The Reconstruction of RDP-Based RAID-6 under Double Disk Failures: An In-depth Analysis

Peizhuo Liu *

Glasgow College, University of Electronic Science and Technology of China, Chengdu, China

* Corresponding Author Email: 2021190501039@std.uestc.edu.cn

Abstract. As the data volume increases rapidly, the necessity for data storage systems is also rising significantly. The RAID technologies have been widely applied in data storage, due to their reliability and reconstruction ability under disk failures. Furthermore, as the disks in RAIDs increase, disk failures occur more frequently, arousing a considerable amount of data loss. Given this situation, this paper proposes a reconstruction algorithm for double-disk failures and an extension coding scheme which is called RDP+ based on Row-diagonal parity (RDP) to address single-disk failures which are the most common fault at a higher efficiency, providing flexible reliability against disk failures. The simulation in MATLAB provided solid proof of the effectiveness and validity of the algorithm. Additionally, the performance analysis which evaluated RDP and its counterpart in terms of encoding complexity and reconstruction cost showed that RDP+ reduces the reconstruction cost while increasing the encoding complexity.

Keywords: Row-diagonal parity, Reconstruction algorithm, Flexible reliability.

1. Introduction

1.1. Research Background and Research Significance

Under the environment of big data [1, 2], data are becoming increasingly important [3], with the rapid increase in their volume [1]. It is predicted in research conducted by IDC and Seagate that the global data volume will reach 175 ZB by 2025 [3]. As the initial process of big data analytics which is involved in daily applications, including health care and social networks [2], data storage has been attached with greater significance [3]. Thus, Data storage systems which are vital to managing and analysing data for scientific research are strongly required [3]. So far, tech giants like Amazon and Google have contributed to establishing the industry standards for storage systems [2]. Popular frames of storage systems include distributed storage systems which can be applied in IoT contexts [4, 5] and traditional disk-based storage systems [6]. The evaluation of a data storage system includes data availability, storage expense, bandwidth cost, input or output cost [7], reliability [6], and so on.

One typical frame of traditional storage systems is a redundant array of independent disks (RAID) [3, 8]. RAID merges several individual disks into a unified and expansive logical storage unit to generate data redundancy and improve storage performance [3] and reliability [8]. There are several levels of RAIDs, including RAID-0, RAID-4, and RAID-6 [8]. RAID-6 is a type of storage array which includes additional levels of parity so that it can survive two concurrent complete or partial disk failures [9]. There are wide applications of RAID-6 [10, 11], with various coding methods [10] including EVENODD, RDP, and Liberation Codes [10, 11]. Unfortunately, as the number of equipped disks [11] and the storage capacity increases, disk failures are occurring with a rising frequency [10]. If over one disk failure occurs, the system must operate at a diminished speed [10]. Furthermore, as the growth in disk capacity outpaces the rate of increase in bandwidth, the time required for disk rebuilds is extended, thus stretching the period of susceptibility when a subsequent disk failure can lead to data loss [6].

1.2. Literature Review

Currently, there is a significant amount of research on the construction algorithm of RAID-6. MDR code can achieve optimal input or output expenses for the recoveries of single disk failures while

minimizing the operational complexity [10]. However, it has twice the stripe size of other codes [10]. Another RAID-6 code named generalized X-code has been proposed in [12], by changing the positions of previous parity blocks in X-code and converting two data blocks into redundant ones. It conducts theoretically the least exclusive or operations, both in construction and updating. In addition, it can adapt to disk arrays of arbitrary sizes, compared to the restriction of being prime of the size of X-code [12]. Nevertheless, it is not an MDS (Maximum Distance Separable) code, which means that it has a lower storage efficiency [12]. [13] Mentions a comprehensive code named HV (Horizontal-Vertical) code, which achieves an even distribution of parities over the disks to balance disk input or output and has an extraordinary performance in partial stripe writes [13]. Its performances in normal encoding/decoding/update operations are also acceptable, while it requires a slightly longer time for writing [13]. Generally, there is a trade-off in different coding systems between different metrics.
The introduction presents the background of this paper and reviews the previous work in this field.

2. Encoding of Row-diagonal Parity and Its Applications in RAID-6

Table 1. An RDP array with a parameter equal to 5.

