Z codes: General Systematic Erasure Codes with Optimal Repair Bandwidth and Storage for Distributed Storage Systems

Qing Liu*, Dan Feng*, Hong Jiang†, Yuchong Hu*, Tianfeng Jiao*
*Wuhan National Laboratory for Optoelectronics (WNLO),
*School of Computer, Huazhong University of Science and Technology (HUST), China
†University of Nebraska-Lincoln, USA
Email: {qing, dfeng}@hust.edu.cn, jiang@cse.unl.edu, {yuchonghu, tfjiao}@hust.edu.cn

***Abstract*—Erasure codes are widely used in distributed storage systems to prevent data loss. Traditional erasure codes suffer from a typical repair-bandwidth problem in which the amount of data required to reconstruct the lost data, referred to as the repair bandwidth, is often far more than the theoretical minimum. While many novel erasure codes have been proposed in recent years to reduce the repair bandwidth, these codes either require extra storage capacity and computation overhead or are only applicable to some special cases.**

To address the weaknesses of the existing solutions to the repair-bandwidth problem, we propose Z Codes, a general family of codes capable of achieving the theoretical lower bound of repair bandwidth for a single data node failure. To the best of our knowledge, the Z codes are the first *general systematic* erasure codes that achieve optimal repair bandwidth under the minimum storage. Our in-memory performance evaluations of a 1-GB file indicate that Z codes have encoding and repairing speeds that are approximately equal to those of the Reed-Solomon (RS) codes, and their speed on the order of GB/s practically removes computation as a performance bottleneck.

***Index Terms*—Erasure Codes; Repair Bandwidth; Distributed Storage System; Failure Tolerance**

I. Introduction

Erasure codes are widely used in distributed storage systems to recover from data loss in the event of server breakdown. These codes incorporate data redundancy in a space-efficient manner to tolerate data loss by reconstructing the lost data and are *systematic* in that the original data is kept unchanged after encoding and can be accessed without decoding. Typical systematic codes include Reed-Solomon (RS) codes and Cauchy Reed-Solomon (CRS) codes.

However, such traditional erasure codes face a known repair-bandwidth problem [1] that becomes increasingly more important in a distributed environment where bandwidth is typically expensive in terms of both performance and power consumption. That is, in a storage system of data size M with k data nodes and m parity (i.e., redundant) nodes that are interconnected by a network of limited bandwidth, each node stores data of size $\frac{M}{k}$ and the repair of one node's failure requires a disk-I/O or network bandwidth of size M, which is k times the size of the lost data ($\frac{M}{k}$). In this paper, we define repair bandwidth as the amount of the data accessed by the disk I/O and transferred over the network.

The minimum storage for an (m, k) code is $\frac{M}{k}$, so k nodes of data can retain the original data. However, Dimakis et al. pointed out that the theoretical minimum storage and minimum repair bandwidth cannot be achieved at the same time and there is a lower-bound trade-off curve between the two [1], as plotted in Fig. 1. Although codes with the minimum storage cannot achieve the minimum repair bandwidth, their theoretical repair bandwidth lower bound, which is called *optimal repair bandwidth* [2], can be calculated as:

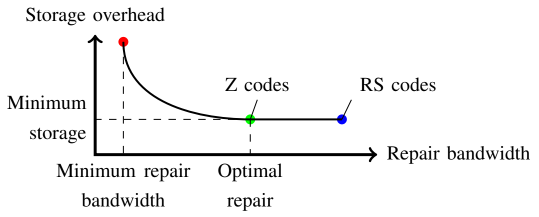


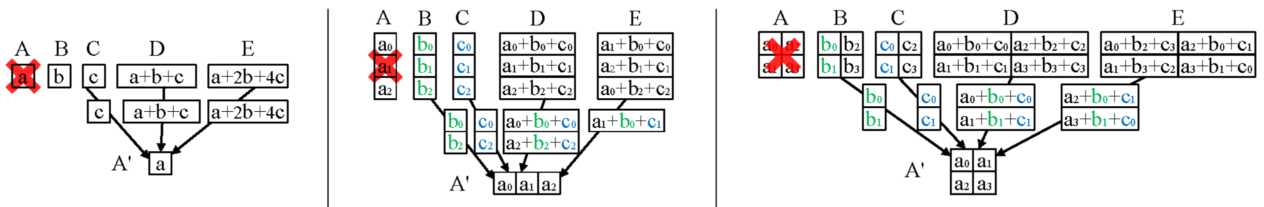
Fig. 1: Theoretical lower-bound trade-off curve of storage overhead and repair bandwidth.

