ISSN 2959-6157

Enhancing the Efficiency of Distributed Storage Systems through XOR-Optimized Cauchy Reed-Solomon Codes

Yu Fu, *

[2]School of Xiamen University, Tan Kah Kee College, Zhangzhou, China
*Corresponding author: fuyu1104227103@gmail.com

Abstract:

In today's society, we have entered the era of big data and the Internet of Things. The rise of services such as artificial intelligence, cloud storage, and the metaverse has led to increased demands for data transmission and storage. People now require not only greater capacity but also higher efficiency, more powerful hardware, and advanced algorithms. storageThe Reed-Solomon (RS) code, as a highly effective error correction code, is officially applied in fields such as data storage and data transmission. The Cauchy Reed-Solomon (CRS) Code, which is based on the RS code, utilizes a Cauchy matrix instead of the original Vandermonde matrix and replaces the original matrix multiplication with XOR operations. This paper primarily explores how the XOR-optimized Cauchy Reed-Solomon Code improves the efficiency of distributed storage systems, It was discovered that using lookup tables to store frequently used data and operations can further enhance the efficiency of storage systems and its future applications.

Keywords: XOR, Distributed Storage, Efficiency Optimization

1. Introduction

As the importance of distributed storage systems continues to grow with the development of big data and cloud computing, cloud storage allows users to store data on the cloud through the network, enabling remote access and data sharing. In big data, distributed storage systems are used to store and process large amounts of information, facilitating parallel computing and data analysis. This technology is widely utilized by companies such as Google and Amazon. In distributed storage systems, error correction codes are particularly important for ensuring data reliability. The RS code, as a traditional error correction code, is used to detect and correct multi-symbol errors. It can correct not only single-bit errors but also errors that affect multiple bits simultaneously, making it widely applied in data storage and transmission.

RS codes have strong error correction capabilities, flexible configuration, and a wide range of applications. RS codes have been in existence for nearly 70 years, yet they are still applied across various fields, demonstrating their reliability and wide applicability. However, traditional RS codes face challenges such as high computational complexity and high redundancy RS codes need a substantial level of redundancy in the sent data to achieve strong error-correcting capabilities which can lower the transmission efficiency. Fixed data length: RS codes are less adaptable than other coding systems since they require a fixed data length[1], difficulties in hardware implementation, and latency issues.

To address these problems, researchers have developed the Cauchy Reed-Solomon code (CRS). The primary focus of this paper is to investigate how CRS codes utilize XOR operations to replace traditional multiplication operations in order to improve efficiency, to demonstrate their advantages over traditional RS codes, and to explore their performance in practical applications.

2. Related work

2.1 RS code and CRS code

2.1.1 RS code

In the mid-20th century, with the development of communication technology, the demand for data transmission increased significantly. Errors in data transmission became a serious problem, especially in long-distance communication where signals were prone to interference, leading to erroneous transmissions. To address the need for effective error detection and correction, Irving S. Reed and Gustave Solomon jointly invented the RS code in 1960. RS codes are based on polynomial mathematics over finite fields, allowing data blocks to be encoded as polynomials and enhancing transmission reliability by adding redundant sym-

bols. Shortly after its invention, RS codes were applied in disk storage systems to correct data corruption caused by read/write errors. Over time, RS codes were gradually adopted by multiple international standards, becoming a core technology in digital communication and storage.[2]
RS codes are very powerful nowadays it is still the only MDS coding alternative in a large number of storage applications. The application of CRS codes is particularly widespread in modern communication systems, especially in high-reliability areas such as mobile communications, satellite communications, digital TV broadcasting, and deep space communication However, they have the disadvantage of requiring n Galois Field multiplications per coding block, and since coding blocks are typically smaller than a machine's word size, they can require 2n to 8n multiplications per machine word.Thus, RS codes are expensive. The application of CRS codes is particularly widespread in modern communication systems, especially in high-reliability areas such as mobile[3] communications, satellite communications, digital TV broadcasting, and deep-space communication
In the encoding and decoding processes of RS codes, operations such as multiplication, addition, and inversion are involved. These operations entail nonlinear processes that cannot be easily mapped onto standard integer or floating-point operations. Specialized circuits, such as finite field multipliers and inverters, are required, and the design of these circuits is quite complex. Therefore, in hardware, achieving high-speed computation often requires more hardware resources, which leads to increased chip area and power consumption. Conversely, reducing resource usage can decrease operational speed. Hence, when it comes to hardware-related aspects, the challenges posed by RS codes become even more complex.

