Proceedings of the 3rd International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/6/20230871

Comparative analysis of Reed- Solomon code and liberation code for two-disk failure recovery in RAID-6

Zichao Cao

Woodhouse, Leeds LS2 9JT, UK

el20zc@leeds.ac.uk

Abstract. As the quantity of data stored in the cloud system is increasing enormously daily requirement for a high tolerance for erasures is stricter than ever before. There are many encoding methods created by engineers to help recover the erased data, and one of the codes that are constantly used in RAID6 store system is the Reed-Solomon code. Though nowadays Reed-Solomon code is applied widely, there are quite a lot of restrictions for it in some specific conditions. Under those scenarios, one more advanced encoding technique, Liberation code is introduced as an alternative coding for Reed-Solomon code. There will be a comparison between the Reed-Solomon code and the Liberation code from two aspects.

Keywords: The Paper Reed-Solomon Code, Liberation Code, Performance Comparison.

1. Introduction

For more than twenty years, disc striping techniques have been utilized to decrease data loss due to disc failure while enhancing speed [1,2]. To protect the disc from the failure of two discs, the RAID -6 approach is frequently utilized.

The rate of double disk failures has increased due to the sharp increase in disk sizes, the comparatively slower development in disk bandwidth, the creation of disk arrays with more disks, and the adoption of less dependable and less efficient disk types. To provide acceptable data integrity, it is necessary to apply algorithms that can guard against double disk failures. To defend against two disk failures, algorithms that adhere to the Singleton bound of information theory [3] only add two more disks to the number of disks necessary to hold the unprotected data. Good algorithms satisfy this constraint and store the data in an unencoded form, allowing for direct disk reading [4].

The Liberation Codes are essential because they offer performance that is ideal or very close to optimal in all stages of coding [5]. When it comes to the modification overhead, and frequently also in terms of encoding performance, they surpass all other RAID-6 codes. And Reed-Solomon code is now widely used in disk striping techniques a potent category of nonbinary block codes that are very helpful for repairing burst mistakes.

A study by Irving Reed and Gus Solomon appeared in the Journal of the Applied Mathematics Society of America [6]. This paper presented a novel code, currently known as Reed-Solomon (R-S) codes, this category of error-correcting codes. These scripts are extremely useful and powerful, and they are now included in numerous applications for everything like CD players. This piece is trying to outline the key characteristics of RS code and the basics of their operation [7].

Applications of RAID-6 fault tolerance have proliferated as storage systems have gotten bigger and more complicated. For storage systems with numerous storage devices that can withstand the failure of any two components, RAID-6 is a specification. RAID-6 has gained significance in recent years when a disc drive failure happens simultaneously when on another drive, the latent failure of a block is included. This set of failures results in the loss of all data on a typical RAID-5 system. Designers of storage systems have therefore begun using RAID-6.

R-S codes have a unique appeal since coding effectiveness rises with code length [8]. Reed-Solomon Codes can be designed with lengthy block lengths (measured in bits) and need less decoding time than other codes of a comparable length. This is because symbol-based arithmetic, rather than bit-based arithmetic, is used by the decoder logic. Therefore, all arithmetic operations for 8-bit symbols would occur at the byte level [7]. Comparing this to binary codes of the same length, the reasoning becomes more sophisticated, but the throughput also rises. This paper compares two codes for RAID-6, separately, they are the Reed-Solomon code and the Liberation code.

2. Description of Liberation code

Based on a bit matrix-vector product, liberation coding and decoding are quite similar to the Cauchy Reed-Solomon coding [9] and Reed-Solomon coding [10] techniques. This product clearly outlines the steps involved in encoding and modification. Decoding is more difficult; thus, we must add the idea of "bit matrix scheduling" to the bit matrix-vector product to proceed with efficiency. After outlining the fundamentals of bit matrix coding, it is defined that the liberation codes and talk about how well they work when being encoded or modified. Then, we discuss decoding and how bit matrix scheduling can enhance its performance. In Section 4, we contrast the Liberation Codes with the other RAID-6 codes.

