. The first one, update complexity [10], is the average number of updated parity blocks that are associated with a data block. The second one, multi-block access complexity [15], is defined in terms of the number of parity updates and read operations when a request needs to access multiple data blocks. The first metric applies to the scenario with only one data block update. The second metric reflects real life applications because it considers multi-block writes.

In the degraded mode read, if the data blocks are accessed by a request without containing any failed blocks, the degraded mode read request is the same as a normal read request. Otherwise, extra reads are required for recovering failed blocks. These extra reads are similar to the reads generated from a write operation for parity updates. Both of them need to read all of the available blocks in the same stripe. Thus, two operations primarily affect the degraded mode read performance – the extra non-failed block reads aiming to recover the failed blocks, and the decoding algorithm. The optimal decoding complexity is $M-3$ XOR operations for one block recovery with M disks, which is exactly the same as the encoding algorithm. For the extra non-failed block reads, we proposed a metric called degraded read complexity to indicate the overhead of reading non-failed blocks. Figure 4 provides an example to show the comparison of number of extra reads with two failed disks between different codes. In this example, Tier-code needs the minimum number of reads to recover the blocks from disk#3 and disk#5 compared to previous works. In the following subsections, we use analytical models to conclude that Tier-code has better performance than previous codes over all above factors. The results in Section V validate the conclusions according to the software and hardware implementation, and simulation results.

B. *Encoding and Decoding Complexity*

To compute parities (operation #2), there are three major types of codes used in RAID-6 systems, Reed-Solomon codes, optimized Cauchy Reed-Solomon code (CRS) [8][22][15] and XOR-based codes. First, the Reed-Solomon code is the most popular code used in practical RAID-6 systems like mdadm [23]. It uses Galois Field arithmetic to compute the second parities. Second, the CRS codes like PS-code optimized Reed-Solomon codes by reducing the encoding and decoding complexity. They convert the operations over $GF(2^w)$ to XORs by converting distribution matrices to binary matrices. Third, the XOR-based codes do not have distribution matrices and directly use XORs to compute parities according to labeling algorithms. The XOR-based codes includes P-code, X-code, RDP-code, HV-code, etc. and our Tier-code belongs to the XOR-based codes as well. To investigate their encoding and decoding complexity, we calculated the number of operations and the type of operations during encoding and decoding processes as seen in Table II and Table III.

In terms of the encoding process, the number of XOR operations is $(M-3)*2$ for the XOR-based codes. The number of XORs needed for the Cauchy Reed-Solomon codes are obtained from the Jerasure simulator [24]. RS-code faces extra multiplication operations during its encoding process. Therefore, Tier-code (XOR-based codes) require the least XORs in Table II. Tier-code requires about 1.5 times fewer XORs than PS-code. Compared to RS-code, Tier-code has no multiplications though they share the same number of XORs. For the decoding process, Tier-code has exactly the same number of XORs as its encoding process. For the decoding process of the CRS codes, it needs to calculate the inverse distribution matrix and then convert multiplications to XORs according to the inverse distribution matrix [25]. Since the inverse distribution matrices are different for different failed disk cases, Table III shows the average number of XORs for all possible failed cases. The RS-code has the normal decoding process as the common RAID-6 [26]. As the results shown in Table III, Tier-code has about 2.4 times fewer XORs than PS-code and has substantially fewer XORs and look-up tables (LUTs) than RS-code. In summary, Tier-code theoretically has much better performance than Cauchy RS-code and RS-code on the encoding and decoding processes.

C. *Complexity Evaluation for Write*

For the operations #1 and #4 for writes, we use the multi-block access complexity to evaluate the write performance.

