PS-Code: A New Code For Improved Degraded Mode Read and Write Performance of RAID systems

Bingzhe Li
University of Minnesota-twin cities
Email: lixx1743@umn.edu

Manas Minglani
University of Minnesota-twin cities
Email: mingl001@umn.edu

David J. Lilja
University of Minnesota-twin cities
Email: lilja@umn.edu

***Abstract*—The newer storage systems often have to deal with multiple disk failures. Thus, RAID (Redundant Array of Independent Disk) is becoming increasingly important for two primary benefits: (1) high read performance and (2) robust fault tolerance. In addition, the recent RAID codes achieve better IO performance by using only XORs (optimal computational complexity) to compute parities. However, this achieved optimal computational complexity limits the further IO performance improvement. In this paper, we are proposing a new code called PS-code (Parity Stripe code) that employs improved Cauchy Reed Solomon codes for computing the parity to further improve write and degraded mode read performance. Furthermore, we extend the novelty by reducing the number of writes required for updating the parity. We simulated the PS-code in the DiskSim environment and devised a new metric called multi-block access complexity to perform an improved evaluation of the performance of the PS-code on workloads that represent real life scenario such as multi-block updates. Also, the experimental results demonstrate that the PS-code, on average, achieves 66%-86% better write performance and achieves 8.9%-23.6% higher degraded-mode read performance compared to previous works including P-code, RDP code, X-code, and HV-code. Finally, the comparison between the vertical and horizontal codes demonstrates that the vertical codes have better read performance than the horizontal codes in most cases.**

I. Introduction

Redundant Array of Independent Disks (RAID) [1] as a storage technology uses multiple disks to build a faster and a more reliable disk system. With the development of the disks, the storage capacity has been steadily increasing but the disks' failure has remained a big concern [2][3][4]. Therefore, researchers are investigating different methods for recovering from the failure of the RAID systems. RAID code is considered one of the key methods to recover from the failure system.

RAID code is of two types - horizontal and vertical. The main distinguishing aspect between the two is the location of the parity in the storage media. For the horizontal codes seen in Figure 1, the parity is stored in the same stripe with its corresponding data. In contrast, the parity for the vertical codes is stored along the columns, with the data, as shown in the Figure 2. Recently, some new types of vertical codes were proposed such as X-code[5], P-code[6], B-code[7] and D-code[8], which mainly focus on achieving better IO performance and balanced I/O workload by using optimal algorithmic complexity of the code. However, the disadvantages of these vertical codes based system is that the write performance and degraded mode read are inefficient because of the layout of the data and the parity blocks. Also, a few of the horizontal codes including EVENODD[9], RDP-code[10], and hybrid codes such as HV-code[11] demonstrates inefficient write performance. To achieve the least computation time for computing the parities all the above codes employ only XORs. This results in achieving optimal computational complexity. However, the optimal computational complexity limits the further write performance improvement. Therefore, there is an opportunity to devise a new code to improve the write performance.

There are two major factors that affect the write performance for a I/O request of the RAID system - the complexity of computing the parity and read-modify-write operation. Generally, one I/O request write can generate multiple requests, which are written sequentially on multiple blocks. We define these writes as "contiguous writes". These"contiguous writes" will update the parity and perform the read-modify-write operation in most of the cases to fulfill the I/O request. As an illustration, an I/O request of 1024 KB will be broken into 16 blocks (one default block is of 64KB). All these 16 blocks will be written sequentially. However, several of the recent works such as X-code, P-code, D-code, EVENODD, RDP and HV-code do not demonstrate good performance for contiguous writes. In our experimental results, we have varied the I/O request size to different sizes for a thorough experimentation. And a metric called multi-block access complexity is proposed to reflect the multiple blocks' write performance in real life scenario.

In this paper, our main purpose is to optimize the IO operations for the write performance and improve the degraded mode read performance. We propose a new RAID code called Partial Stripe code (PS-code) which gives a significantly better write performance and the degraded mode read performance over the recently released RAID codes[5][6][10][11][12][13]. The PS-code is one type of the vertical RAID code and it employs the improved Cauchy Reed-Solomon(CRS) code [14] to compute the parity. Additionally, by using a new labeling algorithm it reduces the number of the parity updates and reads for contiguous writes.

978-1-5090-3315-7/16/$31.00 ©2016 IEEE

We make the following contributions in this paper:
1. We propose a new RAID based code called PS-code. This code optimizes the read-modify-write operation during the write operation and it has the least number of parity updates and read operations amongst the X-code, P-code, RDP code, HV-code and etc. As a result, with the PS-code, we achieve better write performance than the above mentioned codes for RAID-6 .
2. A new metric called multi-block access complexity is proposed to evaluate the write performance. The results demonstrate that the metric can be effectively used to compare the write performance between the RAID systems.
3. The PS-code gave us better degraded mode read performance than other codes for two disk failures.
4. We build the performance and the recovery models for the different RAID codes - the vertical and horizontal codes. These models help us in making a comprehensive comparison between the PS-code and other RAID codes, and between the vertical and horizontal RAID codes.

The remainder of this paper is organized as follows: The motivation analysis for the PS-code is given in Section II. Section III discusses the background and the related work, which covers the horizontal and vertical encoding methods. Section IV describes the algorithms devised for our PS-code. Furthermore, the experimental methodology and the measured results for the PS-code and the previous RAID codes are presented in the Section V. We also make a thorough read performance comparison between vertical and horizontal RAID codes. Finally, the conclusion is presented in the section VI.

