

Research on the equivalence principle and module design of erasure code based on multi tenants

Jing Sun*

Department of Intelligent Science and Information Law, East China University of Political Science and Law, Shanghai 201620, China

ABSTRACT

Erasure coding technology can reduce costs, improve data reliability, and enhance the usefulness of the system. This technology has been widely designed and used in distributed storage systems. However, in the case of multi-tenancy, none of the popular distributed storage systems support user-level flexible and configurable encoding modules. To solve the problem that the cost and reliability cannot be controlled by custom encoding parameters, the proof of equivalence principle under different code construction schemes is provided, and then the erasure encoding module under multi-tenancy is designed. This module allows a flexible selection of stripe ratio parameters and code algorithms, and data conversion under different algorithms. Based on the generator polynomial, the code library is provided which is optimized by avx2 instruction sets. Compared with the open-source generator matrix-based code solution, the library can resist silent errors. Finally, the analysis and evaluation of the throughput performance of the three code libraries that support avx2 instruction sets is provided. The experiment results show that ISA-L library could maintain high encoding throughput with the growth of the size of the stripe. The design of the flexible erasure code module under multi-tenancy environment has filled the gap of the distributed storage system in the refined control of cost and reliability.

Keywords: Reed-Solomon codes, multi-tenant, distributed storage system, erasure coding, elastic code

1. INTRODUCTION

In order to ensure that data in distributed systems is not lost in the event of various faults, a data redundancy mechanism is usually adopted to ensure its reliability. The implementation of the redundancy mechanism can be divided into two categories: one is replica, and the other is an erasure code algorithm. The replication method is simple and easy to implement, but high redundancy will bring cost overhead, and in order to improve system reliability, it can only be achieved by increasing the number of replicas. Compared to the replica mechanism, deletion correction codes can ensure reliability while reducing costs. It is not necessary to rely heavily on the number of stacked storage hardware, but only to adjust the ratio parameters of the erasure code.

In recent years, the erasure code algorithm has been widely used in distributed storage systems^{1,2}. Under the same redundancy, Reed Solomon³ (RS) codes have some natural advantages and the highest reliability. They belong to the maximum distance separable code, are easy to implement at the system design level, have mature encoding and decoding algorithms, and are easy to optimize at the hardware level. Therefore, RS code has been widely used in distributed storage systems. In terms of bandwidth repair, Dimakis et al.⁴ innovatively applied the idea of network coding to bandwidth repair, and provided constraints and theoretical boundaries for encoding and decoding parameters, proposing the concept of regenerating code. Then, the encoding and decoding of regenerative codes that meet theoretical boundaries have received extensive theoretical research⁵⁻⁷.

However, the introduction of regenerative code will bring some problems in system design, such as the design and scheduling of multiple slices of a single strip. This greatly increases the complexity of system design. At present, except that Clay Code⁸ is used as an optional erasure Code module in the distributed storage system Ceph⁹, Clay Code will introduce the problem of reading amplification, and no production-level storage system uses regenerative code. High CPU overhead is another issue brought about by the introduction of erasure codes in distributed storage systems. Therefore, reducing CPU overhead, improving the throughput of encoding and decoding, and improving the speed of I/O reading and writing as well as data repair are very important in system design. Plank et al.¹⁰ split the multiplication mapping table into

*jingsuncs@126.com

upper and lower tables, reducing the number of calls between CPU registers and memory, and improving the hit rate of CPU cache. In addition, using Intel's SIMD (Single Instruction Multiple Data) instruction set for acceleration can significantly improve the throughput of RS. Chen et al.¹¹ evaluated the throughput performance of various encoding and decoding algorithms using SIMD instruction set optimization.

Compared to the replica strategy, erasure codes can significantly reduce storage costs. Reference¹² proposes an optimal method based on integer programming for deploying erasure codes on edge servers. Compared to the storage method of replicas, this method can save 68.58% of storage costs. Reference¹³ proposes a new erasure code process based on the Liberation code, called NewLib code. It designs a data alignment method after encoding data into blocks and optimizes data distribution and task scheduling through consistent hashing, improving read and write performance in distributed storage systems.

The theoretical system and implementation of the throughput of erasure codes under a single algorithm have undergone significant development. However, in combination with the business scenarios of distributed storage systems, especially in multi tenant scenarios, research on how to convert data encoded with erasure codes within or between systems still lacks theoretical and implementation support.