$$(m + k - 1)M/(mk) \tag{1}$$

The repair bandwidth mentioned above refers particularly to a single node failure, which is the most common case in practice.

Recently, many novel repair-bandwidth-efficient codes have been proposed to reduce the repair bandwidth, but at the expenses of (1) extra storage capacity, (2) additional computation overhead or (3) being applicable only to some special cases. The Simple Regenerating Codes (SRC) [3] and Local Reconstruction Codes (LRC) [4] need additional storage resources to store the extra parity information. The Functional Minimum Storage Regenerating (FMSR) codes are not systematic and only store parity information after encoding, thereby resulting in a high computation cost [5]. The Rotate Reed-Solomon (RRS) codes [6] also require additional computation for repairing the failure of a single data node. Under the burden of not having a general construction mechanism, the Zigzag codes [7] are unsuitable for general storage systems. The Product-matrix-MSR (PMSR) codes [2] are only applicable when the *code rate* (the ratio of the data size and size of data after encoding) is less than $\frac{1}{2}$, namely, $m > k$, which greatly limits their applicability.

To address the above weaknesses in the existing codes, we present in this paper a family of novel erasure codes, called the Z codes. The Z codes not only can achieve the theoretical optimal repair bandwidth under the minimum storage for a single data node's failure, but also have the following desirable properties that make them suitable for distributed storage systems. (1) *The minimum storage property*: the Z codes consume exactly the same storage capacity as the RS and



(a) RS code (b) RRS code (c) Z code

Fig. 2: An example of how the three codes, RS, RRS and Z, of $m = 2$ and $k = 3$ repair the lost data in node A (we omit the combination coefficients of parity for the RRS code) and how their corresponding repair bandwidth consumptions decrease with the increase in the number of blocks in each node.

CRS codes. (2) *Low computation overhead*: since the Z codes are XOR codes, all coding operations can be performed by fast bitwise XOR operations; (3) *The systematic property*: the original data remain unchanged after encoding in the Z codes, allowing accesses to data without decoding; (4) *Generality*: the code rate of the Z codes can be arbitrarily high with a flexible parameter set (m, k), meaning that the Z codes can be constructed for arbitrary numbers of data nodes and parity nodes. We call our codes the Z codes because they achieve the ultimate desirable features analogous to the last letter 'Z' in the alphabet.

The Z codes can achieve the optimal repair bandwidth based on the principle that *dividing stored data of each node into a greater number of blocks (denoted by r), more repair bandwidth can be saved by selectively choose some blocks from the remainder nodes during a repair.* In Fig. 2, we compare the repairs of a failed node A for three codes of different rs: the $(2, 3)$-RS code with $r = 1$, the $(2, 3)$-RRS code with $r = 3$ and the $(2, 3)$-Z code with $r = 4$. Repair bandwidths of these three codes, normalized to the size of the original data, are 1 (RS), 0.778 (RRS) and 0.667 (Z) respectively.

Our main contributions are summarized as follows:

- We propose the Z codes, a family of erasure codes with the optimal repair bandwidth under the minimum storage.
- We comprehensively evaluate the performance of the Z codes, in terms of the repair bandwidth consumption and the coding performance.

The rest of the paper is organized as follows. Section II introduces some basic background of all-purpose erasure codes and some repair-bandwidth-efficient codes; We present the Z codes in Section III, including code construction, encoding and repairing; We analyse the repair bandwidth consumption of Z codes in Section IV; Section V evaluates the coding performance of the Z codes; We conclude the paper in Section VI with remarks on the future work.