2.1. The summary of Bit Matrix Coding

Cauchy Reed-Solomon coding introduced the parity array coding method known as bit matrix coding [9] We can see typically k numbers of equipment and m number of equipment that can store bits with w number perfectly. The scheme performs encoding using a w*(k+m) rows and w*k columns matrix in Galois Field of 2. Because of this, the calculation equals modulo 2, and each component of the matrix is zero or one. The matrix is known as a BDM, or binary distribution matrix. The matrix vector product shown in Figure 1 represents the state of a bit matrix coding system [5].

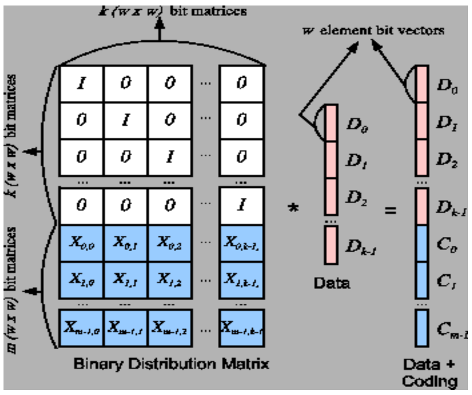

Figure 1. illustration for a scheme of binary matrix.

We can see an exact arrangement in BDM scheme. The initial w*k rows make up a w*k rows and w*k columns identity matrix, as k rows and k columns matrix made up of individual w rows with w columns binary matrices. The following m*w rows' m*k matrices, Xi,j, are all w by w bit matrices [5].
It is added a vector made up of the wk bits of data to the BDM. We represent that as k bit vectors with w elements each. The (k+m)w bits are contained in the product vector. The data vector is represented by the first wk elements, while the coding bits stored in the m coding devices are represented by the last wm elements [5].

Each device represents a row of matrices in the BDM, and each bit of each device corresponds to one of the w(k+m) rows of the BDM. By dividing the row of the bit by the information, every bit of each Ci is encoded. Since every component in the scheme. is in form of bit, the dot product should be considered as the xor calculation of every data bit in which there is a one is included. As a result, the function of encodings is directly impacted by the amount of 1s.

2.2. The Working Principle of Liberation Code

The Liberation codes are now defined. Number of w is constrained and is dependent on k, same as with EVENODD and RDP coding. In specifically, w needs to fall between k and 2, and be a prime number. We employ two different forms of notation to describe the Xi matrices:

- It is defined that $I^w_{\to j}$ is the identity matrix with w columns with w rows which is rotated to the right with j columns. Note that I^w=I_(➝0)^w
- It is defined that $O^w_{i,j}$to be a w by w matrix where every element is zero, except for the element in row (i mod w) and column (j mod w), which equals one.

The Liberation code are defined as:

- $X_0 = I^w$.
- For $0 < i < k,\ X_i = I^w_{\to i} + O^w_{y,y+i-1}$, where $y = \frac{i(w-1)}{2}$. Another way of the representation is that $y = \frac{w-i}{2}$ if i is an odd umber, and $y = w - \frac{i}{2}$ if i is an even number.

The total number of the one which is kw+k-1 in the Xi matrices relies on the values of k and w. When combined with the k*w 1s for P's matrices, this results in CDM has 2kw+k-11s. If the row of the CDM for a given bit contains o ones, then encoding a coding bit from the data bits needs (o-1) XORs. The average number of XORs needed for each coding bit is therefore $\frac{2kw+k-1-2w}{2w} = k - 1 + \frac{k-1}{2w}$. K-1 is ideal.

The CDM's average number of ones per column is$\frac{2kw+k-1}{kw} = 2 + \frac{k-1}{kw}$, or around two. Two is ideal. Thus, both for encoding and alteration, the Liberation codes operate close to optimally.

2.3. Overview of Bit Matrix Scheduling

Considering a scenario where k = 5 and w = 5 to illustrate the requirement for a bit matrix schedule. D0 and D1 fail when we encode using the Liberation code. By erasing the first 10 rows of the BDM and flipping it, we create BDM', which we then decode. We may use the additional components to rebuild D0 and D1 by using the top 10 lines of this inverted matrix.