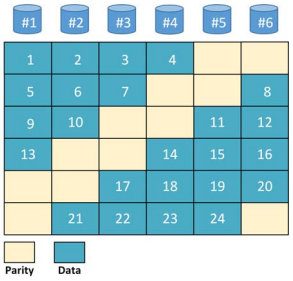

Fig. 1. Horizontal RAID-6

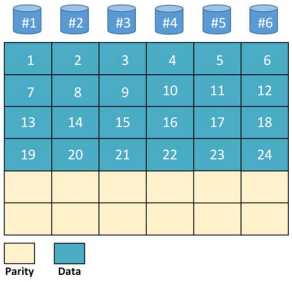

Fig. 2. Vertical RAID-6

II. Motivation Analysis

A. Terms and Notations

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

- **Multi-block Access Complexity:** It is defined in terms of the number of parity updates and read operations when a request needs to access multiple data blocks.
- **Update Complexity:** The concept of "update complexity" means the number of parity blocks associated when one data block is updated [15].
- **Computational Complexity:** The computational complexity corresponds to the operations it takes to calculate the parities in RAID systems. If a RAID code only uses XOR to compute parities, the RAID code is regarded as having optimal computational complexity.
- **Data Block Labeling and Parity Block Labeling:** The labeling demonstrates that the parity and data blocks are dependent on each other. A parity is computed from those data blocks that share the same label value as that with the parity's label. As seen in Figure 7, p1 and q1, the two parity blocks labeled "1" are computed from the four data blocks labeled "1".
- **Stripe:** A set of blocks of data and parity that have the same label value.
- **Read-modify-write:** Suppose a write request can't cover the whole strip arrays, RAID system can't simply write the small portion of data. It must read old data blocks first, read new data from host and then compute the new parity with those data. After computing the parity, it can write the new data to target location and then update new parity.
- **Degraded Mode Read:** A RAID system enters degraded mode when the system experiences a failure of one or more disks. If one request wants to read the failed disks, the operation will read other uncorrupted data and parity blocks correspondingly. Thus, these other data blocks can reconstruct the damaged data based on the system's reconstruction algorithm.
- **Storage Efficiency:** The storage efficiency of the RAID system is the percentage of the total size of the system that can be used for user data, which also can be computed by the total data blocks divided by total number of blocks in RAID system. In general RAID-6 systems, it equals to (M-2)/M. (M is the number of disks.)

B. Motivation Analysis

There are two major factors that affect the write performance of the RAID system - the computational complexity for computing the parity and the read-modify-write operation. The recently devised RAID codes including P-code, X-code, and others have achieved optimal computational complexity. The other attempts by Plank *el al* [14] [16] were made by speeding up the Reed-Solomon code and these efforts made the computation time closer to the optimal computational complexity. In addition, with the improvement of the CPU clock speeds and parallel technology, the computation time is much less important than the parity updates and read operations. Thus, the bottleneck of the write I/O request performance in the RAID systems is the read-modify-write operation including the number of parity updates and read operations generated by write request because it takes relatively long time to access the disks.

The motivation for improving the previous RAID-6 codes[6][8][9][10][11] is that the further IO performance improvement is limited by the optimal computational complexity. Moreover, the write performance and degraded mode read are

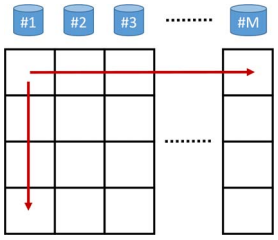

Fig. 3. Accessing order of RAID systems is from left to right, top to bottom with a "contiguous" write.

inefficient because of the layout of the data and the parity blocks of those codes. Therefore, there is an opportunity to improve the above mentioned performances.

In terms of these coding methods, the concept of the update complexity means the number of the parity blocks associated when one data block is updated. However, the real applications are not restricted to updating just one data block. In fact, they update several data blocks, whose overhead associated with the applications is not good with these codes.

For the "contiguous writes", as shown in Figure 3, the order of access is from left to right and then from top to bottom. The parity updates for "contiguous writes" depend entirely on the data labeling algorithm. And the previous works such as P-code, HV-code, and RDP-code have suffered from excessive data parity updates due to their distributed data labels. As seen in Figure 4, the PS-code has the least number of read operations and parity updates for one write because of the optimized labeling algorithm. Therefore, this new proposed code (PS-code) optimizes or improves the write performance associated with the extra read and parity updates. The PS-code with the labeling algorithm is discussed in detail in Section IV.

III. Background and Related Works

Several different RAID levels have been developed. For instance, RAID-0 contains the data stripes without the parity. To add the fault tolerance mechanism, RAID-1 uses the mirroring scheme and replicates the data on two or more disks. However, this methodology severely affects the disk utilization. Moreover, RAID-5 incorporates the block-level stripes and distributes the parities. It is successful in recovering from the failure of one disk. However, there is often a need to recover from multi-disk failure. RAID-6 provides fault tolerance for up to two failure disks because of its two distributed parities. The RAID system has high read performance due to parallel read operations. However, the write operation introduces overhead. For RAID-5/RAID-6, the write penalty is incurred from computing the parity, reading the existing data, writing the new data, and writing the new parity. The purpose of reading the existing data is to re-compute the parity in order to keep the system reliable when the new write is unable to cover the full stripe data. Therefore, the write performance is slowed down by the above mentioned discrete operations.

Some of the previous works are introduced below. The comparison of our PS-code with other codes is present in the

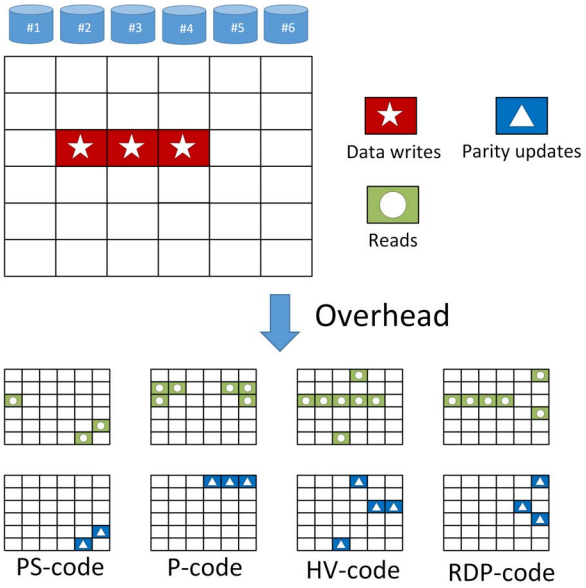

Fig. 4. An example of a write request of three blocks for different codes. The P-code needs 3 parity updates and 3 extra reads to finish the write request. HV-code needs 4 extra writes and 8 extra reads. RDP code needs 3 extra writes and 6 extra reads. Thus, the PS-code has the least number of the read operations and parity updates in this case. Section V will give more cases analysis and comparisons.