Data 0	Data 1	Data 2	Data 3	R Disk	D Disk
0	1	2	3	4	0
1	2	3	4	0	1
2	3	4	0	1	2
3	4	0	1	2	3

An RDP array is specified by a prime-number parameter p that is greater than 2 [14]. It is composed of p+1 disks, each of which contains p-1 blocks. For better illustration, a table is applied to simulate an RDP array, where p is specified as 5. As shown in Table 1, there are 6 (p+1) columns (data disks) and 4 (p-1) rows (stripes). The row parties are constructed by doing exclusive or operations over their corresponding row parity sets. A row parity set consists of the blocks on the corresponding row, except for the diagonal parity block on that row. For example, the first parity in the row parity disk is calculated by the following equation:

$$R_0 = b_{0,4} = b_{0,0} \oplus b_{0,1} \oplus b_{0,2} \oplus b_{0,3} \quad (1)$$

Where R_0 denotes the first data in the row parity disk, $b_{i,j}$ represents the element positioned at the row of index i and the column of index j. The diagonal parity in the row is not involved in the operations. The other row parities can be obtained in the same way.

In terms of the diagonal parity disk, it is composed of four parities, each of which is calculated from a diagonal set. The correspondence between parties and data blocks is illustrated in the table above. The number placed in every block denotes the diagonal parity set that it belongs to [14]. For instance, the number ‘0’ placed in the upper left block of the array indicates that it is one of the four blocks that contribute to the first parity in the diagonal parity disk. The other three blocks are also labelled with ‘0’, which is on the same diagonal as the parity block. The other parties are constructed in the constructed in the same way. After the diagonal parities are calculated and stored, the four data disks (the first p-1 columns) are encoded. It is worth noting that the data blocks labelled with ‘4’ do not contribute to any block in the diagonal parity disk, which means that an RDP array can only store the information of 4 (p-1) diagonal sets instead of all 5 (p) parity sets [14].

3. Simulation of Double Disk Failures and Reconstruction Algorithm

3.1. Simulation of Double Disk Failures

This reconstruction algorithm is inspired by an idea presented in [14] that there will always be a diagonal that only contains one erasure, regardless of the distribution of double disk failures. For convenience of explanation, only full disk failures are considered in this paper.

We consider an arbitrary double failure case, where the data on two arbitrary data disks are lost. In MATLAB, we generated two random numbers as the column numbers of the failed disk. The data stored previously in these two columns were then replaced by 'NaN' to simulate the erasure condition, as shown in Table 2 below.

Table 2. Simulation of the double failure case in MATLAB (The third and fourth columns fail).

0	1	NaN	NaN	1	0
1	1	NaN	NaN	1	0
0	0	NaN	NaN	0	1
1	0	NaN	NaN	0	0

3.2. Reconstruction Algorithm

According to the idea above, the first step in the reconstruction is to locate the special diagonal. By referring to the encoding map, which is presented below as Table 3, it can be concluded that the $i-th$ parity in the diagonal parity disk corresponds to the sum of data in the diagonal starting from the $i-th$ parity and that starting from the $i-th$ data in the first data disk.

Table 3. Illustration of diagonal parity groups of RDP in RAID-6.

D0	D1	D2	D3	D4	D5
A	B	C	D	E	A
B	C	D	E	A	B
C	D	E	A	B	C
D	E	A	B	C	D

With this conclusion, we constructed a for loop in MATLAB to count the total number of data that is not 'NaN' on the two diagonals for every. If the number equals p-2, it can be claimed that the $i-th$ group contains the diagonal we are looking for. After the diagonal was obtained and marked since it only contains one position where data is 'NaN' and the others are known as '1' or '0', the data initially stored at that block can be calculated by conducting an exclusive or operation on all the other data. Once the value was obtained, it replaced the 'NaN' placed at that position, updating the matrix.

After focusing on the diagonal parity, we turned to the row parity. In terms of the row that the calculated value was in, there was only one unknown block that contained 'NaN', which meant that it could be calculated from the corresponding row parity, the process of which was similar to that in diagonal calculations. Once this unknown value was obtained, a row recovered from double failures.

For the other rows, the procedure was almost the same. Therefore, a larger for loop was constructed to include the code block mentioned above, to ensure that the diagonal of every index was checked. Given that the reconstruction may be unordered, which means that the calculations above might not be done row by row and start from the first row, another loop was applied to ensure the full reconstruction. It used the 'isnan()' function to determine if there was still 'NaN' inside the matrix and it would not terminate until the reconstruction work was completely done. Eventually, the program was completed and run. It was proved to be valid. For any given number of p, it generated a random matrix of size $(p-1)\times(p+1)$ which contained only '0' and '1' and had two columns of 'NaN'. To prove its general validity, two tests were conducted. First, the parameter p was set to 5 and the matrix was shown in Table 4. After the execution was done, it successfully reconstructed those two columns using the known data blocks and its result is shown in Table 5 below.