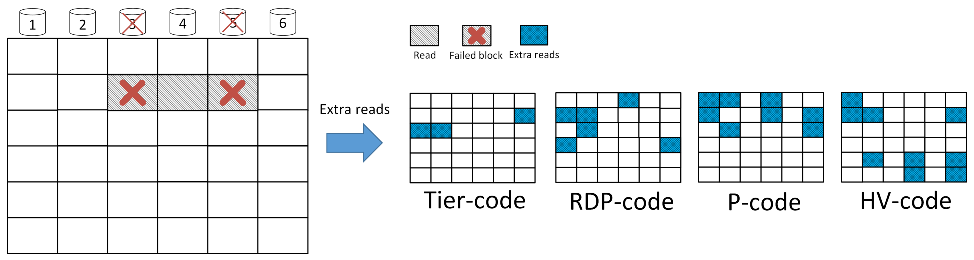


Fig. 4. An example of a degraded mode read request of three blocks for different codes with failed disk#3 and disk#5. RDP-code, P-code, and HV-code need 6, 8 and 9 extra reads, respectively. However, Tier-code only needs 3 extra reads.

TABLE II. COMPARISON OF THE AVERAGE NUMBER OF OPERATIONS BETWEEN DIFFERENT CODES DURING ENCODING.

disk#	Operation	Tier-code (XOR)	PS-code (CRS)	RS-code (RS)
6	$\oplus$	6	9.5	6
	$\times$	0	0	4
8	$\oplus$	10	15	10
	$\times$	0	0	6
12	$\oplus$	18	27.25	18
	$\times$	0	0	10
14	$\oplus$	22	32.75	22
	$\times$	0	0	12
18	$\oplus$	30	52.8	30
	$\times$	0	0	16
20	$\oplus$	34	58.8	34
	$\times$	0	0	18
30	$\oplus$	54	96.4	54
	$\times$	0	0	28
32	$\oplus$	58	104.2	58
	$\times$	0	0	30

TABLE III. COMPARISON OF THE AVERAGE NUMBER OF OPERATIONS BETWEEN DIFFERENT CODES DURING DECODING.

disk#	Operation	Tier-code (XOR)	PS-code (CRS)	RS-code (RS)
6	$\oplus$	6	16.5833	12
	LUTs	0	0	11.67
8	$\oplus$	10	24.6667	15.6
	LUTs	0	0	13.40
12	$\oplus$	18	42.1889	23.33
	LUTs	0	0	17.22
14	$\oplus$	22	51.0379	27.27
	LUTs	0	0	19.19
18	$\oplus$	30	73.7667	35.20
	LUTs	0	0	23.13
20	$\oplus$	34	83.2484	39.18
	LUTs	0	0	25.12
30	$\oplus$	54	130.4378	59.11
	LUTs	0	0	35.07
32	$\oplus$	58	139.9115	63.10
	LUTs	0	0	37.07

Note: Look-up tables (LUTs) are the operations to compute the inverse matrix multiplication. # of LUTs provides how many look-up-table operations are executed.

The complexity indicates the average number of operations with varying the write request sizes. It is defined in Eq. 7.

$$\begin{cases} Ave_{write} = (\sum_{i=1}^{M*(M-2)} ReqW_i)/(M*(M-2)) \\ Ave_{read} = (\sum_{i=1}^{M*(M-2)} ReqR_i)/(M*(M-2)) \\ Ave = Ave_{write} + Ave_{read} \end{cases} \tag{7}$$

where $ReqW_i$ is the number of parity updates with I/O request size of i, $ReqR_i$ is the number of reads with I/O request size of i, and Ave is the average number of operations for M disks.

Different numbers of disks are investigated for the degraded read complexity. Figure 5 compares Tier-code with previous codes in terms of the multi-block access complexity and Tier-code achieves the minimum number of parity updates and reads when varying the I/O request sizes for 6 disks. Moreover, Tier-code, RS-code and PS-code obtain the similar results because they use contiguous layouts. In addition, more investigations have been done with varying the number of disks. Figure 6a shows the average number of parity updates for different numbers of disks. Tier-code requires the fewest parity updates with the different number of disks. Additionally, as seen in Figure 6b, Tier-code requires much fewer reads than the others. Furthermore, the difference between Tier-code and the other codes becomes even larger when increasing the number of disks. In summary, Tier-code has the best multi-block access complexity and comes out ahead in operations #1 and #4 described in Section IV-A compared to the other codes.