Section 5.
Horizontal codes:
Reed-Solomon code[17]: Reed-Solomon uses Galois Field arithmetic ($GF(2^w)$) (where w is the number of columns for the encoding matrix) to calculate the second parities. However, the computation overhead is very expensive.
EVENODD code[9]: This code is similar to the Reed-solomon coding and it uses the diagonal data block to calculate the second parities by XORing an intermediate value S. The disk number is restricted to prime number plus 2. The EVENODD code is neither optimal for computational complexity nor for the update complexity.
RDP Code[10]: This code is similar to the EVENODD code in terms of using the diagonal data blocks to calculate the second parities. However, it directly XORs all diagonal data blocks instead of introducing an extra intermediate value S, as in EVENODD code. Therefore, it is able to obtain the optimal computational complexity. Its disk number is restricted to prime number plus 1.
Liberation code[18]: Liberation codes are based on a bit matrix-vector product. They proposed a new class of X_i matrices for the liberation codes and use the notion of "bit matrix scheduling" to improve the decoding performance dramatically. However, its computational complexity is not optimal.

The codes mentioned above affect deeply on the vertical RAIDs. Some of the vertical codes use the similar labeling methodology as the horizontal codes including D-code [8] with RDP code. However, the performance is different because the

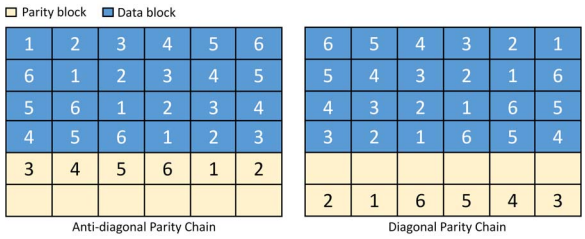

Fig. 5. An example of labeling for X-code with 6 disk drives

Fig. 6. An example of labeling for P-code with 6 disk drives

parities for the vertical RAIDs are placed in the additional rows. Introduction of the vertical codes is given below:

X-code [5]: X-code has the p*p structure (where p is a prime number) and it uses data blocks(first p-2 rows) of the two diagonals with slope 1 and -1 to compute the first and second row parity (last 2 rows). The example illustrating the labeling is shown in the Figure 5. It has the optimal computational complexity and update complexity.

P-code [6]: P-code has the (p-1)/2*(p-1) structure(where p is a prime number). The first row contains the parity blocks and the rest of the (p-3)/2 rows are data blocks. Each party block is assigned its disk number and each data block is assigned a two integer tuple (m, n). Moreover, the tuple (m, n) follows equation 1. The parity with integer i is computed by the data blocks with their patterns containing integer i. Figure 6 shows an example with p=7.

$$C = \{(m,n)|(1 \leq m,n \leq p-1), m \neq n, m+n \neq p\} \quad (1)$$

There is another RAID-6 code called HV-code [11], which has optimal computational and update complexity. They take advantage of the horizontal and vertical parities to reduce the IO requests. However, their "spare" labeling still generates several parity writes and limits their I/O performance.

IV. PS-CODE

In this section, the PS-code for RAID-6 system is explained along with the construction and re-construction algorithm of the PS-code. The primary advantage of this code is that it has the least number of reads and parity updates amongst the several previous codes. Moreover, we also describe the extension of the PS-code to more parity rows, which makes the PS-code tolerate multiple disks' failure. However, previous codes can only handle two disks' failure.

A. The PS-code of Data and Parity Labeling and Construction

The main purpose of the PS-code is to reduce the number of the reads and parity updates during the write operations aiming to improving the write performance of the RAID system. For the RAID-6 system, for each data block write, at least two parity block updates have to take place. This is the scenario that is being exploited or considered in the previous works [5][6][8][9][11][15][18]. However, in the real life scenario, multiple blocks are updated together for each write I/O request. This scenario for RAID-6 codes has not been evaluated from those works. We demonstrate that in terms of each request involving multiple block writes, the number of blocks written for updating the parity are lesser with our PS-code than the number of blocks required by other codes.

The PS-code algorithm can be divided into three primary steps: data block labeling, parity block labeling and construction.

1) Data block labeling: We assume the block matrix of size $M * M$ in Figure 7 represents the data blocks and the parity blocks. The values i (0<i≤M) and j (0<j≤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 j-th disk, and L_{ij} is its corresponding data block's label value. In this step, we assign integer labels to each data block. And the integer label L_{ij} can be obtained from the equation (2).

$$L_{ij} = \lfloor [i * M + (j-1) - 2]/(M-2) \rfloor \quad (2)$$

2) Parity block labeling: In this step, the first parity row is labeled using the equation (3) and the second parity row is labeled using the equation (4). In these equations, r is the parity's label value (1<r≤M). k is the disk number(or column index) for the parity. And the gcd() is the function to calculate greatest common divider.

$$k = [r * (M-2) + \lfloor \frac{(r-1) * gcd(M,2)}{M} \rfloor + 1] \% M \quad (3)$$

$$k = [r * (M-2) - \lfloor \frac{(r-1) * gcd(M,2)}{M} \rfloor + 2] \% M \quad (4)$$

As shown in the Figure 7, with an example of 6 disks, label values of data or parity blocks are distributed across all disks, which are computed from equation (3) and (4). When $gcd(M,N) = 1$(M is the number of disks, N is the number of parity rows), the labels for the second of the parity rows can be directly obtained from the first row's labels by shifting to right one block. If there are more than two parity rows, then the labels for a particular row is obtained by shifting the block label of the previous row by one block to the right. Fig.10 gives an example with $gcd(7,3) = 1$.

