ATLANTIS PRESS
Advances in Computer Science Research, volume 71
4th International Conference on Machinery, Materials and Information Technology Applications (ICMMITA 2016)

Application of Refined Increasing Difference Property in XOR-based MDS Codes

Lei Zhang[1], Yong Peng[2], ZhaoHui Liu[3], Jie Liang[4], and Chang Jin[5]

China Information Technology Security Evaluation Center, Beijing, China
[1]zhanglei@itsec.gov.cn

Keywords: MDS code, refined increasing-difference property, distributed storage system.

Abstract. This paper applies the refined increasing difference property to XOR-based MDS codes and obtains a lower storage complexity for distributed storage nodes. For a message of kL bits stored in n distinct storage nodes, a data collector connects any k out of the n storage nodes to recover the message. In this scheme, there is no transmission overhead, that is the bits transmitted to the data collector is exactly the bit number of the message. In addition, the recovery algorithm is within XOR operations so that the decoding complexity is low. But we need less storage than previous scheme.

I. Introduction

In a distributed storage system, a data file is divided into k blocks with each containing L bits, then are encoded and stored in n distinct nodes. A data collector (called DC for short) can access any part of the nodes and the packets are transmitted to the DC so that DC can recover the message. The definition of MDS codes follows that of [1], that is it is called MDS codes when the DC can recover the source message if it accesses any k out of n nodes.

The first well-known MDS codes is Reed-Solomon code, which is called RS code [2]. RS code is constructed using finite field, and the decoding needs finite filed multiplication, which is of high complexity. The high complexity limits its application. So MDS codes within simple operations have been under consideration. Paper [3]–[9] proposed some MDS codes that can correct some fixed number such as one, two, or three node failures. The first XOR-based MDS code that can correct any node failure was proposed by Sung et al. in [10]. This construction was derived from the finite filed construction of RS codes. They also proposed a decoding algorithm which is called ZigZag decoding using just bit shift and XOR operations. The encoding and decoding are within shift and XOR operations and hence, it is of much lower complexity. The decoding complexity is O k2L , which is the complexity bound from the view of information theory using this encoding scheme. Due to the shift operations, the bits stored in each node is more than L. In addition, the bits transmitted to the DC from each node is also more than L using this scheme, so the MDS property does not hold. Then Fu et al. improved this work by eliminating the overhead [1]. They proved that if the generator matrix satisfies the increasing-difference property, then the overhead-free in-place recovery scheme can work [1]. The most well know generator matrix that satisfies the increasing difference property is the Vanermonde matrix. Then this property was applied to XOR-based regenerating codes [11].

This paper considers the XOR-based storage codes proposed in [10] due to the low encoding complexity. We combine the refined increasing difference property proposed in [11] and overhead-free in-place recovery scheme proposed in [1] together and apply the refined increasing difference property in XOR-based storage codes. Our scheme includes all the characteristics in [1] and we can obtain a lower storage complexity.

The rest of the paper is organized as follows: In Section II, the distributes storage system will be introduced. In Section III, our recovery scheme will be illustrated. Specifically, Section III-A and Section III-B introduce the transmission stage and the decoding stage in the recovery scheme, respectively. Finally, in Section IV, we summarize the paper.

II. System Description

The encoding scheme of the distributed storage system in this paper follows that of [1], [10], [11]. The message consists of $k$blocks, namely,$\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k$. A block is a sequence of L bits and the message contains kL bits. The generator matrix $\mathbf{\Psi}$ is an $n \times k$ matrix,

$$\mathbf{\Psi} = \begin{pmatrix} z^{t_{1,1}} & z^{t_{1,2}} & \cdots & z^{t_{1,k}} \\ z^{t_{2,1}} & z^{t_{2,2}} & \cdots & z^{t_{2,k}} \\ \vdots & \vdots & \ddots & \vdots \\ z^{t_{n,1}} & z^{t_{n,2}} & \cdots & z^{t_{n,k}} \end{pmatrix}$$

where $t_{i,j} \in \mathbb{N}$ $(1 \le i \le n,\ 1 \le j \le k)$ satisfies the *refined increasing-difference property* [11]: for any i, i', j, and j' such that $i < i'$ and$j < j'$, $0 \le t_{i,j'} - t_{i,j} < t_{i',j'} - t_{i',j}$, where the first equality holds only when $i = 1$. (Normally, if $t_{i,j} = (i-1)(j-1)$, the matrix is a Vandermonde matrix, where it is $t_{i,j} = (i-1)(j-1)$ in [1]).