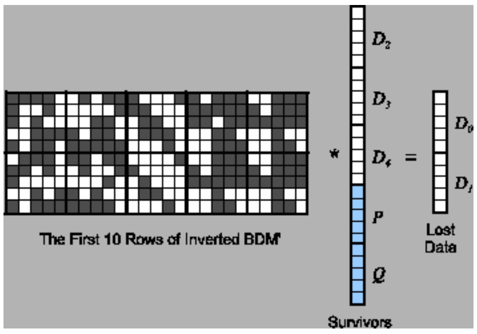

Figure 2. Decoding processes when k=5and w=5.

Since there are 134 ones in the matrix, computing the dot products in the simple approach requires 124 xors. 40 xors would be required for ideal decoding. Consider the matrix's row 0 and row 5, which are utilized to determine D (0,0) and D (1,0). Since line 0 and line5 both contain 16 ones, the simple method of calculating d0, 0 and d1, 0 requires 28 XORs. There are 13 columns, though, where both rows have

a single value. So, let's calculate d1,0 first, which requires 13 xors, and D (0,0) with the following equation: D (0,0) = D (1,0) ⊕ D (2,0) ⊕ D (3,0) ⊕ D (4,0) ⊕ p_0.
The number of xor operations need to combine the two bits is reduced from 28 to 17 with just four extra operations.

We created a simple method that we term BMS to do some sets of dot products in a bit vector product faster than by functioning every product individually. The algorithm is presented using the following assumptions and terminology.:

- The product vector U is calculated by multiplying matrix M by vector V. Every element is a bit, and GF is used for arithmetic (2).
- r rows and c columns in matrix M. M[i,j] represents for the components in a row I column j. As opposed to vector p, which has r elements indicated U[0]... U[r-1], vector V has c components designated V[0]... V[c-1].
- The row i of M is written as Mi.
- The vector Form has r number of integers and every elements is started with -1.
- The vector Ones has r number of integers, and started with the same amount of 1s in line i.
- Vector Sum with i row and j columns has a c-component bit, which is the same as the sum in Galois Filed of 2 of Mi and Mj.
- It is defined that notd is a integer group that includes numbers from 0 to r-1.

The algorithm advances via r stages. Every step carries out the next actions.:

1. Choose value of i which is included in notd and the vector Ones is simplified.
2. When U[i] is determined to become the xor in V[j] in order that M[i,j] equals 1 while From equals -1. When U[i] is computed as the xor in U[From] and all V[j] such that M[i,j] + M[From ,j] = 1 and From is not 1, then From[j] is not equal to 1..
3. Take I away from Notdone.
4. When j is included in notd, determine x as one added by the number of ones in Sum. When x is smaller than Ones, assign From to i and Ones to x.

When producing the product component from a different particular product rather than the original vector is more cost-effective, it is possible to do so using this method. The method produces the following schedule when applied to the example in Figure 2 [5].

- To compute d (1,3) 7 xors are needed.
- To compute d (0,3) 4 xors are needed.
- To compute d (1,4) 5xors are needed.
- To compute d (0,4) 4 xors are needed.
- To compute d (1,0) 5 xors are needed.
- To compute d (0,0) 4 xors are needed.
- To compute d (1,1) 4 xors are needed.
- To compute d (0,1) 4 xors are needed.
- To compute d (1,2) 5 xors are needed.

  To compute d (0,2) 4 xors are needed.

The total number is 46 xors, which is smaller than 124 when no schedule is applied. A decent decoding method would require 40 XORs.

It is feasible to use this technique to calculate the product element if doing so is more effective than utilizing the initial vector.

But when decoding utilizing the Liberation Codes, this method works much better. A fascinating side effect of the method is that it prompts the efficiency of encoding to be better and decoding using Cauchy RS codes but also creates the best scheduling for encodings and decoding with RDP code in the form of the bit matrix style. Finding an effective algorithm that generates ideal schedules for all bit matrix-vector products is still an unresolved problem.