II. Background and Motivation

In this section, we introduce the necessary background on the erasure-coded distributed storage systems to motivate and facilitate the description of our Z codes.

A. *Erasure Coding in Distributed Storage Systems*

To simplify our discussion, we focus on a stripe over n storage nodes as shown in Fig. 3. A *stripe* [8] is the minimum

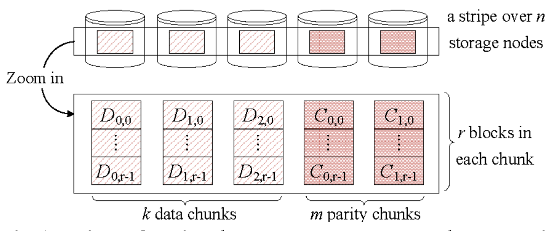

a stripe over n storage nodes
Zoom in
$D_{0,0}$ $D_{1,0}$ $D_{2,0}$ $C_{0,0}$ $C_{1,0}$
$D_{0,r-1}$ $D_{1,r-1}$ $D_{2,r-1}$ $C_{0,r-1}$ $C_{1,r-1}$
r blocks in each chunk
k data chunks
m parity chunks

Fig. 3: A stripe of 5 chunks across 5 storage nodes, one chunk in each node is a set of r blocks.

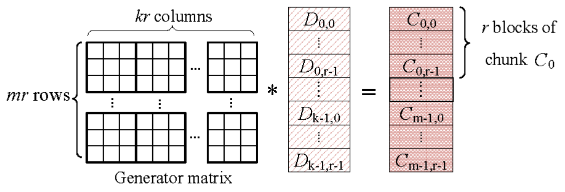


Fig. 4: Encoding data blocks into parity blocks by calculating the matrix product of the generator matrix and all data blocks.

unit for encoding, decoding and repairing, and it contains n chunks over n storage nodes, one chunk in each node. If the code is systematic, n chunks can be partitioned into k data chunks equally divided from the original data and m parity chunks that store the parity information $(n = k + m)$ of data chunks. We let D_i/C_i denote the i-th data/parity chunk. Each chunk is a set of r contiguously stored blocks of equal size. When $r = 1$, a chunk contains only one block, which is the case in the RS codes. When $r > 1$, a chunk is a set of r blocks and $D_{i,j}/C_{i,j}$ denotes the j-th data/parity block in the i-th data/parity chunk, as the case in the CRS codes.

B. *Encoding with the Generator Matrix*

Each linear code is uniquely represented by a generator matrix, which defines how data blocks in a stripe are encoded into parity blocks [8]. Encoding is the process of calculating the matrix product of the generator matrix and all data blocks, as pictured in Fig. 4. Since there are kr data blocks and mr parity blocks in a stripe, a generator matrix of the code has mr rows and kr columns. Each row vector corresponds to a parity block, which can be written as a combination of all data blocks with entries in the row vector as coefficients. Entries of the generator matrix are symbols over a Galois field GF(2^w), so their values are between 0 and $2^w - 1$. When $w = 1$, the code is an XOR code with each entry being either 0 or 1.

C. State-of-the-art Repair-bandwidth-efficient Codes

The Regenerating codes are a class of codes that can theoretically achieve the theoretical lower bound trade-off curve in Fig. 1, and they can be realized by random linear codes [1]. For example, the FMSR codes [5] are a class of the Regenerating codes and achieve the optimal repair bandwidth under minimum storage for any single node failure. However, two main drawbacks make them unsuitable for distributed storage systems. First, the repairs of the FMSR codes are functional, so the lost data are not really restored, instead some alternative data are regenerated to maintain the data redundancy, which makes repairs of the FMSR codes undependable. Second, the FMSR codes are not systematic, which means that the original data are altered after encoding and a read request to the data will require a decoding operation that involves extra data access and computation [5].