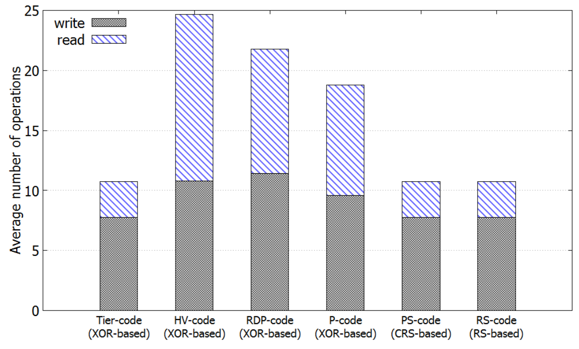
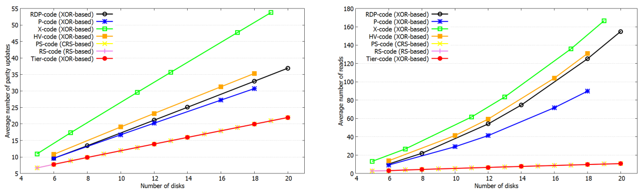


Fig. 5. The average number of operations for the multi-block access complexity with M=6.



(a) Parity updates (b) Reads

Fig. 6. Multi-block access complexity for different numbers of disks.

D. Complexity Evaluation for Degraded Mode Read

As discussed in Section IV-A, the two major operations determine the performance of degraded mode read, the decoding algorithm and the extra non-failed block reads to recover the failed blocks. We have discussed the decoding algorithm in Section IV-B. In this section, we focus on the number of extra reads to recover failed blocks.

A new metric called the *degraded read complexity* is proposed in this paper to compare the degraded mode read performance by counting extra read operations. In our evaluation of the degraded read complexity, we assume the location of the first block of each read operation follows a uniform distribution across all data blocks, 1 to $M*(M-2)$, in one matrix. Thus, the probability of each block being the starting point of a read is the same. For I/O request sizes, we only focus on the range from one block to $M*(M-2)$ blocks because all matrix arrays have the same label layout and the degraded mode reads will be repeated after the I/O request size is larger than $M*(M-2)$. Moreover, we consider all possible combinations of two failed disks. For the degraded read complexity, we calculate the average number of the extra reads for recovering the failed data blocks. This ensures that the degraded read complexity reflects the code's degraded mode read performance accurately after considering all combinations of two failed disks, different I/O request sizes and different request start points. The degraded read complexity is calculated from Eq. 8.

$$d = \frac{\sum_{f_1=1}^{M-1} \sum_{f_2=f_1+1}^{M} \sum_{i=1}^{M*(M-2)} \sum_{j=1}^{M*(M-2)} R_{f_1 f_2 ij}}{M^3 * (M-2)^2 * (M-1)/2} \quad (8)$$

where d is the degraded read complexity, f_1 is the first failed disk number, f_2 is the second failed disk number, i is the starting point of the requests varying from 1 to $M*(M-2)$, j is the I/O request size, and $R_{f_1 f_2 ij}$ is the total number of extra reads for recovering the failed data blocks when the I/O request size is j starting from the i^{th} block with failed disk #f_1 and failed disk #f_2.

The degraded read complexity metric in Eq. 8 is useful in comparing real systems since it analyzes the degraded mode read operation from several aspects, including number of disk failures, I/O request sizes, and request localities. Figure 4 is an example of the degraded mode read of three blocks. In this example, $R_{f_1 f_2 ij} = 3$ for Tier-code with $f_1 = 3$, $f_2 = 5$, $i = 9$, and $j = 3$. This example shows that Tier-code needs three extra reads during the degraded mode read when the I/O request starts from the 9^{th} block with a size of 3 blocks when disks #3 and #5 are failed.

As seen in Figure 7, Tier-code always has the lowest average number of extra reads compared to the previous RAID-6 codes. This is because Tier-code uses a contiguous labeling strategy on the block-level labeling, which is the same reason that Tier-code attains better write performance than previous codes. In terms of the degraded read complexity, Tier-code has 23.3% to 40.9% fewer extra reads on average than previous codes. Considering the conclusion from Section IV-B, Tier-code achieves the best performance for both factors in the degraded mode read.