3) Construction: We have used Cauchy Reed-Solomon (CRS) matrix [19] to calculate the parities. According to the Cauchy matrix definition given by Luby *et al* [19], $X = \{x_1, ..., x_n\}$ *and* $Y = \{y_1, ..., y_m\}$ are the two sets of elements in $GaloisField(2^w)$, where m represents the number of disks, n is the number of disks failures that can be tolerated, and 2^w bits is the word size ($m+n <= 2^w$). So, the elements

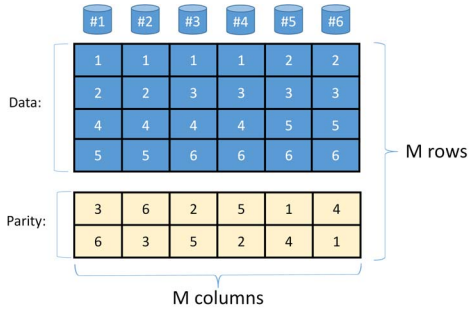

Fig. 7. An example of the PS-code for RAID-6 with M=6. The labels of first and second parity rows are calculated from equations (3) and (4). As an illustration, the two parity blocks p1 and q1 labeled "1" are computed from the 4 data blocks labeled "1".

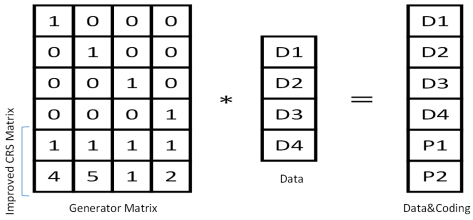

Fig. 8. An illustration of generator matrix for M = 6. The cauchy matrix is incoporated inside of generator matrix.

in the Cauchy Matrix can be obtained with the below given equation.

$$F_{ij} = 1/(x_i + y_j) \tag{5}$$

Plank *et al* [14] used the optimal X and Y sets to minimize the number of XORs in order to improve the time taken for the encoding. Since this process reduces the number of XORs for parity computation, we apply it to our code to construct our parity. We apply the optimized X and Y sets to build a Cauchy matrix, which is to be incorporated inside the generator matrix of the PS-code, as shown in the Figure 8. This method helps us in reducing or optimizing the parity computation time.

For different n, m and w values, the optimal X and Y sets will be different. Plank *et al* in the horizontal RAID codes uses all the data blocks in a row to find the encoding. Since our PS-code is a vertical code, we use only those data blocks which have the same label value, such as D_1 to D_4, as shown in the Figure 8.

Thus, the parities of the PS-code are computed by performing the matrix multiplication between the improved CRS and data matrix. Figure 8 provides an example of the parity computation with label r=1. In equation (6) and (7) in $GF(2^3)$, the first parity row are computed by performing the XOR operation on the data blocks. The parities of the second row are calculated by multiplying the last row of the generator matrix with the data matrix.

$$P_{k_1=5,r=1} = D_{11} + D_{12} + D_{13} + D_{14} \tag{6}$$

$$P_{k_2=6,r=1} = 4 * D_{11} + 5 * D_{12} + D_{13} + 2 * D_{14} \tag{7}$$

Where, the k_1 and k_2 are the parity column index numbers for the first and second parity row respectively. $P_{k,r}$ is the parity value for k-th disk with the label value r.

B. Reconstruction

In this section, the PS-code reconstruction algorithm demonstrated as recovering from one-disk failure or two-disk failure is based on RAID-6 system.

During encoding, the first row of the improved CRS matrix (in Figure 8) contains all "1", therefore, the values of the data matrix were XORed together. So, for one-disk failure, the reconstruction operation only needs to use the XOR operation. Any failed data block can be recovered from other (M-2) blocks that share the same data label with this failed block. These other (M-2) blocks, including the data blocks and the parity block from the first parity row, are XORed together to recover the failed data block. Similarly, all other failed blocks in the failed disk can be recovered using the same process as previously explained.

For the two-disk failure, as shown in Figure 8, the reconstruction process can use the parity check matrix to recover the failed blocks. A similar recovery process is explained by Plank *et al* [20] [14]. For our recovery process of the failed disks there are three major steps:

1) Determining the labels of the data blocks in the two failed disks
2) Determining the parity check matrices for the data blocks whose labels appear twice. Moreover, the data blocks whose label appears only once, can be recovered directly.
3) Recovering the failed blocks by multiplying their parity check matrix with the matrix, which consists the rest of the healthy blocks, which are in the same stripe.

In Figure 7, it can be observed that blocks with the same label "r" are evenly distributed across all the disks. Total number of the blocks with label "r" is M. Two of them are parity blocks and the rest M-2 blocks are data blocks.

Since each disk contains different labels, the PS-code needs to use several parity check matrices to recover the two failed disks rather than using just one parity check matrix, as employed in the previous horizontal code by Plank *et al*[14]. Figure 9 demonstrates a reconstruction example with a failure of disk#3 and disk#4. One column in the Figure 9 represents different label values associated to each disk and we need different parity check matrices to recover them. According to the reconstruction steps, given above, the failed disk#3 and #4 contains the data blocks with the label value 1, 3, 4 and 6. As seen in Figure 9, the data blocks with label 1 and 4, and data blocks with label 3 and 6 will share the same parity check matrix respectively. So, for this example, we need two parity check matrices.

C. Construction and Computational Complexity

We have quantified the computation overhead of our PS-code by the number of XORs, in order to compare with the previous works. The modified CRS by Plank [14] can

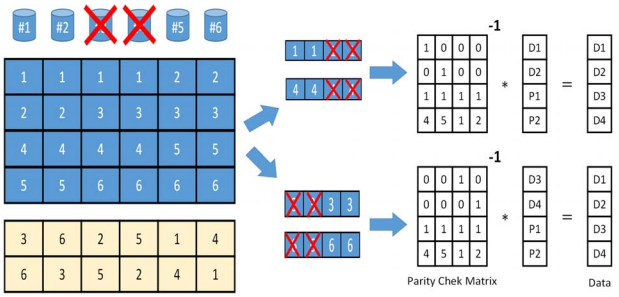

Fig. 9. An example showing PS-code when disk#3 and disk#4 failed. The recovery process involves two different parity check matrices.