Besides the Regenerating codes, other repair-bandwidth-efficient codes have been proposed recently, such as the LRC, SRC and DRESS codes [9]. However, these codes trade the valuable storage resources for limited repair bandwidth reduction. The SRC store extra data in each storage node [3] and the LRC [4] introduce local parity data that confines the repair locally , resulting in additional storage resource consumption. The DRESS codes are based on replication in which at least two replicas are kept for each block, so that the code rate is always less than $\frac{1}{2}$ as the case in the PMSR codes [2].

Two factors guarantee the optimal repair bandwidth of the Z codes. First, all $n-1$ remaining nodes participate in a repair; Second, each of the $n-1$ remaining nodes only needs to selectively access and transfer blocks at a rate of $\frac{1}{m}$. Therefore, the amount of data required in each node is $\frac{M}{mk}$ and the repair bandwidth is $(n-1)\frac{M}{mk}$, exactly the same as the optimal repair bandwidth in Equation (1). In Table I, we compare the storage overhead and the repair bandwidth for representative repair-bandwidth-efficient codes with $m=2$ and $k=10$. It indicates that the Z codes have the least storage overhead, the same as the RS codes and FMSR codes, and the least repair bandwidth, the same as the FMSR codes. Moreover, the Z codes are systematic codes and also applicable for any parameter set (m,k) as long as $k>1$ and $m>1$.

TABLE I: Storage overhead for storing 1 TB data and repair bandwidth for reconstructing 1 TB data for codes with $m=2$ and $k=10$. (r and f are configuration parameters)

Codes	Storage Overhead	Repair Bandwidth	Computation Overhead	Systematic
3-Replicas	3 TB	1 TB	Very low	Yes
RS	1.2 TB	1 TB	Medium	Yes
SRC ($f=5$)	1.44 TB	0.6 TB	Medium	Yes
LRC ($r=5$)	1.3 TB	0.5 TB	Medium	Yes
RRS	1.2 TB	0.75 TB	High	Yes
FMSR	1.2 TB	0.45 TB	Very High	No
Z code	1.2 TB	0.45 TB	Low	Yes

III. Z Codes

We introduce the construction of the Z codes inductively and present their encoding, repairing and decoding. To facilitate these detailed description of the Z codes, we list some notations of erasure codes and their definitions in Table II.

TABLE II: Notations and their definitions

Notations	Meaning
k/m	Number of data/parity chunks in a stripe
n	Number of chunks in a stripe ($n=m+k$)
r	Number of blocks in a chunk
f	Index number of the faulty data chunk
D_i	The i-th data chunk ($0 \le i < k$)
C_i	The i-th parity chunk ($0 \le i < m$)
$D_{i,j}$	The j-th data block of data chunk D_i ($0 \le j < r$)
$C_{i,j}$	The j-th parity block of parity chunk C_i ($0 \le j < r$)
P	The generator matrix of an (m,k)-Z code
Q	The generator matrix of an $(m,k+1)$-Z code
$P_{i,j}$	submatrix of P ($0 \le i < k$, $0 \le j < m$)
P_i	The i-th block row of P, $P_i=[P_{i,0},\ldots,P_{i,k-1}]$
M	Original data size (total size of all data chunks)

A. Code construction

1) *Z Codes with* $k=2$: The generator matrix of an $(m,2)$-Z code is an m^2-row and $2m$-column bit matrix, which can be partitioned into $2m$ $m\times m$ blocks as follows, where $r=m$:

$$P=\begin{bmatrix} P_{0,0} & P_{0,1} \\ P_{1,0} & P_{1,1} \\ \vdots & \vdots \\ P_{m-1,0} & P_{m-1,1} \end{bmatrix}_{m\times 2} = \begin{bmatrix} I_m & I_m \\ I_m & I_m^{+} \\ \vdots & \vdots \\ I_m & I_m^{(m-1)+} \end{bmatrix}_{m\times 2} \quad (2)$$

I_m is an $m\times m$ identity matrix, and $I_m^{i+}(i<m)$ is an $m\times m$ permutation matrix representing i steps' left cyclic shift from I_m. We call I_m^{i+} *cell matrix*, since it is the fundamental submatrix of P. The number of different *cell matrices* is m. For example, when $m=3$, all *cell matrices* are as follows.