2. ENCODING EQUIVALENCE AND DATA CONVERSION PRINCIPLES

Currently, the implementation mechanisms of mainstream RS encoding and decoding algorithms can be divided into two types: construction based on generative matrices and construction based on generative polynomials. Encoding and decoding algorithms based on generative polynomials are relatively complex and may result in relatively low throughput when implemented. However, this encoding and decoding algorithm can detect silent errors in striped data sharding. When repairing data, even if the data has been tampered with due to hardware or human factors and the location is unknown, as long as the number of abnormal data shards $s \leq \left\lfloor \frac{n-k}{2} \right\rfloor$ is met, decoding algorithms can still be used for data repair. However, decoding algorithms based on generative matrices, when encountering silent errors, without a system level data integrity verification mechanism, can lead to decoding errors in fragmented data, thereby contaminating user data and causing data inconsistency. Due to different implementations, the encoding and decoding algorithm based on generative matrices can ensure high throughput and improve the read and write performance of distributed storage systems. The comparison between the above two encoding schemes is shown in Table 1:

Table 1. Comparison of two codes.

Encoding method	Encoding and decoding complexity	Silent data corruption	Encoding and decoding throughput
Generator matrix	General	No	High
Generator polynomial	High	Yes	Low

The encoding and decoding methods based on generative matrices and those based on generative polynomials each have their advantages and disadvantages. The specific method used can be determined based on the design goals and architecture of distributed storage systems. For example, if there is already a mechanism at the storage system layer to ensure that each shard in the stripe undergoes data validation during read and write, and has high throughput requirements, a generative matrix based approach can be adopted. Here, write verification is to ensure that bit reversal, software layer errors, or disk hardware data write bit jumps are avoided during the data writing process; read verification is to avoid issues such as data tampering. If relying on the mechanism of encoding itself to ensure its consistency, a method based on generating polynomials can be used. Here are two encoding algorithms:

2.1 Encoding based on generative matrices

There are two main types of construction algorithms based on generative matrices, one is based on the construction of Van der Mond determinant, and the other is based on the construction of Cauchy. These two constructions essentially involve finding a matrix of $n \times k$, so that any $k \times k$ submatrix is invertible, ensuring that a (n, k) band can be decoded from any k shard data.

We let the length of the storage system stripe be n , the number of information bit shards be k , and α is a generator of $GF(2^8)$. We let the elementary matrix \hat{G} be,

$$\hat{G} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha^1 & \alpha^2 & \cdots & \alpha^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{(n-2)} & \alpha^{2(n-2)} & \cdots & \alpha^{(k-1)(n-2)} \end{bmatrix}_{n \times k}$$

By performing an elementary transformation on \hat{G} and converting the k rows above it into an identity matrix I , we can obtain a matrix in the form of $G = \begin{bmatrix} I \\ A \end{bmatrix}$. We define G as the encoding matrix of a distributed storage system.

Assuming $M = [m_1 \ m_2 \ \cdots \ m_k]^t$, the encoded data, $m_i \in GF(2^8)$ can be obtained by calculating GM . We will write G in the following format:

$$G = [I_1^t \ \cdots \ I_k^t \ A_1^t \ \cdots \ A_{n-k}^t]^t$$

Among them, I_i^t is the transpose of a row vector with a length of n , and the i -th position is I , and all other positions are 0, then A_i is the encoding vector of the i -th checksum fragment.

The above provides a method for generating encoding vectors based on van der Mond determinant. However, in different system implementations, due to the different ways of generating matrices, the same data encoding results in different validation shards, which makes it impossible to directly transfer and verify data between systems. The first equivalence proof is given below, which is the equivalence proof between different generative matrices with the same parameters:

Theorem 1 (Equivalence between Generative Matrices): Two encoding schemes based on the same parameters of a generative matrix exhibit equivalence between data.

Proof: There are two encoding matrices based on generative matrices, G_1 and G_2 . Since G_1 and G_2 satisfy that any submatrix of $k \times k$ is invertible, knowing any k shards under any encoding scheme can solve for the information bit M using the decoding algorithm of this scheme, and then calculate the check bit under the other scheme using AM .