3. Description of Reed-Solomon code

3.1. Working principle of Reed-Solomon Code

The RS Code was created by Living S. Reed and Gustave Solomon in 1960 and is used for data transfer. This code is based on block coding. To transmit data, a block of k information bits must first be converted into a codeword (n>k), which is a block of n bits. Two thousand codewords are conceivable since there are k bits in a block. RS codes are used for digital storage and communication. This code has uses in digital television, wireless communication, storage media, and other areas. Below is an illustration of RS code: - For any n and k, there are RS (n, k) codes on m-bit symbols in which [6]:

$$0 < k < n < 2^m + 2 \tag{1}$$

he length of a code word inside an encoded block is indicated by the number n, the amount of message bits to be encoded by the number k, and the number of bits per symbol by the number m. Thus, RS (n,k) may be written as follows.:

$$(n, k) = (2^{m-1}, 2^{m-1} - 1 - 2t) \tag{2}$$

(n-k) = 2t, here t is the number of mistakes fixed by the RS code, determines the amount of parity bits inserted to the message bits. The RS code's distance is determined by:

$$d_{min} = n - k + 1 \tag{3}$$

As a result, RS code's minimum distance is comparable to hamming distance. The property of a finite field is that operations on its constituent components always produce results that are contained inside the field, according to Galoi’s field, on which the Reed-Solomon code is built.

For the situation about RS code, no more than 2t parity symbols are needed to rectify t symbol faults. Below equation supports the following logical inference. The number of redundant symbols the decoder must "spend" is n - k, or twice as many superfluous symbols as there are correctable errors. Each mistake is represented by a superfluous sign, and its appropriate value is represented by a second superfluous sign [6].

$$t = \left\lceil \frac{d_{min} - 1}{2} \right\rceil = \left\lceil \frac{n - k}{2} \right\rceil \tag{4}$$

Below equation shows the capability for RS code to recover from erasures:

$$\rho = d_{min} - 1 = n - k \tag{5}$$

Any linear code may correct n-k sign erasures when the n – k number of erased signs all to be on the parity symbols. But RS codes use the special ability to undo any group of n–k symbol erasures. RS codes are created with types of redundancy. However, difficulty of a fast operation escalates with RS code with 3 redundancies. As a result, RS codes with the highest coding rates are the most appealing.

The relation between the number of errors corrected and the erasure corrected is expressed below:

$$2\alpha + \gamma < d_{min} < n - k \tag{6}$$

here n is the total amount of signs that can be rectified for symbol errors and n is the sum number of signs that can be recovered for symbol erasures. The comparison that follows shows a benefit of nonbinary codes like a RS code. Through the following example comparison, the advantage of RS code as one of the nonbinary codes can been seen. The example happens under the case for which (n,k)=(7,3). For binary code, there are $2^7 = 128$ n-tuples, and 2^3codewords is included. For nonbinary code with (n,k)=(7,3) and 3 bits, there are $2^9 = 512$ codewords. Only a small portion of all conceivable n-tuples when working with bits, which consists of m bits, are codewords. With rising m values, this fraction falls. The key insight here is that a big d_{min}can be produced when the n-tuple space is seldom ever utilized for codewords.

3.2. Working principle of RS code

S The message to be conveyed, if a finite field of q components is selected, is made up of k components of the GF(2m) that are given by:

$$f = (f_0, f_1, \dots . f_k) \tag{7}$$

Then, through multiplying the factor of message and proper x we can calculate the polynomial:

$$F(x) = \left(f_0 + f_{1x} + \cdots .. + f_{k-1}x^{k-1}\right) \tag{8}$$

Rest of the polynomial is called parity check polynomial:

$$B(x) = b_0 + b_{1x} + \cdots b_{2t-1}x^{2t-1} \tag{9}$$

By adding the two equations together, the codeword is:

$$V(x) = F(x) + B(x) \tag{10}$$

For the decoding process of RS code, there are numerous problems that can arise during message transmission, such as the sent message being tampered with as a result of noise hence the message received at the receiver's end is R(x), denoted by the following expression:

$$E(x) = e_{n-1}x^{n-1} + \cdots + e_1 x + e_0 \tag{11}$$

With RS code, the amount of error that can be recovered is (n-k)/2. When the error is outnumbered of it, the code is invalid.