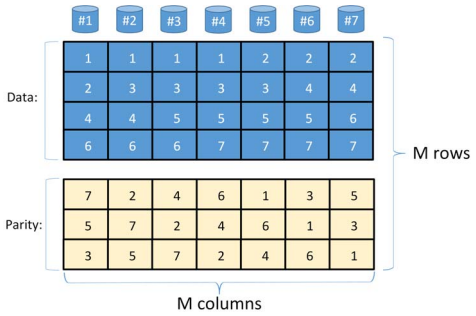

Fig. 10. An example of the PS-code of 3 disk failure tolerance with M=7. Since gcd(M, 3)=1, the labels of parity can be obtained from shifting its previous parity row label.

change the multiplication over GF(2^w) to XOR operations. Each element in GF(2^w) can be represented as a $w * w$ matrix bits. We apply this property to PS-code to convert the multiplication operations to XORs.

As seen in Table I, our PS-code needs more number of XORs to finish the encoding process than other codes. The PS-code has better write performance than other codes but there is an additional computational overhead as discussed in Section V. Additionally, the storage efficiency of the PS-code is same to other codes and it is (M-2)/M.

D. PS-code extension to N-row parities

In the previous sections, we demonstrated that the RAID-6 for PS-code is capable of recovering from 2 disk failures for any number of disks ≥ 4 (like the horizontal RAID-6 by using Reed-Solomon). However, most of the previous RAID codes with optimal computational complexity have limited their disk numbers to prime or prime-1, etc. [8][21][5][6].

In addition, the PS-code is scalable and can be used to recover from multiple disk failures. The RAID system that can recover from N disk failures is called PS-N. The improved CRS matrix for the PS-N code can be generated from the Jerasure simulator [22].

The data block label algorithm is the same as the equation (2). The parity label algorithm is shown in equation (8). As discussed earlier, when $gcd(M, N) = 1$, the labels for an individual row can be obtained by shifting the labels of the previous row to the right, by one block, as described in Section IV-A2. Furthermore, the parities can be calculated by a generator matrix with M*(M-N) over GF(2^w) ($N + M \leq 2^w$). The PS-code for RAID-6 is a specific case for PS-N with N=2(N is the number of disks failure tolerance).

$$k_x = [r * (M-2) + [\lfloor \frac{(r-1) * gcd(M,N)}{M} \rfloor + x]\%N]\%M \quad (8)$$

Where, k_x is the disk number(column index) for the x-th row parity. Fig.10 gives an example with M=7 and N=3.

V. Performance Analysis and Experimental Results

In this section, we present the performance results obtained for the PS-code in comparison to other codes. As to workloads, which represent the case of updating multi-blocks in one request, PS-code has better degraded-mode read and write perfromance than other RAID systems. We employed DiskSim simulator[23] to measure the performances.

The previous RAID codes (HV-code, P-Code, etc.) are applied only for limited number of disks and not for any number of disks. For example, P-code can only be applied for prime-1 number of disks. Therefore, we choose 6 disks for our experiments so that we can make comparisons.

A. Performance Analysis for RAID Systems

The write operation in a RAID-6 system consists of four major operations as given below:

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

For the same write I/O request, the number of the data blocks written are the same as all the systems employing RAID codes. Therefore, it is important to note that the performance of the complete write is primarily dependent on other three operations (#1,#2 and #4). The overhead of #1 and #4 operations is determined by the number of parity written and the number of existing blocks read respectively. The effect of these operations is discussed in depth below.

In the previous works [5][6][15][8], they use the update complexity to evaluate the parity update overhead. By their definition, update complexity is the average number of parity blocks updated that are associated with a data block. According to the labels of the PS-code, we can know that two parities are associated with one data block which is optimal update complexity. However, the update complexity is not enough because in real life scenario, most of the write operations touch multiple blocks each time. Thus, we propose a metric called multi-block access complexity to reflect the real life applications.

From the arrangement of block's perspective, the order of accessing blocks for each write operation is from left to right and from top to bottom, as shown in Figure 3. In our evaluation of multi-block access complexity, we assume the location of the first block of each write operation follows a uniform distribution across all the data blocks in one matrix,

TABLE I

THE COMPARISON OF THE AVERAGE NUMBER OF XOR OPERATIONS PER PARITY UPDATE BETWEEN THE RAID CODES WITH OPTIMAL COMPUTATIONAL COMPLEXITY AND THE PS-CODE WITH DIFFERENT DISK NUMBER FOR RAID-6

Disk #	Last row of generator matrix	# of XORs for PS-code	Optimal # of XORs	PS-code/optimal
6	4-5-1-2	6.67	6	1.11
7	2-7-5-1-4	9.17	8	1.15
8	4-5-1-2-3-7	11.83	10	1.18
9	5-9-6-4-12-2-1	14.5	12	1.21
10	11-13-3-2-6-1-9-12	17.25	14	1.23
32	.....	84.67	58	1.46

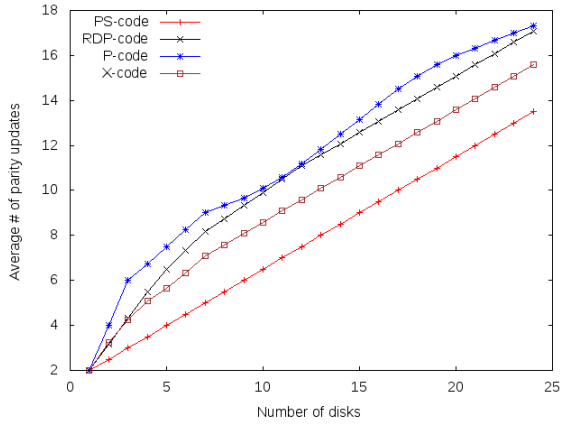

Fig. 11. Multi-blocks access complexity of parity updates for M=6. The graph for PS-code has the minimum average number of parity updates.