2.1.2 CRS code

The improvements in CRS coding mainly involve two aspects. The first aspect is the use of an m*n Cauchy matrix for encoding and decoding, replacing the traditional Vandermonde matrix. Using The Cauchy matrix to replace The Vandermonde matrix simplified the matrix multiplication and inversion operations in the encoding and decoding process Since the elements of a Cauchy matrix are simple fractions, the computation process is much simpler than that of a Vandermonde matrix.
The second modification of CRS is to use projections that convert the operations over GF(2^w) into XORs. CRS codes use XOR operations to replace the original polynomial calculations, and the algorithm has been optimized. This is particularly important in cloud storage or distributed storage systems because it not only reduces data processing time but also decreases the usage of system resources, thereby enhancing the system's scalability and performance. These work as follows.
Cauchy Reed-Solomon (CRS) codes in cloud storage systems consume less bandwidth and enable faster encoding and decoding operations. By reducing the computational complexity of encoding and decoding, CRS codes can lower CPU and memory resource consumption on cloud storage servers. The special structure of the Cauchy matrix allows encoding and decoding operations to be performed with lower computational complexity. This optimization method is particularly effective when processing large-scale data, significantly reducing system computational overhead and enhancing overall system performance.
The performance of CRS codes in addressing data redundancy and fault tolerance requirements in cloud storage. A tiered storage strategy based on CRS codes can effectively reduce the bandwidth requirements and recovery time of cloud storage systems.[4]

3. Methods and Technical Models

3.1 RS code

RS code's compilation process was based on a finite field, For any q that satisfies the upper form,
the finite field GF(q)'s different structures are isomorphism. Thus, a finite field can be fully described according to its size. The basic construction of the RS code is the polynomial. Assume we have k information symbols, denote as{ $m_0, m_1, m_2, \ldots, m_{t-2}, m_{t-1}$ } each of them is the element in the GF(q). These symbols can be used to form a polynomial

$$P(x) = m_0 + m_1 x + \ldots + m_{k-2} x^{k-2} + m_{k-1} x^{k-1} \qquad (1)$$

The code word c can be formed by taking value of the polynomial P(x) on different q places in
the GF(q)

$$c = (c_0, c_1, \ldots, C_{q-1}) = (P(0), p(\alpha), \ldots, P(\alpha^{q-1})) \qquad (2)$$

The Generator Matrix of the RS code is the VanderMent matrix, every sub-matrix of the Generator Matrix is the VanderMent matrix too, and the elements differ from each other so the matrix is invertible. During the process of decoding, we can arbitrarily take k symbols of the (n,k)RS code to solve the k variables.[5]

3.2 CRS code

The Cauchy matrix offers greater flexibility, with its elements constructed from elements in a finite field, specifically over GF (2^w), where n+m≤2^w, CRS coding allows for choosing w to be as small as possible, rather than being restricted to 4, 8, or 16. Defie the sets X = $\{x_1, \ldots, x_m\}$ and Y = $\{y_1, \ldots, y_n\}$, where each xix_ixi and yiy_iyi are distinct elements of the finite field (2^w), and there is no intersection between X and Y (the elements of X and Y do not overlap). For example, if w=3, then GF(2^3) 2^3 = 8 elements. Therefore, the total number of elements in X and Y, which is n + m, cannot exceed 8. For instance, you could choose X={1,2} and Y={3,4,5}, where n+m=2+3=5. The element in the i-th row and j-th column of the Cauchy matrix defined by X and 1/($x_i + y_j$). Compared to the original Vandermonde matrix, where the elements in each row start from a specific base and are then raised to increasing powers, the elements of a Vandermonde matrix grow exponentially. Due to this exponential nature, when the size of the Vandermonde matrix is large, the computational complexity increases significantly. In contrast, the elements of a Cauchy matrix are in the form of simple fractions, which greatly reduces computational complexity and improves the efficiency of the computation.