The principle of decoding based on generative matrix is to select any k nodes, extract the corresponding encoding row vectors from their generative matrix, form a matrix, calculate its inverse, and then multiply left by the shard information used for decoding to obtain the information bit M .

2.2 Encoding based on generative polynomials

Like the encoding method for generating matrices, the uncoded data information is still represented by $M = [m_1 \ m_2 \ \cdots \ m_k]^t$, we let $m(x) = \sum_{i=0}^k m_i x^{k-i-1}$. If α is a generator of $GF(2^8)$, then the generating polynomial $g(x)$ is defined as:

$$g(x) = (x - \alpha^0)(x - \alpha^1) \cdots (x - \alpha^{n-k-1})$$

$g(x)$ can be used to map k bytes of uncoded data to a k -dimensional subspace in an n -dimensional space, where each encoded codeword is an integer multiple of $g(x)$. Define: $b(x) = x^{n-k}m(x) \pmod{g(x)}$ then $x^{n-k}m(x) = q(x)g(x) + b(x)$, and then $c(x) = x^{n-k}m(x) - b(x)$.

If $b(x) = x^{n-k}m(x) \pmod{g(x)}$ is defined, then $x^{n-k}m(x) = q(x)g(x) + b(x)$, then the codeword polynomial can be defined as

$$c(x) = x^{n-k}m(x) - b(x)$$

If $b(x)$ is represented as $b(x) = \sum_{i=0}^{n-k-1} b_i x^{n-k-i-1}$, then the fragments encoded by RS can be represented as $[m_0 \ m_1 \ \cdots \ m_k \ -b_0 \ -b_1 \ \cdots \ -b_{n-k-1}]$. Because $-\alpha = \alpha$ under $GF(2^8)$, for simplicity, shards can be represented as $[m_0 \ m_1 \ \cdots \ m_k \ b_0 \ b_1 \ \cdots \ b_{n-k-1}]$.

To prove that encoding based on generative polynomials is equivalent to encoding based on generative matrices, a lemma is given below:

Lemma 1 For any root β of the generating polynomial $g(x)$, there is $\beta^{n-k}m(\beta) = b(\beta)$.

Proof: Due to $x^{n-k}m(x) = q(x)g(x) + b(x)$ and $g(\beta) = 0$, the above lemma can be directly obtained.

Theorem 2 (Equivalence between Generative Polynomials and Generative Matrices): Under the condition of $n \leq$

$2k - 1$, any encoding scheme based on Generative Polynomials can be transformed into an encoding scheme based on Generative Matrices with the same parameters, resulting in equivalence between data.

Proof: we let the generating polynomial under $GF(2^8)$ be $g(x) = (x - \alpha^0)(x - \alpha^1) \cdots (x - \alpha^{n-k-1})$, as can be obtained from Lemma 1

$$\begin{aligned} m(1) &= b(1) \\ \alpha^{n-k} m(\alpha) &= b(\alpha) \\ &\dots \\ \alpha^{(n-k)(n-k-1)} m(\alpha^{n-k-1}) &= b(\alpha^{n-k-1}) \end{aligned}$$

We let

$$\begin{aligned} A_l &= \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \alpha & \cdots & \alpha^{n-k-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-k-1} & \cdots & \alpha^{(n-k-1)(n-k-1)} \end{bmatrix} \\ \Lambda &= \begin{bmatrix} 1 & & & \\ & \alpha^{n-k} & & \\ & & \ddots & \\ & & & \alpha^{(n-k-1)(n-k)} \end{bmatrix} \\ A_r &= \begin{bmatrix} 1 & \cdots & 1 \\ \alpha^{n-k} & \cdots & \alpha^{k-1} \\ \vdots & \ddots & \vdots \\ \alpha^{(n-k-1)(n-k)} & \cdots & \alpha^{(n-k-1)(k-1)} \end{bmatrix} \end{aligned}$$

The above equation can be expressed as

$$\Lambda[A_l|A_r]M = A_l[b_0 \cdots b_{n-k-1}]$$

Based on the generated polynomial, the check sharding can be expressed as

$$[b_0 \cdots b_{n-k-1}] = A_l^{-1} \Lambda[A_l|A_r] M$$

The encoding scheme based on the generating polynomial $g(x) = (x - \alpha^0)(x - \alpha^1) \cdots (x - \alpha^{n-k-1})$ is equivalent to the encoding scheme based on the generating matrix $A_l^{-1} \Lambda[A_l|A_r] A$.