In summary, Table IV indicates the comparison between different RAID-6 codes. For the parity computation complexity, Tier-code achieves the minimum number of XORs for computing parities. For the write and degraded mode read complexities, Tier-code obtains the minimum average number of operations because of its contiguous layout. Therefore, it shows that Tier-code theoretically achieves the best results on all metrics. In next section, we validate the conclusion of those RAID-6 codes through real systems tests and simulations.

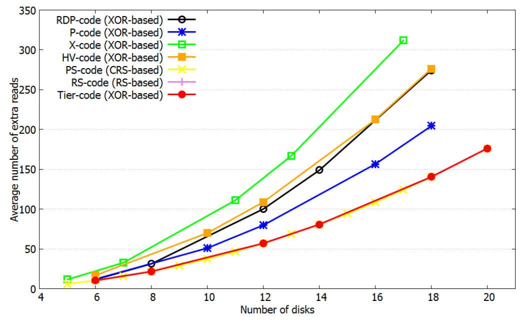


Fig. 7. Degraded read complexity for different numbers of disks showing that Tier-code has the minimum average number of extra reads.

V. Experimental Results

In this section, we present the experimental results from two perspectives – computation complexity evaluation and I/O performance simulation – to validate that Tier-code has better performance than other codes. For the computation complexity, we used two methods, software testing and ASIC design, to measure the computation performance and hardware overhead of different encoding and decoding algorithms. For the overall I/O performance, we employed the DiskSim simulator [27] to measure the write and degraded mode read performance.

A. Computation Complexity Comparison

We begin by comparing the computation complexity amongst different codes using two methods, software testing and ASIC design synthesis. The software testing results provide a straightforward comparison between Tier-code (XOR-based code), PS-code (Cauchy Reed-Solomon code) and Reed-Solomon code in terms of the encoding and decoding throughputs. The ASIC design synthesis shows the hardware cost of different codes when performing those encoding and decoding in the hardware implementation such as RAID hardware controllers.

In the software testing, we performed encoding and decoding processes in the real system. The 64-bit system has 4 Intel Core i7-4790 CPUs with 3.60 GHZ. It has 4GB RAM size. For the test process, we used total 1GB data to perform encoding and decoding and the type of data is char. The PS-code and RS-code are performed in GF(16). For encoding, the testing results are shown in Figure 8. Tier-code is always better than PS-code and RS-code with varying the number of disks. Tier-code has around 1.2GB/s encoding speed which is about 1.4 times better than PS-code and about 5 times better than RS-code. In Table II, Tier-code has about 1.5 times fewer XOR operations than PS-code. Therefore, the software test results match the theoretical performance estimation well. For decoding, in the software test, suppose there are two disks failed. Similar to encoding, the number of disks is varied from 6 to 14. As seen in Figure 9, Tier-code has similar decoding speed as its encoding speed because their numbers of XORs are exactly the same with each other. However, PS-code only attains about 500MB/s decoding throughput and RS-code has only about 120MB/s decoding speed. Tier-code has about 2.5 times and 10 times faster decoding speed than PS-code and RS-code for all disk number cases, respectively. Considering the I/O performance of modern disks, current HDDs can reach

TABLE IV. Comparison of RAID-6 codes.

Codes	Parity update complexity	Multi-block access complexity	Degraded read complexity	Computation complexity	Disk number
Tier-code	2 updates	**Low cost**	**Low cost**	**Optimal**	p+1
RDP-code [13]	more than 2 updates	High cost	High cost	**Optimal**	p+1
PS-code [15]	2 updates	**Low cost**	**Low cost**	Not optimal	Any
RS-code [7]	2 updates	**Low cost**	**Low cost**	Not optimal	Any
HV-code [16]	2 updates	High cost	High cost	**Optimal**	p-1
X-code [9]	2 updates	High cost	High cost	**Optimal**	p
P-code [10]	2 updates	High cost	High cost	**Optimal**	p-1

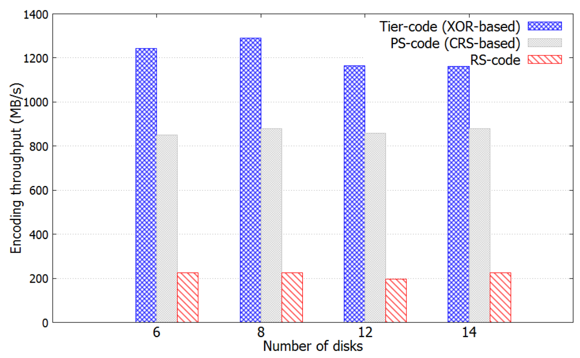


Fig. 8. Software measurement results for the encoding process with different numbers of disks.