Table 4. Simulation of the double failure case in MATLAB (The first and third columns fail).

NaN	1	NaN	1	0	1
NaN	0	NaN	1	0	1
NaN	0	NaN	0	1	1
NaN	1	NaN	1	0	0

Table 5. The simulation matrix after the reconstruction (p = 5).

1	1	1	1	0	1
0	0	1	1	0	1
0	0	1	0	1	1
0	1	0	1	0	0

In another case where p = 7, the randomly generated matrix is shown in Table 6, and similar results are presented in Table 7.

Table 6. Simulation of the double failure case in MATLAB (The second and fifth columns fail).

1	NaN	1	0	NaN	1	1	1
0	NaN	1	0	NaN	0	0	1
1	NaN	0	1	NaN	0	1	1
0	NaN	1	0	NaN	0	0	1
0	NaN	1	1	NaN	1	0	1
0	NaN	0	0	NaN	0	0	1

Table 7. The simulation matrix after the reconstruction (p = 7).

1	1	1	0	1	1	1	1
0	1	1	0	0	0	0	1
1	0	0	1	1	0	1	1
0	0	1	0	1	0	0	1
0	0	1	1	1	1	0	1
0	1	0	0	1	0	0	1

From the results above, it can be concluded that RDP-based RAID-6 of any size can recover from any two data disk failures through this algorithm.

4. Extension Code Based on RDP and Its Applications in Improving the Flexible Reliability of RAID-6

4.1. Overview of the Extension Code of RDP

With the development of information technology, the requirements for more storage ability have been increasing. As a major configuration of data storage systems, RAID-6 is also required to have higher and more flexible reliability [15]. Therefore, there is an increasing number of extensions of RAID-6, based on different coding algorithms. Given that our project focused mainly on the RDP algorithm, we investigated one RDP-based extension which was named RDP+.

4.2. The Practical Applications of RDP+ in Improving the Flexible Reliability of RAID-6

In terms of the motivations, firstly, in practice, the reconstruction window of a single disk failure cannot be ignored, which can lead to multiple failures during the reconstruction window. Second, the occurrence of single disk failures is significantly more frequent than that of multiple disk failures. The third point is that the current extension codes are not able to deliver flexible reliability.

To implement it, we first add one parity drive to the original RAID-6. Then we consider the optimal rebuild scheme for a single disk failure to maximize the overlapping data when accessing data disks. After that, we use short combinations of two elements involved in the optimal reconstruction to construct X parity, as shown in Figure 1.

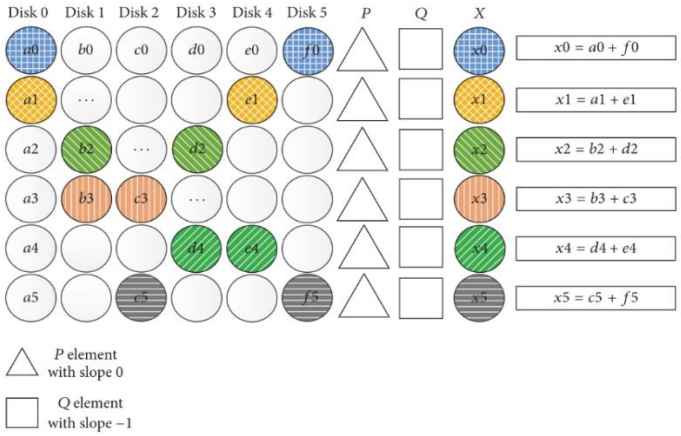


Figure 1. The implementation of RDP+ [15].

RDP+ has several features. First, it minimizes the elements needed and effectively reuses the overlapped elements. In addition, it minimises the time for reconstruction by reducing the exclusive or operations during single data failures, therefore diminishing the possibility of multiple failures. It has a better system performance compared to other extension codes and more flexible reliability than RAID-6. It also strikes a balance between reliability, performance, and practicality.

To effectively evaluate this coding scheme, we consider it from four aspects: operational complexity, update penalty, reconstruction cost, and the benefit-to-cost ratio. First, it has the least exclusive or operations per data block among common EXCLUSIVE-OR-based codes. Second, it bears the least penalty when updating data disks. It also requires fewer data reads to reconstruct a single data disk and is faster than other codes. Besides, it has the highest ratio of induced benefits to cost among common extension codes.