It should be noted that $[A_l|A_r]$ needs to ensure that $n - k \leq k - 1$, i.e. $n \leq 2k - 1$.

2.3 The design principle of the first verification shard of the strip

Whether based on generating matrices or generating polynomials, when designing encoding schemes, some systems may require the first checksum shard to be the XOR of all data shards. This design has two advantages: (1) In a stripe, the most common occurrence is when a single data shard is damaged (the most common situation in a distributed storage system is when a single disk fails). Therefore, designing the first checksum shard to be the XOR of all data shards can simplify the decoding algorithm. When recovering single shard data, CPU overhead can be reduced only through XOR. (2) In multi tenant scenarios, when the data in the system needs to undergo encoding scheme changes, such as converting from RS encoding to LRC encoding, there is no need for large-scale data validation adjustments. Just split the first check shard into two check shards according to the LRC algorithm.

If it is based on the encoding construction of generative matrices, A_l^t in G can be transformed into all 1s according to elementary row transformations to meet this requirement. The encoding construction based on generating polynomials is related to the selection of generating polynomials under $GF(2^8)$. In fact, the $g(x) = (x - \alpha^0)(x - \alpha^1) \cdots (x - \alpha^{n-k-1})$ we have chosen the following properties:

Lemma 2: Through strip sharding under $g(x) = (x - \alpha^0)(x - \alpha^1) \cdots (x - \alpha^{n-k-1})$ encoding, the XOR of all data shards is equal to the XOR of all check shards, i.e. $\sum_{i=0}^k m_i = \sum_{i=0}^{n-k} b_i$.

Proof: According to Lemma 1, $1^{n-k} m(1) = b(1)$, that is, Lemma 2 holds.

If the encoding is constructed through the method of generating polynomials, the first check shard can be replaced with

the XOR of all check shards after encoding, that is, $\sum_{i=0}^{n-k} b_i$, to meet this design requirement.

3. DESIGN OF ENCODING AND DECODING MODULE FOR MULTI TENANTS

Multi tenant is a feature of distributed storage systems. Granting tenants greater freedom to choose what configuration to use for encoding and decoding can enable them to choose how to store their data based on their analysis of complex business and reliability needs. Although both open-source and commercial distributed storage systems currently have multi tenant functionality, there is no open custom stripe ratio feature. Generally, a single cluster can only use a fixed ratio encoding and decoding method, or, like Ceph, customize a fixed ratio algorithm for a single storage pool, which cannot achieve tenant level granularity. The flexible definition of encoding and decoding algorithms and parameter ratios has multiple practical application scenarios, including but not limited to (1) Users decide whether to support silent error detection and repair based on the design goals of distributed storage systems and thus choose whether to use encoding and decoding algorithms based on generative matrices or polynomials. (2) Users have different reliability requirements for their data sources and need to use different parameter allocation schemes for different data. (3) Users need to improve the reliability of some of their data, which leads to parameter expansion or, based on cost considerations, encode and crop the stored data. (4) Users migrate data with the same configuration between different storage systems. (5) Users need to change the encoding scheme to improve the speed of disk repair, such as converting RS encoding to LRC encoding¹⁴.

This chapter is based on multi tenant scenarios and abstracts the encoding and decoding module from the perspective of allowing flexible definitions of encoding and decoding algorithms and parameter ratios. Due to the different goals of system design, there will be significant differences in metadata management and data management solutions. Therefore, we will not discuss the design details of metadata management and data management here but only focus on the design of the encoding and decoding module itself. The module design under multi tenant is shown in Figure 1:

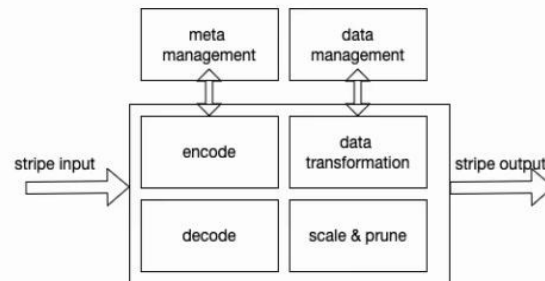


Figure 1. Encode module design for multi-tenant.

