Proceedings of the 2023 International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/31/20230140

Exploring and evaluating various data structures within RAID 6 architecture: An application-oriented study

Pengfei Yan

Henry Samueli School of Engineering, University of California, Irvine, 92697, USA

pengfy2@uci.edu

Abstract. With the development of modern technology, people have more and more chance to use large storage systems. It is important to keep the reliability of the modern storage system. There are different architectures to achieve the goal. Redundant Array of Independent Disks (RAID) architectures are one efficient way to recover the storage system from disk failures. From different RAID data structures, RAID 6 refers to use two additional parity disks to allow the users to recover from up to two disk failures. However, there are different ways to perform the RAID 6. For example, the EVENODD code uses an exclusive OR (XOR) operation to calculate parity. It has low storage requirements and simple computation. The Row Diagonal Parity (RDP) code is an upgraded version of the EVENODD code. Effectively reduce the computational consumption of parity-check encoding. On the other hand, the Reed-Solomon code has an efficient recovery algorithm and a quantitative calculation process. Plus, there are other implementation methods with their advantage and limitations for the RAID 6 architecture. To assist the application of RAID6, this paper aims to analyze, implement, and apply different RAID6 structures. The methodology of the paper is the exclusive literature review of published paper in the field in recent 10 to 20 years.

Keywords: RAID 6, EVENODD Code, Row-Diagonal Parity Code, Reed-Solomon Code.

1. Introduction

In contemporary storage systems, the technology of Redundant Array of Independent Disks is utilized to enhance failure tolerance or performance. Various RAID architectures have been developed to cater to distinct requirements [1]. RAID 0, for instance, optimizes hard disk performance by partitioning data across multiple disks. Theoretically, a RAID 0 array with n drives can operate n times faster than a single drive. This architecture is often employed in personal computers where users prioritize performance over security. In contrast, RAID 1 consists of multiple disks and preserves multiple copies of the same data on all disks to ensure fault tolerance. As long as one disk remains operational, the data remains secure.

RAID 2 and 3 introduced the concept of parity checking. However, due to early technology limitations, they couldn't handle multiple service requests and thus were infrequently utilized. RAID 4 and 5, though similar, use additional parity disks and the exclusive OR operation for error detection. Theoretically, they can recover from a single disk failure. The only difference between them is that RAID 4 stores all parity on one disk, while RAID 5 distributes them across all disks.

DOI: 10.54254/2755-2721/31/20230140

In the early days, RAID 4 and 5 were sufficient to meet people's needs for storage systems. However, with the advancement of modern computing systems, the demand for storage systems' precision and performance has increased [2]. As a response, RAID 6 was proposed to extend the RAID 5 structure, with an objective to handle dual disk failures. While RAID 0 to 5 are relatively straightforward, and their implementation methods are defined by their structure, the complexity of managing double hard drive failure left RAID 6 without a specific data structure. Therefore, the exact implementation method of RAID 6 has been a longstanding issue since its introduction. This paper aims to list, introduce, and analyze different algorithms of RAID 6, which are either widely used in the industry now or have been recently proposed with high-performance potential.

2. Typical Analysis of algorithms

2.1. EVENODD codes

The EVENODD code is a widely used way to implement RAID architectures. It was first introduced by M. Blaum to improve the standard RAID 5 architecture with a better capacity of disk failure tolerance. Since then, this data structure that uses two parity disks in RAID to handle double disk failures is defined as RAID 6 [2].

Unlike other structures such as the Reed-Solomon code. The EVENODD is purely based on the exclusive or operation (XOR), in which the same inputs result in 0 as output and different inputs result in 1 as output. The RAID 6 storage system based on EVENODD code requires a prime number of data disks and two parity disks, which means the minimum number of total disks in the system is four. In calculations, parity 1, the first parity disk, is the row-wise XOR sum of all data disks, which is similar to the parity in RAID 4 and 5 [3]. The critical part of RAID 6 is the calculation of parity 2. The parity 2 is the XOR sum of a series of data disks in one diagonal and the "missing diagonal S." To be more specific, the parity 2 in one position can be calculated with the formula:

$$a_{l,m+1} = \mathrm{S} + \sum_{t=0,t\,!=l+1}^{m-1} a_{<l-t>m,t}. \quad (1)$$

In addition, another characteristic of the EVENODD code is that the sum of parity 1 and parity 2 becomes S due to the characteristics of the XOR operation. In computing, data are often colored to separate them into blocks on a diagonal.