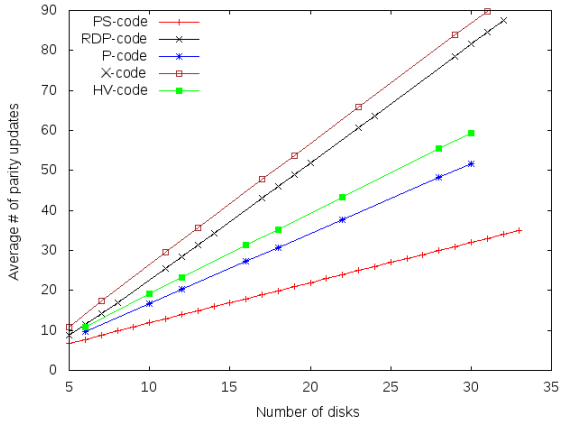

Fig. 12. Multi-block access complexity of parity updates for different numbers of disks. The graph for PS-code has the minimum average number of parity updates for different numbers of disks.

which means that the probability for each block, from 1^{st} to $M*(M-2)^{th}$, as a starting point of a write is the same. For the I/O request size, we only focus on the range from 1 block to M*(M-2) blocks because all matrix arrays share the same label layout and the read-modify-write overhead will be repeated after request size is larger than M*(M-2). For the multi-block access complexity, we calculate the average number of the reads and the parity updates that takes place for different sizes of the request made. We compute this complexity by choosing different start points (from 1^{st} block to $M*(M-2)^{th}$ block in one matrix). This ensures that the workload is distributed and the multi-block access complexity captures a real application behavior. Based on our results, shown in the Figure 11, the PS-code has the minimum number of the parity updates with varying request size for 6 disks. We also compare the average number of the parity updates for different numbers of disks. As seen in Figure 12, PS-code keeps the best number of parity updates over different numbers of disks. Additionally, the partial stripe writes also introduce reading existing blocks for computing the new parities. As seen in Figure 13, the PS-code has much lesser number of reads than others and the difference between PS-code with other codes becomes even larger with number of disks increasing. In summary, the PS-code has the best multi-block access complexity and wins at #1 and #4 operations compared to other codes.

Moreover, we made a comparison between the computational complexity(#2 operation) of PS-code and the previously

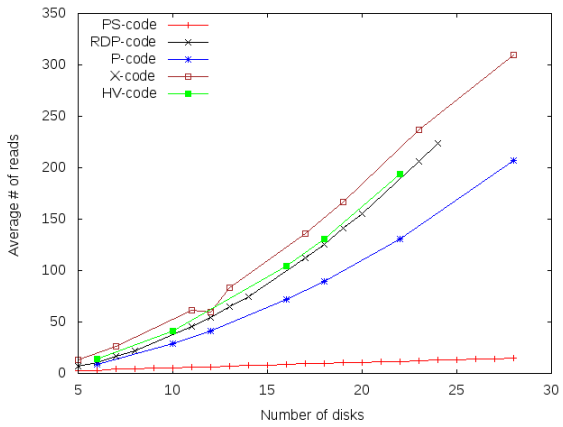

Fig. 13. Multi-block access complexity of reads for different numbers of disks. The graph for PS-code has the minimum average number of reads for all numbers of disks.

introduced codes including P-code, HV-code. These previous codes perform slightly better in terms of computational complexity because they achieve optimal computational complexity for the parity. As shown in Table I and Table II, the party computational complexity of the optimal code is approximately 1.11 times better than the PS-code for M=6. However, the PS-code has 1.24-1.47 lesser number of write operations and significantly lesser number of extra reads than other codes per write request. For the recently introduced hard

disk drive, the maximum write performance can only reach to about 200MB/s and it has even lesser write throughput for random writes. However, the bandwidth for the XOR operations can be up to 2997 MB/s, as demonstrated by Plank et al[14]. This shows that the the number of the write operations has much more significant impact on the performance than the parity computation. In conclusion, the RAID based system will benefit significantly from the reduced number of the parity write and read operations.

Lastly, in the degraded-mode, the extra reads from recovering the failed blocks performed by the system is similar to the reads generated from write operation for the parity computation. Both of them need read the part or whole rest of the available blocks in the same stripe. Thus, as explained previously, the PS-code has the lesser number of reads because of its continuous labeling. In the next section, based on the empirical data, we can demonstrate that PS-code achieves the best write performance and degraded-mode read performance amongst the previous codes.

B. RAID Performance Comparison

DiskSim[23] is used to model the codes based on the RAID-6 system. Also, we make a comparison between the PS-code and previously discussed codes. The simulation configuration has 64KB as the default block size and the total number of disks is 6. In the simulation, we set 10,000 I/O requests for sequential and random writes and one I/O request size is varied from 64KB to 2048KB. The reason 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 would have same the write performance since they have same number of data and parity blocks to write. Thus, if the request is larger than 1563KB, we can only focus on the incompletely covered parts which results in the different performance for different RAID codes.

As shown in the Figure 14 and Figure 15, the PS-code has the best write performance compared to other codes. Based on the results obtained, the PS-code improves the write performance from 66% to 74% and from 82% to 86% on average as compared to the P-code and HV-code respectively. The primary reason for the improved performance is the lesser number of parity updates and the lesser number of reads for computing new parities. Moreover, the parity computation overhead is insignificant to our code as explained in section V-A. Hence, it is not considered for the simulation results. Furthermore, the PS-code achieves significant performance gains from 66% to 82% and similar gains can be expected even if computation overhead is included to the performance metrics.

C. Degraded-mode Performance Comparison

In this section, we present the comparison between the PS-code and other codes for the read operation in the degraded mode. For this mode, the case of two disk failures is considered. In the disk failure condition, the RAID-6 system can correctly continue to read data. This is because in the degraded mode, the system can read uncorrupted data and parity blocks to recover the failed date blocks in the same stripe.

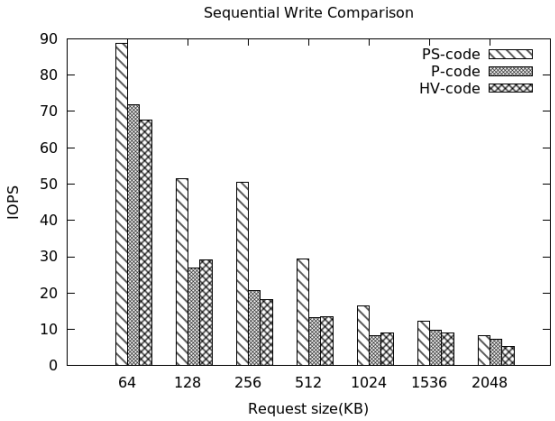

Fig. 14. Sequential write performance between RAID-6 codes with M=6 and chunk size=64KB.