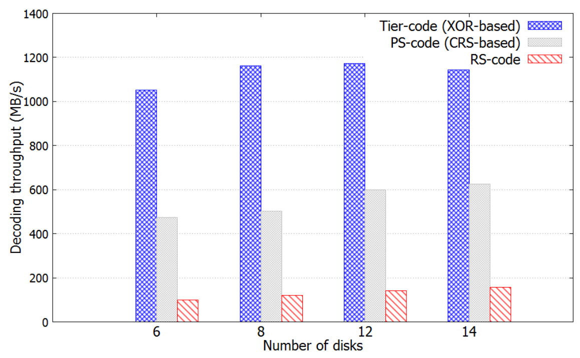


Fig. 9. Software measurement results for the decoding process with different numbers of disks. The recovery process is only for the case when disks #1 and #3 have failed.

over 150MB/s throughput and modern SSDs like Intel 3D Xpoint [28] even reach to over 2GB/s throughput. Therefore, the high speed of encoding and decoding in Tier-code will significantly improve performance of current RAID systems.

Additionally, we also explored the hardware implementation for those three types of codes. We used the design compiler to synthesize the three codes with FreePDK 45nm library [29]. The hardware ASIC design is different from the software implementation. For the hardware implementation, all operations can be performed in parallel, thus it becomes a trade-off between the hardware costs and encoding/decoding performance. In our implementation, we used 4 bits to express the data from each disk. To achieve a wider datapath or higher throughput, we can directly duplicate the circuits. As seen the encoding results in Table V, Tier-code has a little better results than PS-code in terms of area and power. The reason is that the parallelism in hardware implementation relieve the

TABLE V. Synthesis results for the encoding

	# of disks	Area (um^2)	Power (mW)	Frequency (GHz)	Energy (fJ)
Tier-code (XOR)	6	175.5	0.023	2.99	7.6
	8	369.3	0.046	2.41	19.1
	12	981.3	0.13	1.76	73.3
	14	1380.7	0.18	1.72	102.6
PS-code (CRS)	6	182.6	0.023	2.46	9.5
	8	380.8	0.049	1.77	27.8
	12	1033.6	0.14	1.30	105.2
	14	1396.6	0.19	1.15	163.8
RS-code (RS)	6	1796.5	0.11	1.06	99.3
	8	4022.4	0.23	0.97	240.4
	12	11143	0.58	0.82	709.1
	14	16035	0.92	0.82	1115.8

TABLE VI. Synthesis results for the decoding

	# of disks	Area (um^2)	Power (mW)	Frequency (GHz)	Energy (fJ)
Tier-code (XOR)	6	173.6	0.029	3.20	9.0
	8	370.3	0.056	2.67	20.9
	12	980.4	0.14	1.74	81.8
	14	138.2	0.19	1.63	117.3
PS-code (CRS)	6	214.0	0.033	1.27	26.4
	8	455.5	0.072	1.11	64.6
	12	1132.2	0.16	1.14	143.4
	14	1554.3	0.22	1.21	182.8
RS-code (RS)	6	2945.3	0.25	0.38	669.1
	8	5741.4	0.47	0.38	1238.7
	12	13987	1.04	0.38	2762.2
	14	19268	1.39	0.38	3707.3

difference between Tier-code and PS-code. Additionally, the synthesis optimization in design compiler may narrow the computation gap between two codes. For example, in Eq 9, there are 6 XORs to compute P and Q. However, in the hardware implementation, it can reduce to 5 XOR gates by reusing $a_0 \oplus a_2$ in two equations.

$$\begin{aligned} P &= a_0 \oplus a_1 \oplus a_2 \oplus a_3 = (a_0 \oplus a_2) \oplus a_1 \oplus a_3 \\ Q &= a_0 \oplus a_2 \oplus a_4 \oplus a_6 = (a_0 \oplus a_2) \oplus a_4 \oplus a_6 \end{aligned} \quad (9)$$