$$I_3=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\ I_3^{1+}=\begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix}\ I_3^{2+}=\begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix}$$

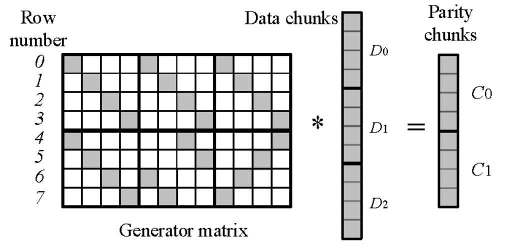
2) *Z Codes with an arbitrary* k: We first present a inductive construction method of a Z code that generates the generator matrix of an $(m,k+1)$-Z code from the generator matrix of an (m,k)-Z code. Let P and Q denote generator matrices of an (m,k)-Z code and an $(m,k+1)$-Z code separately, a two-step method of constructing Q from P is as follows:

1) Replicate the last block column of P and tile it onto the P along the horizontal dimension. The result is denoted as an intermediate matrix T.

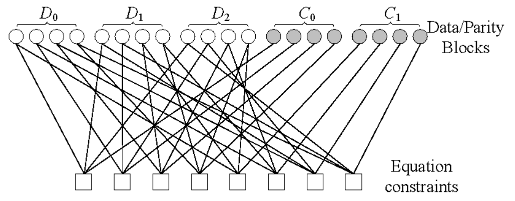
$$T=\begin{bmatrix} P_{0,0} & \cdots & P_{0,k-1} & P_{0,k-1} \\ P_{1,0} & \cdots & P_{1,k-1} & P_{1,k-1} \\ \vdots & \vdots & \vdots & \vdots \\ P_{m-1,0} & \cdots & P_{m-1,k-1} & P_{m-1,k-1} \end{bmatrix}_{m\times(k+1)}$$

2) Construct Q from T as follows, where each submatrix $Q_{i,j}$ is a tensor product of $T_{i,j}$ and a *cell matrix*.

$$Q=\begin{bmatrix} Q_{0,0} & \cdots & Q_{0,k-1} & Q_{0,k} \\ Q_{1,0} & \cdots & Q_{1,k-1} & Q_{1,k} \\ \vdots & \vdots & \vdots & \vdots \\ Q_{m-1,0} & \cdots & Q_{m-1,k-1} & Q_{m-1,k} \end{bmatrix}_{m\times(k+1)}$$



(a) Encoding with generator matrix. Note that shaded boxes in the generator matrix represent 1s and others are 0s and we follow the same plotting convention in the rest of the paper.



(b) Tanner graph

Fig. 5: Generator matrix encoding of $(2,3)$-Z code (a) and its tanner graph expression (b).

$$\text{and} \begin{cases} Q_{i,j} = T_{i,j} \otimes I_m, \; i = 0, \cdots, m-1, j = 0, \cdots, k-1 \\ Q_{i,k} = T_{i,k} \otimes I_m^{j+}, \; i = 0, \cdots, k-1 \end{cases}$$

where $\otimes$ denotes the tensor product, which is also called Kronecker product [10]. Since we have already introduced the generator matrix's construction of an $(m,2)$-Z code, we can construct the generator matrix for any (m,k)-Z code inductively. For an (m,k)-Z codes, the number of blocks in a chunk (r) is a function of m and k, $r = m^{k-1}$. The generator matrix P actually has a general expression as Equation (3), which can be deduced from the above inductive construction.

$$\begin{aligned} &P = [P_{i,j}]_{m \times k}, \quad P_{i,j} = (I_m^{i+})^{\otimes j} \otimes (I_m)^{\otimes(k-1-j)} \\ &\text{with } 0 \le i < m, \; 0 \le j < k \end{aligned} \tag{3}$$

where i, j are the block row index number and the block column index number respectively, and the power $(I_m^{i+})^{\otimes j}$ of matrix I_m^{i+} denotes the tensor product of j copies of I_m^{i+}. Each submatrix $P_{i,j}$ is a tensor product of $k-1$ cell matrices that are all permutation matrices, so $P_{i,j}$ is also a permutation matrix. Thus, the numbers of 1s in each row and each column of generator matrix P are k and m respectively.