The second method is to use XOR. As shown in Fig.1 Each element e of GF(2^w) may be represented by a 1 × w column vector of bits, V (e). This vector is equivalent to the standard binary representation of the element. Each element e of GF(2^w) may also be represented by a w × w matrix of bits, M(e), where the i-th column of M(e) is equal to the column vector V($e2^{i-1}$)

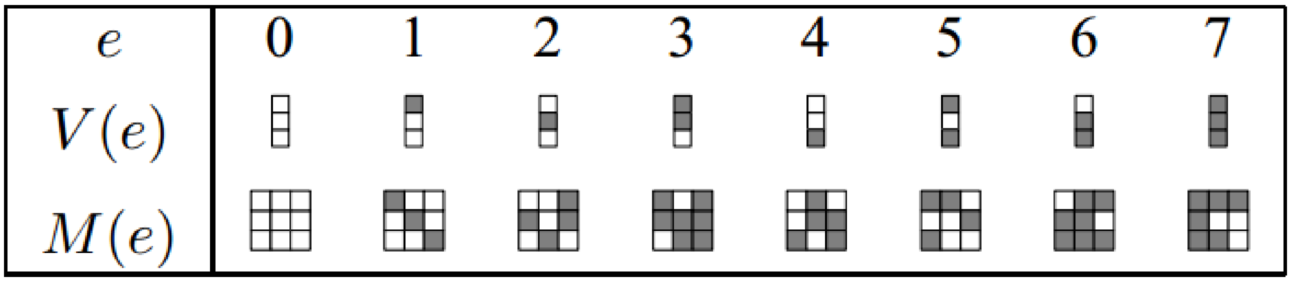

Fig. 1: Vector and matrix representation of the elements of GF(2^3)

An important property of these projections is that using standard bit arithmetic addition is XOR, multiplication is bitwise-and, $M(e_1)*V(e_2)=V(e_1e_2)$ and $M(e_1)*M(e_2)=\ M(e_1e_2)$ As shown in Fig. 2, we can obtain the multiplication answer.