Table 1. Example of EVENODD Code architecture.

Disk/ Block	Data disk 0	Data disk 1	Data disk 2	Data disk 3	Data disk 4	Parity 1	Parity 2
0	1(Red)	0(Blue)	1(Green)	1(yellow)	0	1	0(Red)
1	0(Blue)	1(Green)	1(yellow)	0	0(Red)	0	0(Blue)
2	1(Green)	1(yellow)	0	0(Red)	0(Blue)	0	1(Green)
3	0(yellow)	1	0(Red)	1(Blue)	1(Green)	1	0(yellow)

As illustrated in Table 1, the colored blocks signify the color diagonals, whereas the uncolored block indicates the absent diagonal S. Consequently, the sum of parity 1 equals the total of all colored blocks and S, whereas the sum of parity 2 is the sum of all colored blocks plus four times S. Adding parity 1 and parity 2, we get S, since the nature of the XOR operation is such that identical inputs always yield 0. While the calculations under the EVENODD code are not overly complex, this architecture faces several challenges. Primarily, the number of data disks causes a degree of trouble because it must be a prime number, like 2, 3, 5, 7, 11, and so on. The predicament here is that, aside from 2, dividing a file into 3 or 5 parts necessitates the use of the division operation, which is a time-consuming operation in computer hardware architecture. This operation typically takes between 7 and 10 clock cycles. Furthermore, arranging an even number of disks physically is simpler than configuring an odd number of disks [4].

DOI: 10.54254/2755-2721/31/20230140

Moreover, the efficiency of the algorithms is restricted by the calculation of the missing diagonal S. At the time of Blaum and Bruck's work, the EVENODD code represented the most efficient method to implement RAID 6 since it only necessitated the XOR operation. However, as modern system storage evolved, user expectations regarding the performance of hard drives have risen. The EVENODD code requires the calculation of the missing diagonal S every time parity 2 is coded. This characteristic imposes a limit on its performance.

2.2. RDP codes

The row-Diagonal Parity method was first introduced in about 1994. Then, P. Corbett et al. introduce the use of RDP code in RAID 6 architecture in Proc. 3rd USENIX Conf. in 2004 [2]. With similar data structures, the RDP code improves on the EVENODD code by reducing the computational cost of writing and restoring. The number of disks in a storage system with an RPD data structure is defined by the control parameter p as shown below. First, p must be a prime number. Then, a system with controlling parameter p has p – 1 data disks and 2 parity disks, which is p + 1 disk in total. Also, the number of stripes or columns is also p – 1.

Table 2. Example of RDP code architecture.

Data Disk 0	Data Disk 1	Data Disk 2	Row Parity	Diag. Parity
0	1	2	3	0
1	2	3	0	1
2	3	0	1	2

For parity calculations, parity 1, the row parity, is also similar to parity 1 of EVENODD code or parity in RAID 4 and 5. It is the XOR sum of the data disks in one row. The parity 2, that is, the diagonal parity, calculates the XOR sum of the data disk on a diagonal and the parity 1. Table 2 above shows what the data disks and parity 1 in one diagonal look like. What's more, the blocks with the number "3" present the missing diagonal S. However, RDP code does not use the missing diagonal in encoding and decoding.

The performance of RDP code is very high for both writing and recovering. First, for writing data, since the controlling parameter p of the system is a prime number, which is almost all odd numbers, the number of data disks tends to be an even number. This fact enables RDP code to use the shift operation rather than the dividing operation to get a high writing speed. In addition, compared with the EVENODD code, the lack of the calculation of the missing diagonal S makes the computation of parity 2 simple. The calculation of parity 2 is consistent with the use of multithreading, which is an efficient way to improve the performance of computers from the perspective of hardware architectures. For example, to calculate the parities in table 2 above, one thread calculates parity 1 in the third row while the other one calculates parity 2 in the third row since the data marked with the number 2 are all available for parity 2 computation. These two threads can do that at the same time. Then, after the parities in the third row are written, two threads can calculate the parities in the second row. Since parity 2 in the second row only requires data disks and parity 1 in the third row, which is already generated in the last step. Therefore, with the use of multithreading, the performance of RAID 6 with RDP code structure is close to other algorithms with single parity such as RAID 4 and RAID 5. In a recent study [3], H. Hou et al. use specific mathematically model proving that the performance of RPD code compared with other theoretical code with high performance. They state that their efficient decoding method have the smallest complexity compared with other existing methods. In addition, they implement the RPD code with one additional parity disk, which is three parity disks in total. This study shows the potential of RDP code to be further extended in higher standard than the RAID 6 in the future [5].