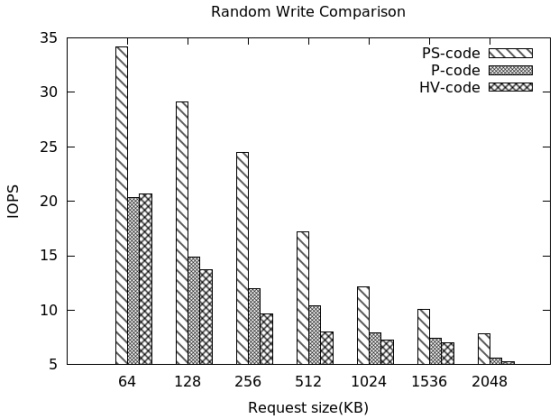

Fig. 15. Random write performance between RAID-6 codes with M=6 and chunk size=64KB.

As shown in the Figure 16 and Figure 17, in the degraded mode, the read performance of PS-code with two disk failures is 8.9% to 12.3% better than the HV-code. And the read performance of PS-code with two disk failures is 22.3% to 23.6% better than the P-code. The main reason is that the PS-code reads lesser number of data and parity blocks for recovering the failed blocks.

D. Comparison of Read Performance between Vertical RAID and Horizontal RAID

In the previous sections, the analysis and performance measurements demonstrate that the PS-code has better write performance than the other RAID-6 based codes[5][10][11]. In this section, we have attempted to make comparisons of the read performance between the PS-code and the other codes. Since the read performance is only dependent on the layout of data blocks, the codes with the same layouts have the same read performance. Therefore, PS-code (a type of vertical code) has the same read performance as other vertical codes

TABLE II
THE COMPARISON OF THE AVERAGE NUMBER OF PARITY UPDATES AND COMPUTATIONAL COMPLEXITY OVERHEAD BETWEEN THE PS-CODE AND PREVIOUS CODES FOR M=6 FOR THE WRITE REQUESTS.

	PS-code	RPD-code	HV-code	P-code
# of XORs	6.67	6	6	6
Ave # of partiy updates	7.75	11.42	10.77	9.61
Ratio(codes/PS-code) for # of partiy updates	1	1.47	1.38	1.24

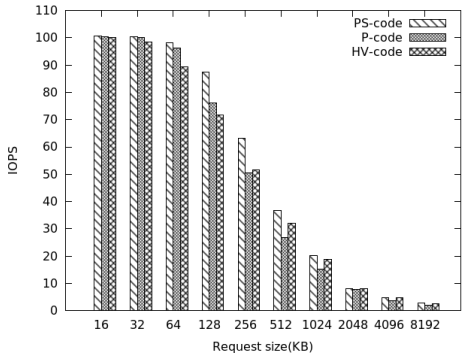

Fig. 16. Sequential read performance in the degraded mode with 2-disk failure for RAID-6 with M=6.

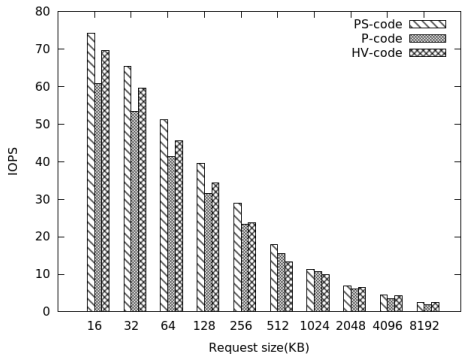

Fig. 17. Random read performance in the degraded mode with 2-disk failure for RAID-6 with M=6.

(for the same reason, horizontal codes also have similar read performances). However, a strong comparison of the read performances could be made between the PS-code and the horizontal codes. This is primarily because of the difference in layouts of the data blocks. In this comparison, the horizontal code uses the RAID-6 with Reed-Solomon, as shown in Figure 1, and the vertical RAID uses the newly devised PS-code.

As we mentioned previously the difference between the two types of codes depends on the different placement schemes of their parities. Due to the different placements, the read performance of the vertical code is better than the horizontal codes. Although the horizontal codes rotate the parity across disks to read in parallel, under certain conditions these codes have to incur the overhead because the parities are stored alongside with the data blocks. For example, with 6 disk drives, as shown in Figure 18, if the size of request is 5 blocks with address from 5th data block to 10th data block,

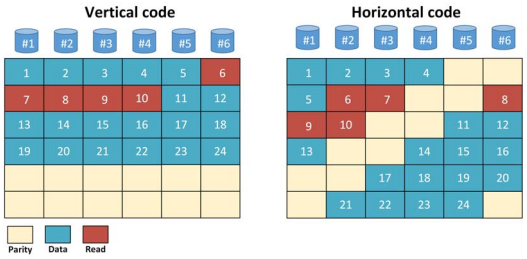

Fig. 18. An example of read performance comparison between vertical code and horizontal code with M=6.

the vertical codes can finish the request by reading at most one data block (row) for each disk drive, while for the horizontal codes, the #2 disk drive needs to read two data blocks(rows). Therefore, vertical RAID-6 would read lesser rows than the horizontal codes. Table III illustrates a comparison between the horizontal codes and vertical codes. This comparison is for the number of rows to be accessed with different request sizes. As seen in Table III, the vertical RAID-6 reads lesser or equal rows than the horizontal RAID-6.