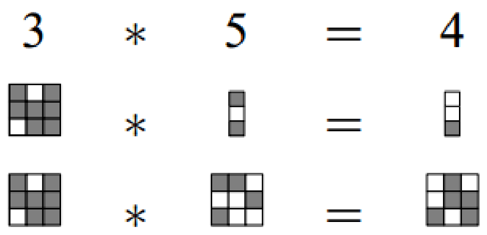

Fig. 2: Illustrative Graphic Multiplication

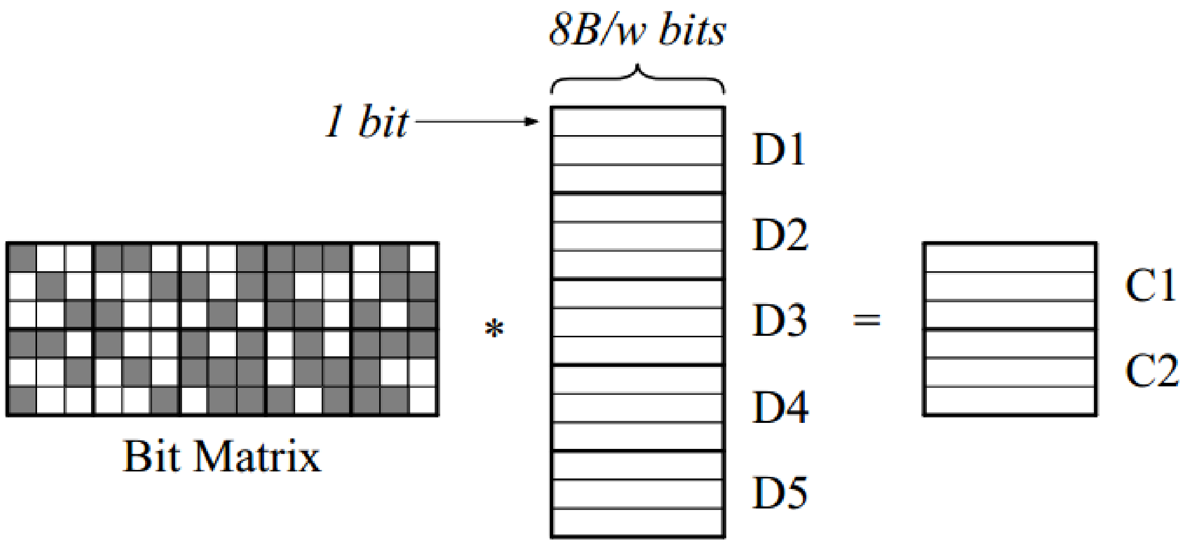

Fig. 3: The RS coding converted to use bit arithmetic

As shown in Fig.3,If we want to get C1 we just need to XOR the element

$$C_{1,1} = D_{1,1} \oplus D_{2,1} \oplus D_{2,2} \oplus D_{3,3} \oplus D_{4,1} \oplus D_{4,2} \oplus D_{4,3} \oplus D_{5,2} \quad (3)$$

Let o be the average number of ones per row in the distribution matrix. Then the number of XORs to produce a word in each coding packet is equal to o − 1.For example, in the distribution matrix of Figure 4, there are 47 ones. Since there are six rows, o =47/6, and thus the average number of XORs per coding word is o − 1 = 47/6 − 1 = 6.83. Compared to standard Reed-Solomon coding, where each coding word would require 4 XORs plus 20 multiplications over GF(2^8) [6]

3. Look up table in CRS code

A lookup table is a data structure that precomputes and stores specific operations as function results. The lookup table in digital circuits can improve the speed and efficiency of the circuit.[7]The core idea is to store common computational results in a table so that these results can be directly retrieved when needed, without the need for recalculation. This approach can break down a complex problem into a combination of modular problems, allowing you to directly utilize the necessary modules, thereby reducing the overall complexity. In the encoding and decoding process, many computational results need to be reused, such as the elements of the Cauchy matrix, which are used multiple times, as well as inversion operations. Since these operations are often repeated, storing the results in a lookup table can significantly improve speed.[8]

In a Cauchy matrix, because each data block needs to be multiplied by the corresponding row of the matrix to generate parity blocks, the data stored in the lookup table can be repeatedly accessed and used to avoid performing the same calculations each time. Additionally, during decoding, especially when multiple lost data blocks need to be recovered, the computation of inverse matrices may need to be repeated. If these complex calculations were to be repeated every time we use the data, it would consume a significant amount of time. Therefore, by storing frequently used data in a lookup table, we can reduce the need to recompute data, thereby further enhancing efficiency.

Embedded systems typically need to store data, including operating systems, application code, configuration files, log data, sensor data, and more. This data must be stored in the storage devices within the embedded system. Embedded systems often face strict resource constraints, such as limited storage capacity, power consumption, and processing power. Therefore, the storage system in an embedded system needs to be optimized to ensure efficient use of storage space and fast data access. Additionally, there are strategies for designing efficient lookup tables within embedded systems, particularly in environments where memory resources are limited.[9]

4. Experiment and Model Evaluation

Simulate the process of generating a multiplication table for Reed-Solomon (RS) codes and utilizing a lookup table using C++ code. In the two programs, the only difference is that in the first program, the lookup table function is executed at the beginning, and the results are stored in a lookup table, while in the RS code implementation, the calculations are performed every time the data is needed.

Table.1: The time required for each method and the size of the data.

Data size Method	1000000	10000000	100000000
Traditional Encoing	20616	229665	2185805
XOR Encoing	18222	185950	1777002
Look up Table	17863	185890	1653120

By using C++ to simulate the encoding process, we can obtain the data needed for encoding, as shown in Table.1. We can clearly see the time required for each method. The data(1000000) approximately corresponds to a medium-resolution JPEG image or a few seconds of standard-definition video This single modification alone makes the program noticeably faster. As the scale of data increases, the gap between lookup tables combined with CRS codes and traditional RS codes further widens. In more complex encoding processes, there are numerous repetitive operations. By pre-storing these results, we can dramatically increase our efficiency, making the entire system more effective and responsive.