B. Encoding

Encoding is the procedure of generating m parity chunks by calculating the matrix product of the generator matrix P and data blocks, which is exactly the same as the encoding of other matrix-based codes [8], such as RS codes and CRS codes. We illustrate an encoding example of the $(2,3)$-Z code in Fig. 5a and its tanner-graph expression in Fig. 5b. A Tanner graph is a bipartite graph that is commonly used for representing the LDPC codes [11]. There are two distinctive sets of vertices in a tanner graph, namely, vertices representing data and parity blocks and those representing equation constraints, which are connected by edges. An equation constraint is connected to some blocks by edges in the graph, whose summation (XOR) is equal to 0. *Each equation constraint in a tanner graph corresponds to a row vector in the generator matrix of the same code.* For example, that an equation constraint connects to nodes of $D_{0,0}$, $D_{1,2}$, $D_{2,3}$ and $C_{1,0}$ represents the equation:

$$D_{0,0} \oplus D_{1,2} \oplus D_{2,3} \oplus C_{1,0} = 0$$

where $\oplus$ denotes XOR. This equation constraint corresponds to the 4-th row vector of the generator matrix of the $(2,3)$-Z code, and the product result of the row vector and all data blocks is the parity block $C_{1,0}$.

C. Repairing

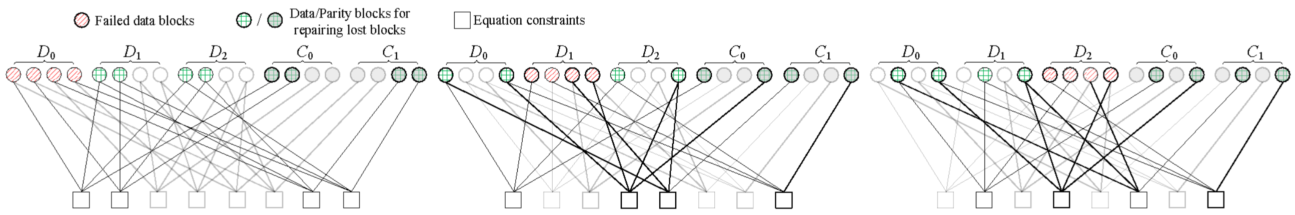
We focus on the case of a single data node's failure [1], which in the paper refers in particular to the failure of a data node that stores the data information, excluding the failure of a parity node. We make this trade-off in the consideration of different reconstruction priority of a parity chunk and a data chunk. If a parity chunk is lost, data chunks of the same stripe are still readable, which allows the reconstruction of the parity chunk to be done at a later time or in the background. This is generally not allowed for a lost data chunk, because once a data chunk is lost, a read request to the lost data causes a degraded read in which the lost data must be reconstructed as quickly as possible to satisfy the read request.

With the help of the tanner graph of a Z code, we introduce a four-step method to determine $\frac{r}{m}$ blocks of each chunk in a node for repairing a faulty data chunk D_f as follows. (1) Select any one block of D_f and any one of the equation constraints the block is connected with, then remove other equation constraints the block is also connected with; (2) For each removed equation constraint, remove all blocks it is connected with, except for the data blocks from D_f; (3) For each removed block, remove all equation constraints it is connected with; (4) Repeat (2) and (3) until no more blocks and equation constraints can be removed. The remaining blocks are the blocks required to repair the faulty node, with the remaining equation constraints as equations to solve all data blocks of D_f.