In addition, Tier-code is about 21%-47% faster on frequency and saves 25% - 60% energy compared to PS-code. Compared to RS-code, Tier-code has much better results in the hardware implementation because RS-code uses the look-up tables (LUTs) to implement the multiplication over GF(16). For the decoding results in Table VI, Tier-code derives much lower hardware cost than PS-code and RS-code. Tier-code attains 18%, 19%, 44% and 133% less hardware costs than PS-code and 11.78X, 5.66X, 2.14X and 20X less hardware cost than RS-code in terms of area, power, frequency and energy.

B. DiskSim Simulation Results

This section focuses on the I/O performance. We used DiskSim [27] simulations to compare the I/O performance of

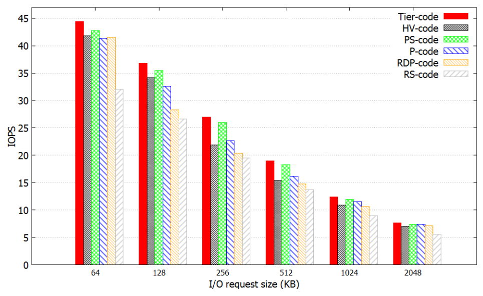


Fig. 10. Sequential write performance between RAID-6 codes with M=6 while varying I/O request sizes.

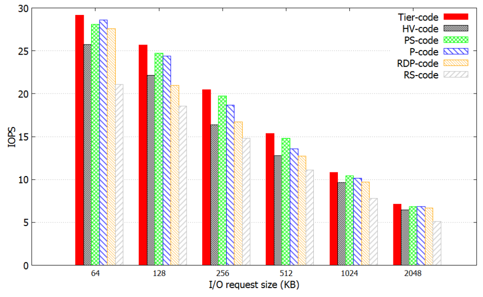


Fig. 11. Random write performance between RAID-6 codes with M=6 while varying I/O request sizes.

the different codes. The encoding and decoding computation time is integrated in the DiskSim simulation for write and degraded read performance. Since Tier-code is limited to (a prime number + 1) disks, and the previous RAID codes also have limitations on the number of disks they can use, we choose 6 disks for our experiments. The simulations use 64KB as the default block size and the total number of disks is 6. We use 10,000 I/O requests for sequential and random writes and the I/O request size is varied from 64KB to 2048KB. The reason for varying the request size is that, if one request size can completely cover the whole matrix data array (one matrix data array size is 24*64KB=1536KB for 6 disks), then these RAID-6 codes would have same the write performance since they have the same number of data and parity blocks to write. Thus, if the request is larger than 1563KB, we only need to focus on the incompletely covered parts of the blocks. So, we ignore the I/O request sizes larger than 2048KB.

As shown in Figures 10 and 11, Tier-code has the best write performance compared to the other codes. Tier-code improves the sequential write performance 20.5%, 14% and 11.5%, and random write performance 15%, 16.3% and 6.8% on average compared to RDP-code, HV-code and P-code, respectively. The primary reason for the improved performance is the smaller number of parity updates and the smaller number of reads for computing new parities. Compared to PS-code and RS-code, Tier-code achieves about 3.8% and 38.4% better write performance. This is because Tier-code has much better encoding speed than PS-code and RS-code.

To compare the degraded read mode operation of Tier-code to the other codes, we consider the cases of one and two disk failures. Even with two disk failures, a RAID-6 system can correctly continue to read data since the system can use a combination of uncorrupted data and parity blocks

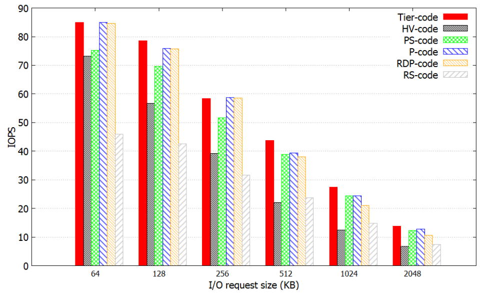


Fig. 12. Sequential read performance in the degraded mode with one disk failure for RAID-6 with M=6.