In encoding there are also like Inversion Operations, Matrix Computations, and In decoding it can significantly accelerate error detection and correction during the decoding process. Using lookup tables for these operations not only reduces real-time computation complexity but also enhances the overall speed of encoding and decoding. It transforms complex problems into simpler steps that can be handled individually. This modular approach allows for more efficient processing and quicker resolution of com-

plex tasks.
Although a lookup table can improve computational efficiency, it requires additional storage space. We need to trade off more storage space for better time efficiency and consider the relationship between the implementation complexity and performance improvement of CRS codes. However, whether it significantly increases the size of the data depends on the design and implementation of the lookup table. Typically, a lookup table should only contain the results that are frequently used during the encoding and decoding processes, rather than all possible operation results. The lookup table should prioritize storing the most commonly used data, reducing the need for real-time calculations of less frequently used data. In certain high-efficiency scenarios, such as cloud storage or streaming media, faster storage is required to ensure the proper functioning of the system. In environments that involve processing large-scale data or require efficient encoding and decoding, the advantages of a lookup table often outweigh its storage space overhead, thereby achieving the desired efficiency.
In addition to CRS, there are several optimization techniques for RS codes, such as iterative algebraic geometry algorithms (AG codes). These methods can correct errors beyond the traditional error correction limits within polynomial time, thereby enhancing the practical effectiveness of RS codes.[10]

5. Conclusion

The study found that CRS codes, compared to traditional RS codes, offer higher efficiency by reducing computational complexity in the encoding and decoding processes. This is achieved through the use of XOR operations and optimized matrix structures, which minimize computational overhead, making CRS codes particularly suitable for large-scale storage solutions.
However, CRS still involves redundant operations like multiplication and finding inverses. By storing these results in lookup tables, the process becomes more efficient, although this increases data size. The development of the metaverse and blockchain demands more efficient storage systems, where sacrificing space for time efficiency is often worthwhile.
The development of the metaverse and blockchain has imposed higher demands on storage systems, particularly in terms of data security, privacy, decentralization, and large-scale storage. Cloud storage and other network services also require more efficient storage solutions. When high efficiency is necessary, the benefits of sacrificing space for time efficiency can be significant.

References

[1] Plank, J. S. (2005). *Optimizing Cauchy Reed-Solomon codes for fault-tolerant storage applications* (Technical Report CS-05-569). Department of Computer Science, University of Tennessee.
[2] Wicker, S. B., & Bhargava, V. K. (1999). Reed-Solomon Codes and Their Applications. *John Wiley & Sons*.
[3] Lin, S., & Costello, D. J. (2004). Parallel Architectures for Fast Reed-Solomon Encoding and Decoding. *IEEE Transactions on Computers*, 53(4), 487-495.
[4] Li, J., & Tang, Z. (2011). Cauchy Reed-Solomon Coding for Cloud Storage Applications. *IEEE Transactions on Cloud Computing*, 9(2), 345-355.
[6] Plank, J. S. (2005). *Optimizing Cauchy Reed-Solomon codes for fault-tolerant storage applications* (Technical Report CS-05-569). Department of Computer Science, University of Tennessee.
[5] Du, X. (2023, July 14). *Performance analysis of several common error correction codes*. Paper presented at The 5th International Conference on Computing and Data Science (CONF-CDS 2023), Macau, China.
[7] Smith, R. J., & Brown, T. E. (2013). Using lookup tables in digital circuits: Benefits and limitations. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(6), 1473-1485.
[8] Cheng, F., & Wu, J. (2017). Power-efficient lookup table designs for energy-constrained systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10), 2910-2919.
[9] Nguyen, H., & Zhao, L. (2015). Memory-efficient lookup table designs for embedded systems. *Embedded Systems and Applications Journal*, 7(4), 210-221.
[10] Guruswami, V., & Sudan, M. (1999). Efficient Decoding of Reed-Solomon Codes Beyond the Error-Correction Bound. *IEEE Transactions on Information Theory*, 45(6), 1757-1767.