The encoding and decoding module under multi tenant includes four main functions: (1) encoding, (2) decoding, (3) data conversion, and (4) expansion and cropping. Its input is in a striped manner, with bytes as the minimum granularity, while supporting batch operations to improve memory and CPU utilization. Based on actual system design experience, the batch size is 16KB-256MB. In the following chapter, we will provide a detailed introduction to the implementation of these modules.

3.1 Encoding

There are two types of strip encoding algorithms: strip encoding algorithms based on generative matrices and strip encoding algorithms based on generative polynomials. When inputting data, the source information is written into the distributed storage system and can be configured through parameters. The first encoding scheme based on generative matrix adopts the algorithm principles in Chapter 2.1, and at the same time, adjusts the generative matrix so that the first check fragment is equal to the XOR of all data fragments. The second encoding scheme based on generative polynomials adopts the algorithm principles in Chapter 2.2 and adjusts the first check shard by XOR all the data from the check shards and replacing it with the first check shard.

3.2 Decoding

If the encoding method is based on generating matrices, during decoding, the first step is to determine whether only a single shard of data cannot be read. If so, XOR can be obtained by reading the first checksum shard and other data shards. If two or more data shards cannot be read, it is necessary to read shards, form a matrix of their encoding vectors, calculate the inverse, and finally multiply with shards to obtain all data shard information.

The decoding algorithm based on generating polynomials is relatively complex, and there are various implementations of decoding algorithms. We have summarized that its implementation requires the following four steps: (1) calculating the polynomial, (2) creating an error localization polynomial, (3) determining the error bit based on the error localization polynomial, and (4) error correction. In step 3, if the location of the error is already known at the system design level, it can be ignored and corrected directly. References^{15,16} provide specific decoding algorithms.

3.3 Expansion and cutting

When users adjust the cost and reliability of their data, they need to expand or crop the corresponding bands of the data. Here we analyze the method based on generating matrices. To avoid recalculating the generation matrix G , a relatively large value can be initialized for the number of rows in G to generate a pre-generated encoding row sequence A_1, A_2, \dots, A_{n-k} . For example, if the parameter ratio of the current stripe is $(n, k) = (12, 4)$, the encoding vector A_1 - A_6 for 6 check shards can be generated in advance. When expansion $(n, k) = (14, 6)$ is needed, simply take out A_5 and A_6 , and encode 2 verification shards again.

According to the properties of the van der Mond determinant, the configured parameters need to satisfy $n \leq 2^8$. When designing the system, the value of k should not be too large, otherwise, the penalty for bandwidth transmission during data recovery will be too large (at least k times the amount of data is needed to recover the data). So, the verification sharding can be configured very large. In a production environment, $n - k = 4$ is a common allocation method for storage systems below the EB level. On smaller cluster sizes (not exceeding 1PB) or flash array systems, $n - k = 2$ is also more common. So, as long as several pre-generated encoding vectors are allocated for verification, they can meet the requirements of distributed storage system design.

The cutting and expansion of stripes are the opposite, as long as the verified shards are deleted one by one or in batches from the back to the front, and the metadata information is modified. Distributed storage systems require a combination of metadata management module and data management module to achieve stripe expansion and cropping. Before expansion, the data management module needs to allocate idle shards for writing verification data. After cropping, the system needs a mechanism to recycle and reuse the physical space that has been cropped.

3.4 Data conversion

Theorem 2 provides the principle of data conversion based on generative polynomials and generative matrices. In fact, for an encoding based on generative polynomials $g(x) = (x - \alpha^0)(x - \alpha^1) \dots (x - \alpha^{n-k-1})$, it can be implemented as an encoding based on generative matrices $A_l^{-1} \Lambda[A_l | A_r]$. Without considering silent errors, data recovery and repair can be achieved through decoding algorithms that generate matrices, thereby improving decoding throughput and reducing disk repair time.

Different systems, whether using generative matrix based construction or generative polynomial based construction, can convert data without the need to split into decoding and encoding sub steps, provided that the generative polynomial and generative matrix are given.

4. PERFORMANCE EVALUATION EXPERIMENT

4.1 Encoding throughput

In order to reduce CPU overhead and improve throughput, many CPU manufacturers provide registers with more bits and provide SIMD (single instruction multiple data) for data read, write, and operation on these registers. By providing parallelism for data operations and reducing the number of instruction pipelines, the performance of encoding and decoding can be improved. In the current production environment system, almost all encoding and decoding engines adopt SIMD optimization.