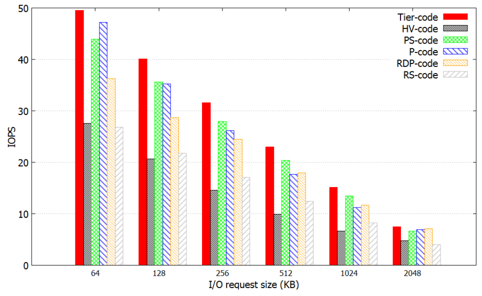


Fig. 13. Random read performance in the degraded mode with one disk failure for RAID-6 with M=6.

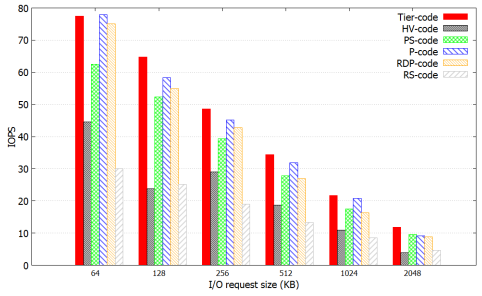


Fig. 14. Sequential read performance in the degraded mode with two disk failures for RAID-6 with M=6.

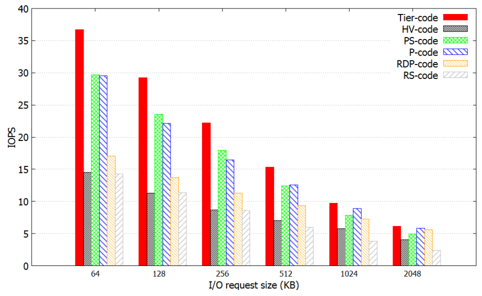


Fig. 15. Random read performance in the degraded mode with two disk failures for RAID-6 with M=6.

to recover the failed data blocks in the same stripe. For the one disk failure case, as shown in Figures 12 and 13, in the degraded mode, Tier-code on average derives 86.4%, 12.8%, 12.3%, 20.7%, and 84.6% higher performance than HV-code, PS-code, P-code, RDP-code, and RS-code, respectively. For the two disk failure case, as shown in Figures 14 and 15, in the degraded mode, Tier-code on average obtains 117.5%, 23.8%, 15.7%, 46.9%, and 157% performance improvement compared to HV-code, PS-code, P-code, RDP-code, and RS-code, respectively. The main reason is that Tier-code reads fewer data and parity blocks, and takes less decoding time when recovering the failed blocks. Compared two cases between one disk and two disk failure, the two disk failure has larger performance difference between those codes because two disk failure has higher probabilities to read failed blocks and also need to read more available blocks to recover them. Therefore, it enlarges the performance difference and Tier-code attains larger performance gains on two disk failure conditions than on the one disk failure conditions compared to previous RAID-6 codes.

VI. Conclusion

This work proposed a new RAID-6 code called Tier-code that focuses primarily on improving the write performance and the degraded mode read performance of redundant arrays of independent disks. Our new labeling approach reduces the number of parity updates that need to be written to the disks and results in optimal parity computation complexity. A new metric called degraded read complexity demonstrates the extra number of reads in the degraded-mode read. Different complexity metrics show that Tier-code has the best computation time, fewer write updates and fewer reads in terms of computation complexity, write performance and degraded-mode read performance, respectively. In the experimental results, the analytical model conclusions are validated for computation time and I/O performance. Computation time is compared using real system measurement and ASIC synthesis results. The Disksim simulator is used to validates the overall I/O performance conclusion. These results show that Tier-code improves the write and the degraded mode read performance compared to the previous codes.

VII. acknowledgment

This work was supported in part by the Center for Research in Intelligent Storage (CRIS), which is supported by National Science Foundation grant no. IIP-1439622 and member companies. The work of S. Mohajer is supported in part by the National Science Foundation under Grant CCF-1617884. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] Fenggang Wu, et al. Data management design for interlaced magnetic recording. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*. USENIX Association, 2018.

[2] M. Minglani, et al. Kinetic action: Performance analysis of integrated key-value storage devices vs. leveldb servers. In *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*, pages 501–510, Dec 2017.