Then these k blocks of message are encoded into n packets, $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n$, by multiply the generator matrix with the source blocks, i.e.,$\mathbf{y} = \Psi\mathbf{x}$. Specifically, to generate $\mathbf{y}_i$ $(1 \le i \le n)$, shift $t_{i,j}$ bits to the right and padding $t_{i,j}$ zero bits and $(t_{i,k} - t_{i,j})$ zero bits to the leftmost and the rightmost of $\mathbf{x}_j(1 \le j \le k)$, respectively, which results in a sequence of bits of length $(L + t_{i,k})$ denoted by

$$z^{t_{i,j}}\mathbf{x}_j = \left(\mathbf{0}_{1\times t_{i,j}}, \mathbf{x}_j, \mathbf{0}_{1\times\left(t_{i,k}-t_{i,j}\right)}\right),$$

where $\mathbf{0}_{1\times m}$ is an m-dimensional row vector whose elements are all zero.
Then $\mathbf{y}_i = (y_i[1], \cdots, y_i[L + t_{i,k}])$ is formed by

$$\mathbf{y}_i = \sum_{j=1}^{k} z^{t_{i,j}}\mathbf{x}_j. \tag{1}$$

There are n nodes, indexed by $1, 2, \cdots, n$, in the distributed storage system. The i-th coded packet $\mathbf{y}_i$ is stored in node i $(1 \le i \le n)$. So the total storage complexity for node i is$L + t_{i,k}$.

III. Recovery Scheme

This section gives our scheme to recover the message from arbitrary k nodes. This scheme consists of two stages, namely, the transmission stage and the decoding stage.

A. Transmission Stage

The transmission stage follows that of [1]. Suppose that the DC connects k out of n nodes. Now DC needs to recover the original message from the coded packets stored in these k nodes. First, the indices of the nodes connecting to DC are sorted in descending order, so that $i_1, i_2, \cdots, i_k$ $(n \ge i_1 > i_2 > ... > i_k \ge 1)$. This operation can be executed in DC. Then node i_u that its index is the u-th largest one among those k nodes. Then node $i_u$transmits L bits of the packet $\mathbf{y}_{i_u}$from the $(t_{i_u,u} + 1)$th bit to the $(t_{i_u,u} + L)$th bit, to DC. Then DC stores the L bits in $\hat{\mathbf{x}}_u = (\hat{x}_u[1], \cdots, \hat{x}_u[L])$.

Transmission bandwidth: Each node needs to transmit L bits exactly, so the total number of bits to transmit to DC is kL, which is equal to the number of the message bits.

B. Decoding Stage

After each of the selected k nodes transmits L bits of its packet to DC, DC uses an in-place decoding algorithm to recover the message from $\hat{\mathbf{x}}_u$ $(1 \le u \le k)$. The detail of this algorithm is presented in Algorithm 1 of [1]. Differently from the ZigZag decoding in [10], there is no copy of the original data. We use a vector $(l_1, \ldots, l_k)$ to record the number of decoded bits in $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k)$. It means l_u $(1 \le u \le k)$ bits of $\mathbf{x}_u$ have been decoded. Specifically, when $l_u = l$ $(1 \le u \le k)$, the bits

$\hat{x}_u[1], \ldots, \hat{x}_u[l]$ have been exactly equal to $x_u[1], \ldots, x_u[l]$. When $l_k = l$, all blocks have been decoded and hence, the decoding algorithm can stop, which is shown in Line 2. In the initialization step, all entries in $(l_1, \ldots, l_k)$ are set to zero for that there is no any decoded bit. During the iterations, each entry in $(l_1, \ldots, l_k)$ increases gradually to L. Finally, all entries in $(l_1, \ldots, l_k)$ become L, which means that the message has been fully recovered. So the blocks are decoded one by one, x_u is the u-th decoded block.

The decoding algorithm consists of two aspects, of which the first is to free one more bit in $\hat{x}_u$ and the second is to eliminate the superposed bits. For each iteration of the for loop in Line 3, DC judges whether $\mathbf{x}_u$ has been fully recovered. If $l_u = L$, the $\mathbf{x}_u$ has been fully recovered, and no further operations are needed. Otherwise ($l_u < L$), the two aspects are involved and DC tries to decode the bit $x_u[l_u + 1]$ and eliminate $x_u[l_u + 1]$ from other packets. Let l be the current value of $l_u + 1$ $(1 \leq l \leq L)$. When $u = 1$, $l_u = 0$ and DC can increase l_u from 0 to 1 because the first bit of $\hat{x}_1$ is the first bit of $\mathbf{x}_1$. When $l_{u-1} > t_{i_u,u} - t_{i_u,u-1}$, all the other bits except $\hat{x}_u[l]$ superposed on this bit have been eliminated, so the bit $\hat{x}_u[l]$ has become the original message bit $x_u[l_u + 1]$, so DC increases l_u from $l - 1$ to l to record the successful recovery of the bit $x_u[l]$. Then DC executes the second aspect to eliminate the recovered bit $x_u[l]$ from $\hat{\mathbf{x}}_v$ $(v \neq u)$. According to the encoding scheme, $x_u[l]$ is superposed on $y_{i_v}[l + t_{i_v,u}]$ $(v \neq u)$. Note that node i_v transmits $\mathbf{y}_{i_v}$ ranging from $(t_{i_v,v} + 1)$ to $(t_{i_v,v} + L)$. Therefore, when $t_{i_v,v} < l + t_{i_v,u} \leq t_{i_v,v} + L$, $x_u[l]$ is involved in generating $\hat{x}_v[l + t_{i_v,u} - t_{i_v,v}]$. Line 8 eliminates these superposed bits according to the superposition scheme.