5. Evaluation of RDP+

5.1. Evaluation Metrics

To effectively evaluate these two coding schemes, this paper considers four aspects: encoding complexity and reconstruction cost [15].

5.2. Performance Analysis

First, for the encoding complexity, the exclusive or operations of RDP+ are composed of three parts, the exclusive or operations for constructing row parity, those for diagonal parity, and those for X parity [15]. For the first part, its number is calculated through the following expression:

$$XOR_{row} = n_{row} \times XOR_{\frac{row}{n_{row}}} = (p-1) \times (p-2) \tag{2}$$

Where XOR_{row} denotes the number of exclusive or calculations, n_{row} denotes the rows in an RDP+ array, $XOR_{\frac{row}{n_{row}}}$ denotes the number of exclusive or calculations conducted per row, and p is the parameter.

Similarly, the number of exclusive or calculations for diagonal parities can be calculated through the following equation:

$$XOR_{dia} = n_{dia} \times XOR_{\frac{dia}{n_{dia}}} = (p-1) \times (p-2) \tag{3}$$

Here XOR_{dia} denotes the number of exclusive or calculations, n_{dia} denotes the number of diagonals in an RDP+ array, $XOR_{\frac{dia}{n_{dia}}}$ denotes the number of exclusive or calculations conducted per diagonal, and p is the parameter.

The number for X parities can also be obtained by:

$$XOR_X = (p-1) \times 1 \tag{4}$$

Where XOR_X denotes the number of X parities. It equals the number of rows since X parity adopts the short combination and is calculated by combining two blocks [15].

With all three parts expressed, the number of exclusive or OPERATIONs required per data block can be obtained by:

$$\begin{aligned} C_{RDP+} &= \frac{XOR_{all}}{Blocknumber} \\ &= \frac{XOR_{row} + XOR_{dia} + XOR_X}{(p-1)(p-1)} \\ &= \frac{(p-1) \times (p-2) + (p-1) \times (p-2) + p - 1}{(p-1)(p-1)} = \frac{2p-3}{p-1} \\ &= 2 - \frac{1}{p-1} \end{aligned} \tag{5}$$

Where C_{RDP+} denotes the total number of exclusive or calculations required per data block of the RDP+ array, 'Blocknumber' refers to the total blocks, and p is the parameter.

In terms of the RDP code, its encoding complexity can be expressed similarly, which is:

$$\begin{aligned} C_{RDP} &= \frac{XOR_{all}}{Blocknumber} \\ &= \frac{XOR_{row} + XOR_{dia}}{(p-1)(p-1)} \\ &= \frac{(p-1) \times (p-2) + (p-1) \times (p-2)}{(p-1)(p-1)} = \frac{2p-4}{p-1} \\ &= 2 - \frac{2}{p-1} \end{aligned} \tag{6}$$

Where C_{RDP} denotes the total number of exclusive or calculations per data block of the RDP array, XOR_{all} denotes the total number of exclusive or calculations, and p is the parameter. Comparing these two results, the encoding complexity of RDP+ is more significant than that of the RDP.

Second, the reconstruction cost is measured by the total data read of a coding scheme. During an arbitrary disk failure, two parties in the X parity disk can recover their corresponding elements in the failed disk [15]. In addition, as mentioned in section IV, RDP+ optimizes the reconstruction process by applying the same quantity of parities from the two parity disks [15]. Therefore, the total data read for the rebuild of a single disk failure in RDP+ DR_{RDP+} can be represented by:

$$2 \times 2 + \frac{(p-1)-2}{2} \times (p-1) + \frac{(p-1)-2}{2} \times (p-1) - \frac{(p-1)-2}{2} \times \frac{(p-1)-2}{2} - 2 = 2 + \frac{(p-3)(3p-1)}{4} \tag{7}$$

Where p is the parameter. However, the total data read of RDP DR_{RDP} is$(p-1) \times (p-1)$. The result of the subtracting DR_{RDP+} from DR_{RDP}, denoted as ΔDR can be calculated by: $\frac{1}{4}p^2 + \frac{1}{2}p - \frac{7}{4}$, which can be easily proved to be larger than 0 for any prime greater than 2 and has a continuously rising trend, indicating that the RDP+ coding scheme requires increasingly less data reads than the RDP as the array size increases.