There are many open-source implementations based on generative matrix encoding, but open-source implementations optimized by SIMD mainly include Intel's ISA-L¹⁷, James Plank's Jerusalem¹⁸ (only supports sse3, not avx2), and Klauspost's erasure code library implemented using Golang and assembly¹⁹. With the development of hardware, the extension of the avx2 instruction set has been achieved in Intel CPUs, and in general, the performance of the avx2 based instruction set is more than twice that of the sse3 based instruction node. So based on the encoding library of generative matrices, only the implementation of ISA-L and Klauspost will be evaluated here.

So far, there is no open-source library implementation that supports AVx2 optimization based on generative polynomial

encoding and decoding. This article is based on the equivalence principle in Chapter 2 and implements an open-source library for encoding and decoding based on generative polynomials that support avx2²⁰. The core code is implemented in C, and a layer of wrapper is created using Golang to facilitate interface calls.

The environment for encoding throughput performance testing is Intel (R) Xeon (R) CPU E5-2640 v4@2.40GHz Model CPU, single core operation, 1GB memory limit, stripe parameter ratio $(n, k)=(14,4)$, single stripe size is 16KB-258MB. The throughput performance of the encoding is shown in Figure 2.

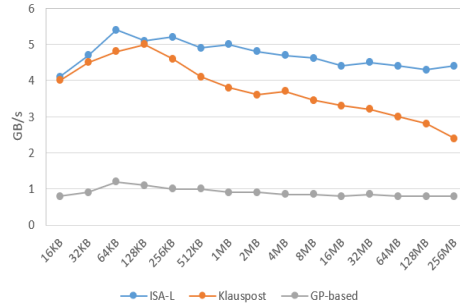


Figure 2. Encoding throughput.

From the performance test in Figure 2, it can be seen that ISA-L can provide the best encoding and decoding performance. The optimization of ISA-L is relatively good. On the one hand, Intel has carried out a lot of assembly optimization on the encoding and decoding process, implementing the overall matrix operation using assembly, but it has also led to difficulties in understanding the implementation. Klauspost utilizes Golang for assembly support, making great use of SIMD technology. The matrix operation part is implemented at the high-level language level, and only some optimized code is implemented using Golang's built-in Plan9 assembly. When the stripe size is relatively small, the encoding throughput of Klauspost is similar to that of ISA-L, but when the stripe size exceeds 256KB, ISA-L can better demonstrate the performance advantage. The encoding and decoding library based on generating polynomials implemented in this article has a coding throughput as shown in GP-based (generator polynomial based).

The performance of GP-based is between ISA-L and Klauspost. When the shard size is less than 512KB, Klauspost has a greater advantage in encoding and decoding throughput (Klauspost has optimized for small shards). When the shard size is greater than 512KB, the throughput of GP-based itself is better than Klauspost. Although the overall throughput of GP-based is lower than ISA-L, it can be used directly in the Golang language environment because it already supports the wrapper layer.

4.2 Decoding throughput

Due to the existence of Lemma 2, one check in the band can be transformed into XOR. When decoding in this way, both traditional matrix inversion and XOR decoding can be used for decoding. Through XOR, the throughput can be significantly improved during the assembly optimization process.

When a single node fails, data comparison will be conducted based on four decoding algorithms: ISA-L, Klauspost, GP-based, and GP-based (optimize), as shown in Figure 3. Among them, GP-based uses matrix inversion method, and the first checksum node of the strip corresponding to GP-based (optimize) is encoded using XOR method.

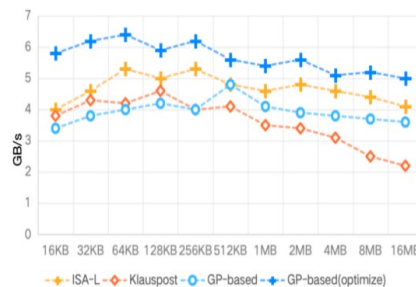


Figure 3. Encoding throughput with a single failure node.

The decoding throughput of GP-based is between ISA-L and Klauspost. When the shard size is greater than 256KB, the

GP-based method is superior to Klauspost. And GP-based (optimize) has the optimal decoding throughput because it uses XOR decoding method through linear permutation.