DOI: 10.54254/2755-2721/31/20230140

2.3. Reed-Solomon Code

The Reed-Solomon code uses polynomial encoding and decoding that based on the Galois field to calculate parities. Its theoretical algorithm was first introduced in the 1960s by Reed, I. S., & Solomon, G [4]. Therefore, when the definition of RAID 6 was proposed, the Reed-Solomon code was the first widely used implementing method. First, it uses the Galois field to generate the coefficient of parity 2. After that, parity 2 becomes row-wise, which is similar to parity 1. Then, with such architecture, the process of decoding becomes a simple equation-solving problem with two unknown variables. From the perspective of coding, RS code should be the simplest architecture since it only needs to read data in the same row. There are three types of two-disk failure. If two parities fail, the storage system only needs to re-calculate the two parities. If one data disk and one parity disk fail, the storage system should calculate the data disk with the parity that is still working. Then, with all data disks working, the system can calculate the missing parity again [6]. If two data disks fail, we can use two parities to get two equations. Since only the data of the same row needs to be read, and the coefficients of each parity check are known at the beginning, the recovery mechanism of the RS code is efficient. In addition, E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev prove the linear arithmetic complexity of advanced RS code in *2012 ACM Subject Classification* [5]. With improved algorithms, the RS code gets better performance than it was before.

However, everything has two sides, and the RS code is not an exception. The problem with this is that Galois field-based RS codes involve more 0s and 1s than 0s and 1s compared to the XOR operation. Parity in RS codes usually requires more storage space. For similar reasons, the original data needs to be stored in Galois Field form. Two important parameters, the time complexity and the space complexity, are both limited by its property. Although nowadays, it is widely used for many years, its potential of performance is not as good as other architectures such as RDP code [7]. Therefore, in the future, the RS code needs an additional upgrade and development to keep it on the track with the performance required by modern computer systems.

2.4. RAID 6L

RAID 6L is a further upgraded version of RAID 6 with Reed-Solomon code. It was introduced by C. Jin [6]. While Ben-Sasson and Bentov also made improvements to the RS code, Jin and Feng used more aggressive structures to improve write performance. RAID 6L uses an additional log disk with a hash table to reduce the cost in writing. This hash table has several hash slots corresponding to each data block. Each slot has four different parts: logical block address, data block's original physical address, data block's current physical address, and a pointer to the next hash slot.

With the log disk, the RAID 6L has three working states: the normal state, the accelerating state, and the transitional state. In the normal state, in which the log disk is not working, the system is the same as normal RAID 6 systems. The accelerating state means the system is trying to write data without calculating parity at this state to improve its speed of coding. At this state, the data should be first searched in the hash table. If there is not a corresponding hash slot searched in the hash table, it means the system is in the consistent state. This requires a pre-read operation. Then, when data is pre-read, the log disk will save the value of the data block. In the transition state, the system will go through the log disk and calculate the parity stripe of the corresponding address.

For different cases of double disk failure, the RAID 6L can handle all of them. First, if two parity disks fail, all data disks and the log disk can provide information to recalculate parities. If two data disks fail, they can be recovered from parity for data in the consistent state and from the origin address in the log disk for data that is not in the consistent state. If one parity disk and one data disk fail, the system will try to recover the data first from another parity or the log disk. Then, the system recalculates the lost parity.

According to Jin and Feng's theoretical study of the Linux kernel, RAID 6L has significantly improved performance compared to normal RAID 6 with RS code architecture. From their report, the number of pre-read blocks and the average response time in writing have decreased up to 48% and 45%,

DOI: 10.54254/2755-2721/31/20230140

respectively. In addition, the parity logging in the following process reduces 17% and 27%, respectively. Overall, the average response time of while system has been reduced by 30%.