[3] Eduardo Pinheiro, et al. Failure trends in a large disk drive population. In *FAST*, volume 7, pages 17–23, 2007.

[4] David A. Patterson, et al. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, SIGMOD '88, pages 109–116, New York, NY, USA, 1988. ACM.

[5] Bingzhe Li, et al. Tracerar: An i/o performance evaluation tool for replaying, analyzing, and regenerating traces. In *Networking, Architecture, and Storage (NAS), 2017 International Conference on*, pages 1–10. IEEE, 2017.

[6] J. Axboe. Fio - flexible i/o tester. http://freshmeat.net/projects/fio/.

[7] Irving S Reed et al. Polynomial codes over certain finite fields. *Journal of the Society for Industrial & Applied Mathematics*, 8(2):300–304, 1960.

[8] James S Plank et al. Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications. In *Network Computing and Applications, 2006. NCA 2006. Fifth IEEE International Symposium on*, pages 173–180. IEEE, 2006.

[9] Lihao Xu et al. X-code: Mds array codes with optimal encoding. *Information Theory, IEEE Transactions on*, 45(1):272–276, 1999.

[10] Chao Jin, et al. P-code: A new raid-6 code with optimal properties. In *Proceedings of the 23rd international conference on Supercomputing*, pages 360–369. ACM, 2009.

[11] Lihao Xu, et al. Low-density mds codes and factors of complete graphs. *Information Theory, IEEE Transactions on*, 45(6):1817–1826, 1999.

[12] Mario Blaum, et al. Evenodd: An efficient scheme for tolerating double disk failures in raid architectures. *Computers, IEEE Transactions on*, 44(2):192–202, 1995.

[13] Peter Corbett, et al. Row-diagonal parity for double disk failure correction. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, pages 1–14, 2004.

[14] Ping Xie, et al. V 2-code: A new non-mds array code with optimal reconstruction performance for raid-6. In *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, pages 1–8. IEEE, 2013.

[15] Bingzhe Li, et al. Ps-code: A new code for improved degraded mode read and write performance of raid systems. In *Networking, Architecture and Storage (NAS), 2016 IEEE International Conference on*, pages 1–10. IEEE, 2016.

[16] Zhirong Shen et al. Hv code: An all-around mds code to improve efficiency and reliability of raid-6 systems. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 550–561. IEEE, 2014.

[17] James Lee Hafner. Weaver codes: Highly fault tolerant erasure codes for storage systems. In *FAST*, volume 5, pages 16–16, 2005.

[18] Chao Jin, et al. A comprehensive study on raid-6 codes: Horizontal vs. vertical. In *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*, pages 102–111. IEEE, 2011.

[19] Kumar Chinnaswamy, et al. Read-modify-write operation, April 16 1991. US Patent 5,008,886.

[20] James S Plank. The raid-6 liber8tion code. *International Journal of High Performance Computing Applications*, 2009.

[21] Yingxun Fu et al. D-code: An efficient raid-6 code to optimize i/o loads and read performance. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 603–612. IEEE, 2015.

[22] Xiaowen Chu, et al. Perasure: a parallel cauchy reed-solomon coding library for gpus. In *Communications (ICC), 2015 IEEE International Conference on*, pages 436–441. IEEE, 2015.

[23] Linux raid. https://raid.wiki.kernel.org/index.php/Linux_Raid.

[24] James S Plank, et al. Jerasure: A library in c/c++ facilitating erasure coding for storage applications-version 1.2. Technical report, Technical Report CS-08-627, University of Tennessee, 2008.

[25] James S Plank, et al. A performance evaluation and examination of open-source erasure coding libraries for storage. In *FAST*, volume 9, pages 253–265, 2009.

[26] H Peter Anvin. The mathematics of raid-6. http://www.dei.unipd.it/~capri/LDS/MATERIALE/raid6.pdf.gz.

[27] John S Bucy, et al. The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101). *Parallel Data Laboratory*, page 26, 2008.

[28] Intel. Revolutionizing memory and storage. https://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html.

[29] James E Stine, et al. Freepdk: An open-source variation-aware design kit. In *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*, pages 173–174. IEEE, 2007.