4. Comparison between Reed-Solomon code and Liberation code

4.1. The number of XOR

From the encoding perspective, the comparison is conducted by counting the number of xor operation required per coding word. The comparison result is from the paper by James S. Plank [5]. The P and Q devices are both encoded in this. Since k-1number of xors per coding word constitute ideal encoding, It can be adjusted if we divide the amount of xors by k-1 to obtain the codes' overhead or the number that makes encoding ability subpar. Low values are thus preferred, with one being the ideal value. It can be seen in the Figure3:

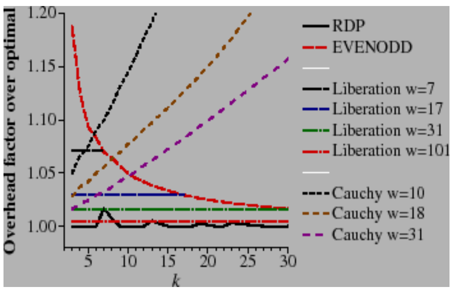

Figure 3. Encoding performance of liberation code and Cauchy RS code.

This is not the case with Liberation codes; greater w outperforms smaller w. We exhibit four w values as a result in Figure 3. The reason why line is even is that the amount of xors is equal to k-1+(k-1)/2wand as a result, their factor above optimum is 1+1/2wThe codes are therefore asymptotically stable if the situation is ideal as w→∞. ideal as in w. However, in practice, Because of the impact of memory and caching, lower w could function better than larger w, since they need the coding engine to hold less data in memory. Thus, the choice of an appropriate w in Liberation Coding entails a compromise between the smaller number of XORs needed by big w number and the smaller memory requirement of little w number. The graph includes the Cauchy Reed-Solomon codes for varied w. Cauchy RS code function more excellent with bigger w number than that with smaller w, like Liberation Codes.

4.2. Different device

Figure 4 shows measurements of the fundamental RS coding processes performed on 3distinct computers. The first device is a MacBook Pro with an Intel Core 2 Duo processor clocked at 2.16 GHz. The second computer has a 1.5 GHz Intel Pentium processor and is a Dell Precision. The third is a Toshiba Tecra with an Intel Pentium CPU clocked at 1.73 GHz. We calculate the bandwidth of the 3 types of procedure xor, multiplication by a random value in Galois Filed of 28 and multiplication by 2 using Anvin's optimization [10] for each. A 256 by 256 multiplication table is specifically used to implement multiplication by any arbitrary constant [5].

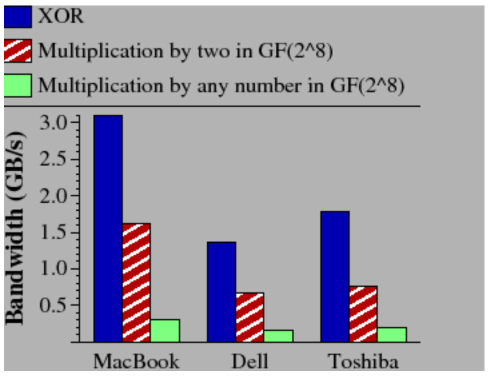

Figure 4. The bandwidth for RS code.

Following are some predictions for how well maximized RS coding will function. Make $B_{\otimes}$ be the XOR bandwidth, be the arbitrary multiplication bandwidth, and $B_{\otimes}$ be the two-fold multiplication bandwidth. Using conventional Reed-Solomon coding, one gigabyte of data can be encoded in the following time on k devices:$2 * \frac{k-1}{B_{\otimes}} + \frac{k-1}{B_{\otimes}}$.

The reason is the number of which is Q needs (k-1) multiplications by a value, whereas the P device still uses parity encoding. The time required to encode one-megabyte using Anvin's optimization is calculated by replacing $B_{\otimes}$ with $B_{\otimes 2}$. The ideal time for encoding, which accounts for k-1 XORs each coding word, is $2 * \frac{k-1}{B_{\otimes}}$.