6. Conclusion and Future Work

6.1. Conclusion

We propose a reconstruction algorithm of an RDP-based RAID-6 storage system and simulate it in MATLAB. The main idea is to locate the diagonal parity set in which there is only one missing block. The simulation results show that the RDP code can effectively reconstruct any two data disks in an RDP array of arbitrary sizes. Given that the single disk failures are dominant among all types of fault modes, an extension code scheme based on RDP and called RDP+ is proposed. We add an extra parity drive to the RDP code, which is called X parity. It is constructed by combining two blocks

that are involved in the reconstruction of a single disk failure, maximizing the number of overlapped elements, and optimizing the reconstruction process. The performance analysis shows that in terms of the reconstruction cost, the RDP+ is superior to the RDP code. It means that RDP+ can maintain its basic reliability of recovering from double failures while providing flexible reliability to accelerate its reconstruction of single disk failures. However, the RDP code requires fewer exclusive or operations to construct its parities. Therefore, this extension is essentially a “tradeoff” between different aspects of the performance of a coding scheme.

6.2. Future Work

As an extension method of RAID-6, the X parity constructed by short combinations of two data blocks can effectively reduce the reconstruction window of a single disk failure, minimising the system's vulnerability. Due to the insignificance of tripe failures in practice, this method can cope with almost all fault modes, covering both single failures and double failures. In the future, this extension method can be combined with previously proposed speedup methods to further accelerate the reconstruction process.

References

[1] W. Chen *et al*, "Security vulnerability and encryption technology of computer information technology data under big data environment," *Journal of Physics. Conference Series,* vol. 1800, *(1)*, pp. 12012, 2021.

[2] A. Siddiqa, A. Karim and A. Gani, "Big data storage technologies: a survey," Frontiers of Information Technology & Electronic Engineering, vol. 18, (8), pp. 1040-1070, 2017.

[3] J. Lee et al, "Performance Evaluations of Distributed File Systems for Scientific Big Data in FUSE Environment," Electronics (Basel), vol. 10, (12), pp. 1471, 2021.

[4] Bardis, Nikolaos, Doukas, Nikolaos and Markovskyi, Oleksandr P. (2015) “Effective method to restore data in distributed data storage systems,” MILCOM 2015 - 2015 IEEE Military Communications Conference, pp. 1248–1253.

[5] J. Wu et al, "Robust and auditable distributed data storage with scalability in edge computing," Ad Hoc Networks, vol. 117, pp. 102494, 2021.

[6] Qin Xin et al. (2003) “Reliability mechanisms for very large storage systems,” 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003. (MSST 2003). Proceedings, pp. 146–156.

[7] Y. Wang and S. Li, "Research and performance evaluation of data replication technology in distributed storage systems," Computers & Mathematics with Applications (1987), vol. 51, (11), pp. 1625-1632, 2006.

[8] S. Im and D. Shin, "Flash-Aware RAID Techniques for Dependable and High-Performance Flash Memory SSD," IEEE Transactions on Computers, vol. 60, (1), pp. 80-92, 2011.

[9] A. Ma et al, "RAIDShield: Characterizing, Monitoring, and Proactively Protecting Against Disk Failures," ACM Transactions on Storage, vol. 11, (4), pp. 1-28, 2015.

[10] Y. Wang, X. Yin and X. Wang, "MDR Codes: A New Class of RAID-6 Codes with Optimal Rebuilding and Encoding," IEEE Journal on Selected Areas in Communications, vol. 32, (5), pp. 1008-1018, 2014

[11] Y. Fu et al, "Short Code: An Efficient RAID-6 MDS Code for Optimizing Degraded Reads and Partial Stripe Writes," IEEE Transactions on Computers, vol. 66, (1), pp. 127-137, 2017.

[12] X. Luo and J. Shu, "Generalized X-code: An efficient RAID-6 code for the arbitrary size of the disk array," ACM Transactions on Storage, vol. 8, (3), pp. 1-16, 2012.

[13] Z. Shen, J. Shu and Y. Fu, "HV Code: An All-Around MDS Code for RAID-6 Storage Systems," IEEE Transactions on Parallel and Distributed Systems, vol. 27, (6), pp. 1674-1686, 2016.

[14] “Row-diagonal parity for double disk failure correction | Proceedings of the 3rd USENIX conference on File and storage technologies,” *Guide Proceedings*, 2013.

[15] M. Deng et al, "RAID-6Plus: A Comprised Methodology for Extending RAID-6 Codes," Mobile Information Systems, vol. 2017, pp. 1-12, 2017.