TABLE III
THE COMPARISON BETWEEN HORIZONTAL AND VERTICAL RAID-6 FOR THE NUMBER OF ROWS TO BE ACCESSED FOR THE DIFFERENT REQUEST SIZES

request sizes (# of blocks to read)	# of rows to read		
	Horizontal	Vertical	Difference
1-4	1	1	0
5-6	2	1	1
7-8	2	2	0
9-12	3	2	1
13-16	4	3	1
17-24	4	4	0

Lastly, we give the experimental results to demonstrate a comparison between the horizontal code with the vertical codes. For the read operation, the vertical RAID systems exploit the more parallelism per stripe available than the horizontal codes, as explained in the previous paragraph. The Figure 19 presents the simulation results of the sequential read operation with different I/O request sizes. For those small sized I/O requests, a whole stripe can not be covered, therefore, horizontal and vertical RAID system have a similar performance. When the request size is increased, the vertical RAID system has better performance than the horizontal RAID system because the horizontal codes need to read more data rows. As shown in the Figure 20, the vertical RAID-6 has better random read performance for larger request sizes

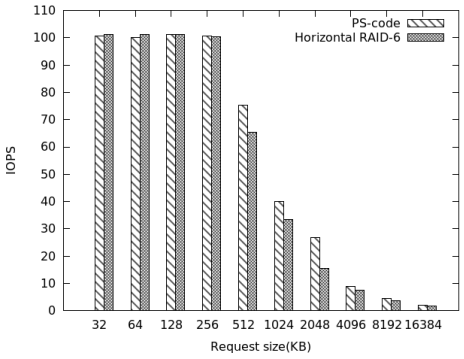

Fig. 19. Comparison of sequential read performance between the horizontal RAID-6 and PS-code RAID-6 with 8 disks.

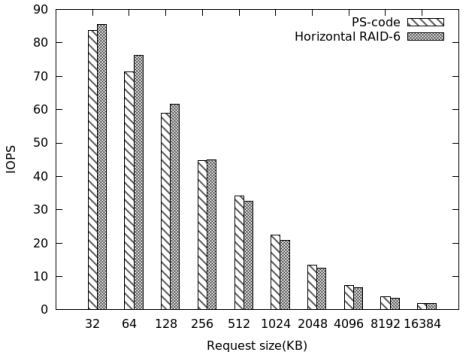

Fig. 20. Comparison of Random read performance between the horizontal RAID-6 and PS-code RAID-6 with 8 disks.

($\geq$256KB).

VI. Conclusion

We proposed a new RAID code called Partial Stripe Code (PS-code) and it focused primarily on improving the write performance and the degrade-mode read performance. With this code we observed that the number of parity updates can be reduced by using efficient labeling. Moreover, a new metric called multi-block access complexity is proposed to evaluate the performance in the real world applications. We used the multi-block access complexity and DiskSim simulation results to make comparisons between the PS-code and other codes. Because the PS-code made the least number of parity updates therefore, we were able to conclude that the PS-code has the best degraded-mode read performance and write performance amongst the previous codes. Finally, we are able to prove that the PS-code, a vertical RAID code, can significantly benefit those applications which read the data more often that writing the data.

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. 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] D. A. Patterson, G. Gibson, and R. H. Katz, *A case for redundant arrays of inexpensive disks (RAID)*. ACM, 1988, vol. 17, no. 3.

[2] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population." in *FAST*, vol. 7, 2007, pp. 17–23.

[3] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you?" in *FAST*, vol. 7, 2007, pp. 1–16.

[4] F. J. MacWilliams and N. J. A. Sloane, *The theory of error correcting codes*. Elsevier, 1977.

[5] L. Xu and J. Bruck, "X-code: Mds array codes with optimal encoding," *Information Theory, IEEE Transactions on*, vol. 45, no. 1, pp. 272–276, 1999.

[6] C. Jin, H. Jiang, D. Feng, and L. Tian, "P-code: A new raid-6 code with optimal properties," in *Proceedings of the 23rd international conference on Supercomputing*. ACM, 2009, pp. 360–369.

[7] L. Xu, V. Bohossian, J. Bruck, and D. G. Wagner, "Low-density mds codes and factors of complete graphs," *Information Theory, IEEE Transactions on*, vol. 45, no. 6, pp. 1817–1826, 1999.

[8] Y. Fu and J. Shu, "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*. IEEE, 2015, pp. 603–612.

[9] M. Blaum, J. Brady, J. Bruck, and J. Menon, "Evenodd: An efficient scheme for tolerating double disk failures in raid architectures," *Computers, IEEE Transactions on*, vol. 44, no. 2, pp. 192–202, 1995.

[10] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, 2004, pp. 1–14.

[11] Z. Shen and J. Shu, "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*. IEEE, 2014, pp. 550–561.

[12] P. Xie, J. Huang, Q. Cao, X. Qin, and C. Xie, "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*. IEEE, 2013, pp. 1–8.

[13] J. L. Hafner, "Weaver codes: Highly fault tolerant erasure codes for storage systems." in *FAST*, vol. 5, 2005, pp. 16–16.

[14] J. S. Plank and L. Xu, "Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications," in *Network Computing and Applications, 2006. NCA 2006. Fifth IEEE International Symposium on*. IEEE, 2006, pp. 173–180.

[15] C. Jin, D. Feng, H. Jiang, and L. Tian, "A comprehensive study on raid-6 codes: Horizontal vs. vertical," in *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*. IEEE, 2011, pp. 102–111.

[16] J. S. Plank, K. M. Greenan, and E. L. Miller, "Screaming fast galois field arithmetic using intel simd instructions." in *FAST*, 2013, pp. 299–306.

[17] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial & Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

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

[19] M. Luby and D. Zuckermank, "An xor-based erasure-resilient coding scheme," Tech Report, Tech. Rep., 1995.

[20] J. S. Plank, "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, 2008, pp. 85–92.

[21] C. Wu, S. Wan, X. He, Q. Cao, and C. Xie, "H-code: A hybrid mds array code to optimize partial stripe writes in raid-6," in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 2011, pp. 782–793.

[22] J. S. Plank, S. Simmerman, and C. D. Schuman, "Jerasure: A library in c/c++ facilitating erasure coding for storage applications-version 1.2," Technical Report CS-08-627, University of Tennessee, Tech. Rep., 2008.

[23] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger, "The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101)," *Parallel Data Laboratory*, p. 26, 2008.