5. CONCLUSION

This article provides an encoding and decoding algorithm based on erasure correction codes and a proof of equivalence principle. And under multiple tenants, multiple encoding and decoding modules with parameters were designed, which can meet the different storage cost and reliability requirements of different tenants, and can flexibly expand and reduce the storage scheme. At the implementation level, this article provides an implementation of encoding and decoding based on generative polynomials. Through the principle of equivalence, the conversion mechanism of data under different schemes has been established. However, research on multiple issues related to elastic encoding and decoding schemes is still in its early stages. On the one hand, it is due to compatibility issues with the system architecture, which require IO process modifications at the system level. On the other hand, supporting different encoding and decoding algorithms and optimizing their repair bandwidth remains a challenge in system design. Further in-depth research will be conducted on these two aspects in the future.

REFERENCES

- [1] Blaum, M., Plank, J. S., Schwartz, M., et al., "Construction of partial MDS and sector-disk codes with two global parity symbols," *IEEE Transactions on Information Theory*, 62(5), 2673-2681 (2016).
- [2] Sun, J., Liang, S. T., Lu, X. J., "An approximately optimal disk repair algorithm for distributed storage systems (in Chinese)," *Scientia Sinica Informationis*, 50(12), 1834-1849 (2020).
- [3] Yu, L., Lin, Z., Lin, S. J., et al., "Fast encoding algorithms for Reed–Solomon codes with between four and seven parity symbols," *IEEE Transactions on Computers*, 69(5), 699-705 (2020).
- [4] Dimakis, A. G., Godfrey, P. B., Wu, Y., et al., "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, 56(9), 4539-4551 (2010).
- [5] Duursma, I. M., "Shortened regenerating codes," *IEEE Transactions on Information Theory*, 65(2), 1000-1007 (2018).
- [6] Shao, S., Liu, T., Tian, C., et al., "On the tradeoff region of secure exact-repair regenerating codes," *IEEE Transactions on Information Theory*, 63(11), 7253-7266 (2017).
- [7] Lakshmi, V. S. and Deepthi, P. P., "Error correction scheme for regenerating code based distributed storage systems," *Proceedings of the 2018 International Conference on Communication Engineering and Technology*, 5-7, (2018).
- [8] Vajha, M., Ramkumar, V., Puranik, B., et al., "Clay codes: Moulding {MDS} codes to yield an {MSR} code," 16th {USENIX} Conference on File and Storage Technologies ({FAST} 18), 139-154 (2018).
- [9] <https://ceph.io/>
- [10] Plank, J. S., Greenan, K. M., Miller, E. L., "Screaming fast Galois field arithmetic using intel SIMD instructions," *FAST*, 299-306 (2013).
- [11] Chen, R. and Xu, L., "Practical performance evaluation of space optimal erasure codes for high-speed data storage systems," *SN Computer Science*, 1(1), 1-14 (2020).
- [12] Jin, H., Luo, R., He, Q., Wu, S., Zeng, Z. and Xia, X., "Cost-effective data placement in edge storage systems with erasure code," *IEEE Transactions on Services Computing*, 16(2), 1039-1050 (2022).
- [13] Yin, C., Xu, Z., Li, W., Li, T., Yuan, S. and Liu, Y., "Erasure codes for cold data in distributed storage systems," *Applied Sciences*, 13(4), 2170 (2023).
- [14] Tamo, I., Papailiopoulos, D. S. and Dimakis, A. G., "Optimal locally repairable codes and connections to matroid theory," *IEEE Transactions on Information Theory*, 62(12), 6661-6671 (2016).
- [15] Berlekamp, E. R., [Non-binary BCH decoding], North Carolina State University, Dept. of Statistics, (1966).
- [16] Thomas, H., Cormen, Charles, E., et al., [Introduction to Algorithms] Second Edition, MIT Press and McGraw-Hill, ISBN 0-262-03293-7. 859-861 of section 31.2: Greatest common divisor, (2001).
- [17] <https://github.com/intel/isa-l>
- [18] <http://jerasure.org/jerasure/jerasure/>
- [19] <https://github.com/klauspost/reedsolomon>
- [20] <https://github.com/cloudlounge/erasurecode>