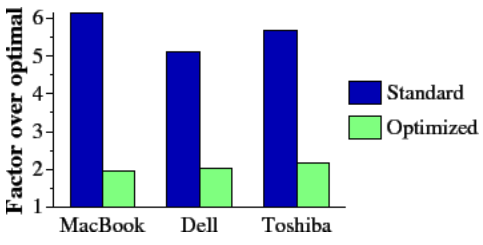

Figure 5. Performance of RS code from bandwidth measurements.

The performance of decoding for typical Reed-Solomon coding is like that of encoding, as shown in Figure 5. Anvin's optimization increases encoding performance by around 3 times. It is far worse than xor-based codes, though. Additionally, decoding is not affected by the optimization and will operate at the same speed as conventional RS code. As a result, we draw the conclusion that RS code is a poor choice for RAID-6 implementation, even with optimization.

5. Conclusion

Reed-Solomon code is a good coding scheme when there is no strict requirement for speed and bandwidth saving and the storage is adequate since it is MDS code and can encode most of the RAID6 system. When the situation is stricter, like high demand for operation speed, tight storage, we need some coding that is more sophisticated like Liberation code to help save bandwidth and space while increasing operation speed. It is proved that RS code is a code method with highly efficiency. It entails dividing the information w-bit, and then using a specific type of arithmetic known as Galois Field arithmetic, calculating the i_{th}word on the Q device with the i_{th} word on each data device. Multiplication is far more challenging and requires one of several implementation strategies, while GF addition is equal to the xor procedure. As a result, Reed-Solomon Coding is more expensive than the other methods and the also slow than Liberation code. For the shortage of the RS code, in some certain cases, liberation code can be used to improve the performance. The liberation code has an approach for scheduling the xor procedures of a bit vector product, which enables fast decoding. The decoding overhead is reduced by a factor of six when w=17 and by a factor of over eleven when w=31 using the simple, ineffective technique for most bit matrices. And from the performance comparison with RS code, we can see that it surpasses the RS code in all the conditions. But when the word length is in certain numbers, other technique can surpass the liberation code. Overall, the Liberation code is a good alternative for Reed-Solomon code.

References

[1] Salem K, and Garcia-Molina H 1986 Disk striping, *Proceedings of the 2nd Internation Conference on Data Engineering* pp336-342.
[2] Patterson D, Gibson G, and Katz R 1988 A case for redundant arrays of inexpensive disks (RAID) In *Proceedings of the ACM SIGMOD International Conference on Management of Data* pp 109-116.
[3] MacWilliams F and Sloane J 1977 *The Theory of Error-Corrrecting Codes* (North-Holland)
[4] Corbett P, English B, Goel A, Grcanac T, Kleiman S, Leong J, & Sankar S 2004 Row-diagonal parity for double disk failure correction *In Proceedings of the 3rd USENIX Conference on File and Storage Technologies* pp. 1-14.
[5] Plank J S 2009. The raid-6 liber8tion code. *The International Journal of High Performance Computing Applications,* 23(3), 242-251.
[6] Reed I S and Solomon G 1960 Polynomial Codes Over Certain Finite Fields *SIAM Journal of Applied Math* vol. 8 pp. 300-304.
[7] Sklar D and Bernard B 2001 Reed-solomon codes Downloaded from URL http://www. informit. com/content/images/art. sub.--sklar7. sub.--reed-solomo-n/elementLinks/art. sub.--sklar7. sub.--reed-solomon. pdf: 1-33.
[8] Wicker S B and Bhargava V K, ed., 1983 Reed-Solomon Codes and Their Applications Piscataway*, NJ: IEEE Press*.
[9] Blomer J, Kalfane M, Karpinski M, Karp R, Luby M and Zuckerman D 1995 An XOR-Based Erasure-Resilient Coding Scheme *Technical Report* TR-95-048 (International Computer Science Institute).
[10] MacWilliams F J and Sloane N J A 1977 The Theory of Error-Correcting Codes Part I North-Holland Publishing Company (Amsterdam, New York, Oxford).