Algorithm 1 can be verified by the following illustration. During the execution of the decoding, $(l_1, \ldots, l_k)$ increase in the following way: At the first several iterations, only l_1 increases because $l_1 \leq t_{i_2,2} - t_{i_2,1}$. After l_1 becomes $l_1 \leq t_{i_2,2} - t_{i_2,1}$, l_2 begins to increase in the same pace with l_1. From then on, $(l_2 - l_1)$ is kept to be $(t_{i_2,2} - t_{i_2,1})$ until l1 reaches L and stops increasing. Similarly, for any $u \neq 1$, l_u begins to increase when l_{u-1} becomes $(t_{i_u,u} - t_{i_u,u-1} + 1)$, and $(l_u - l_{u-1})$ is kept to be $(t_{i_u,u} - t_{i_u,u-1})$ until l_{u-1} reaches L. Finally, all entries in $(l_1, \ldots, l_k)$ end up in L. So $\mathbf{x}_u$ is the u-th decoded block.

Normally, for any snapshot of the decoding process, $l_u > l_{u-1}$ $(1 < u \leq k)$. Therefore, we have

$$L \geq l_1 \geq l_2 \geq \cdots \geq l_k \geq 0$$

at any time. When l_k becomes L, $l_u = L$ $(1 \leq u \leq k)$, the stored block have been the message blocks, i.e.,

$$\hat{\mathbf{x}}_u = \mathbf{x}_u, \quad 1 \leq u \leq k.$$

Algorithm 1 can be verified by modifying the proof in [1]. For more detail of Algorithm 1, you can refer to [1].s

C. Complexity Analysis

Space Complexity: In this algorithm, the only auxiliary space needed is the vector $(l_1, l_2, \ldots, l_k)$ to record the currently decoded bits. So the space complexity is $O(k \log L)$ bits.

Time Complexity: Due to the complexity analysis in [1], the time complexity of our scheme is $O(k^2 L)$.

D. Comparison with Previous Recovery Scheme

1) Comparison with Scheme in [10]: Our scheme can guarantee the overhead-free property, that is the bits transmitted to the DC are exactly kL, which is optimal in information theory. In addition, we need much less auxiliary space for recovery, which is $O(k \log L)$ while it is $O(kL)$ in [10].

2) Comparison with Scheme in [1]: The number of bits stored in node i is $L + (i - 1)k$ while it is $L + ik$ in [1]. So the storage complexity is lower than that of [1].

IV. Conclusion

This paper applies the refined increasing-difference property in XOR-based MDS codes. Lower storage complexity can be obtained using this new construction. In addition, the overhead-free in-place recovery scheme can be applied using this construction.

References

[1]X. Fu, Z. Xiao, and S. Yang, "Overhead-free in-place recovery scheme for XOR-based storage codes," in IEEE Int. Conf. Trust, Security and Privacy in Computing and Communications, Sep. 2014, pp. 552–557.
[2]I. Reed and G. Solomon, "Polynomial codes over certain finite fields," Journal of the Society for Industrial and Applied Mathematics, pp. 300–304, Jun. 1960.
[3]J. Blmer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme," IGSI Technical Report No. TR-95-048, Aug. 1995.
[4]M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," IEEE Trans. Comput., vol. 44, no. 2, pp. 192–202, Feb. 1995.
[5]M. Blaum, J. Bruck, and A. Vardy, "MDS array codes with independent parity symbols," IEEE Trans. Inf. Theory, vol. 42, no. 2, pp. 529–542, Mar. 1996.
[6]L. Xu, V. Bohossian, J. Bruck, and D. Wagner, "Low-density MDS codes and factors of complete graphs," IEEE Trans. Inf. Theory, vol. 45, no. 6, pp. 1817–1826, Sep. 1999.
[7]P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in
USENIX Conf. File and Storage Technologies, Mar. 2004, pp. 1–14.
[8]G. Feng, R. Deng, F. Bao, and J. Shen, "New efficient MDS array codes for RAID, Part I: Reed-Solomon-like codes for tolerating three disk failures," IEEE Trans. Comput., vol. 54, no. 9, pp. 1071–1080, Sep. 2005.
[9]C. Huang and L. Xu, "STAR: An efficient coding scheme for correcting triple storage node failures," IEEE Trans. Comput., vol. 57, no. 7, pp. 889–901, Jul. 2008.
[10] C. Sung and X. Gong, "A ZigZag-decodable code with the MDS property for distributed storage systems," in IEEE Int. Symp. Inf. Theory, Jul. 2013, pp. 341–345.
[11] X. Fu, Z. Xiao, and S. Yang, "Overhead-free in-place recovery and repair schemes of xor-based regenerating codes," in 2015 IEEE International Symposium on Information Theory (ISIT). IEEE, 2015, pp. 854–858.