3. Applications

In recent years, the demand for expansive storage systems has surged significantly. With the proliferation of artificial intelligence and machine learning, vast quantities of data are generated daily, necessitating robust and capacious storage systems. Additionally, the rising establishment of databases by a myriad of organizations, universities, enterprises, and even governments illustrates this growing need. In this era of information, these entities must construct databases to bolster their online operations. For such databases, the tolerance of hard drives, a feature built into the RAID 6 architecture, is a crucial factor. Moreover, social media and entertainment companies are seeking increasingly larger storage systems to accommodate high-resolution and high-quality content like music, videos, and movies. These scenarios highlight the burgeoning market demand for high-capacity and high-performance storage systems [8].

RAID 6 offers a variety of approaches to meet specific requirements. Storage systems leveraging EVENODD codes and RS codes have been evolving for decades, exhibiting a commendable balance of stability and cost-effectiveness. Their fault tolerance safeguards the integrity of invaluable data and files, making them suitable for databases, online libraries, and cloud storage applications. On the other hand, the newer RDP code and RAID 6L code offer superior performance [9]. However, given their novelty, their cost remains high until their manufacturing scale reaches an economical level. These new codes cater to large storage systems with stringent performance requirements. For instance, companies training artificial intelligence require high storage bandwidth. The sooner their products become available, the more competitive they are in the market [10]. In conclusion, there are exceptional development prospects for large storage systems and RAID architectures.

4. Conclusion

This paper presents a comprehensive analysis of the various data structures inherent to RAID 6. Reed-Solomon code has been widely accepted and employed over several decades, given its proven reliability. RAID 6L code, which builds upon the foundation of the RS code, utilizes an additional log disk to significantly augment the performance of the RS code. Furthermore, to simplify computations, the EVENODD code has been introduced. The RDP code, an advanced version of the EVENODD code, has also been proposed.

From a theoretical standpoint, RAID 6L and RDP code exhibit high performance. Each architecture holds unique advantages and is intended for specific use cases. Theoretical research, however, doesn't imply that other structures are redundant or less valuable. Users must choose the appropriate implementation that suits their specific circumstances and requirements. Nonetheless, it's essential to note that this paper is solely based on a review of existing literature, given the constraints of real-world equipment. The author anticipates the opportunity to incorporate more empirical evidence in future studies.

References

[1] Gao S and Fang Z 2020 Multilayer Feature-Rich Satellite Network Analysis: An Application-Oriented and Time-Evolving Approach IEEE Transactions on Network Science and Engineering pp.1-1

[2] Liu Z, Cao Y, Pan L, et al. 2020 Exploring and Evaluating Attributes, Values, and Structures for Entity Alignment

[3] Rodriguez N A, Gomez A, Nava L, et al. 2018 FPGA-Based Data Storage System on NAND Flash Memory in RAID 6 Architecture for In-Line Pipeline Inspection Gauges IEEE Transactions on Computers pp.1-1

[4] Brunel-Saldias N, Ferrio J P, Elazab A, et al. 2020 Root Architecture and Functional Traits of Spring Wheat Under Contrasting Water Regimes Frontiers in plant science vol.11 581140

DOI: 10.54254/2755-2721/31/20230140

[5] Chatterjee P, Mahalingam A, Mallavaram V, et al. 2023 Systems, methods and devices for performing fast RAID re-synchronization using a RAID sandwich architecture US Patent US10229014B1
[6] Enz M and Kamath A 2023 Lock-free raid implementation in multi-queue architecture US Patent US10430336B2
[7] Anderson C R, Natera-Cordero N, Guarochico-Moreira V H, et al. 2023 Exploring room temperature spin transport under band gap opening in bilayer graphene Scientific Reports vol.13 no.1
[8] Tiwari R, Nigro A, Bondada M V 2023 Analysing Urban Form on Transit Oriented Development (TOD) Principles International Review for Spatial Planning and Sus Development vol.11 no.1 pp.141-157
[9] Likai W 2018 Digital Representation of Architecture: An Outline of Exploring Artificial Intelligence Oriented Architectural Digital History New Architecture
[10] Zhou B, Jiang H, Cao Q, et al. 2021 A-Cache: Asymmetric Buffer Cache for RAID-10 Systems under a Single-Disk Failure to Significantly Boost Availability IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems pp.1-1.