Taking the $(2,3)$-Z code as an example, we plot the $(2,3)$-Z code's optimal repairs for the failure of each data node in Fig. 6. For a better view, we faded the colors of the data/parity blocks and equation constraints that would not be used for repairing. To repair the faulty data chunk D_f $(f = 0,1,2)$, each remaining chunk contributes 2 blocks, so the repair bandwidth is 8 blocks. These 8 selected blocks and 4 lost blocks ($D_{f,0}$, $D_{f,1}$, $D_{f,2}$ and $D_{f,3}$) are associated with 4 equation constraints, which are only associated with these 8 selected blocks and 4 lost blocks. Thus, D_f can be repaired, since each lost data block is reconstructed by solving an equation constraint with the lost block as the only unknown argument. Compared with the RS or CRS codes that require all blocks from each of $k = 3$ chunks, the $(2,3)$-Z code saves the repair bandwidth by an amount of $3 \times 4 - 8 = 4$ blocks.

Because of the length limit of the paper, we omit the theoretical proof of the optimal repair bandwidth of the Z

[1]We use the phrases "failure of a data node" and "failure of a data chunk" interchangeably, since "failure of a data node" causes a subsequent repair by rebuilding all chunks of the failed node.



(a) repair data chunk D_0 (b) repair data chunk D_1 (c) repair data chunk D_2

Fig. 6: Tanner graphs representing repairs of D_0, D_1 and D_2 with the optimal repair bandwidth for the $(2,3)$-Z code.

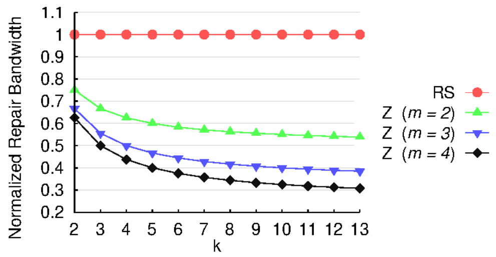


Fig. 7: Repair bandwidth under a single data node failure of the Z codes as a function of m and k, normalized to that of the RS codes, where $m = 2 \sim 4$ and $k = 2 \sim 14$.

codes for a single data node failure and the readers can prove it in a inductive way.

IV. Analytical Evaluations

We analyze the storage efficiency and repair bandwidth of the Z codes in this section. An (m,k)-Z code consumes the minimum storage capacity, equal to that of a RS code with the same parameter set (m,k). That is, for each stripe over n nodes, the Z codes store a data chunk of size $\frac{M}{k}$ in each node.

Under the minimum storage usage, the Z codes achieve the optimal repair bandwidth for a single data node failure, as given in Equation (1). Fig. 7 plots the repair bandwidth under a single data node failure of the Z codes as a function of k and m, normalized to that of the RS codes, which is calculated as the ratio of the amount of data required for repairing a data chunk to the amount of the original data in a stripe M. Compared to the RS codes, which must retrieve the whole chunk from any k remaining nodes with a repair bandwidth of M, the amount of repair bandwidth saved by the Z codes is

$$k\frac{M}{k} - (n-1)\frac{1}{m}\frac{M}{k} = \frac{(m-1)(k-1)}{mk}M.$$

For the same m value, the Z codes with a larger k value requires less repair bandwidth, while the Z codes with a larger m value requires less repair bandwidth for the same k value. In addition, the repair of a parity chunk by the Z codes costs the same repair bandwidth as the RS codes, with no repair bandwidth saved, since repairing a parity chunk is exactly the same process of re-encoding the parity chunks with all data chunks being involved.

In Fig. 8, We plot the normalized repair bandwidth of the Z codes and two other repair-bandwidth-efficient codes, the FMSR codes and RRS codes. The X-axis shows the storage overhead, defined to be the ratio of parity size and the original data size. With the growth of k, the proportion of the parity

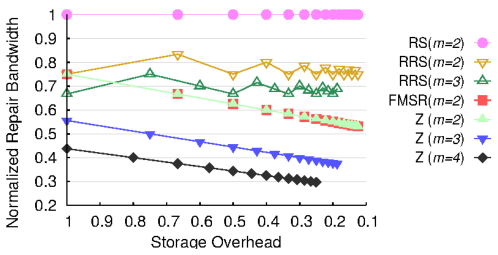


Fig. 8: Normalized repair bandwidth vs. storage overhead of the RS codes, FMSR codes, RRS codes and Z codes. From the left to the right, k increases from m to 16 and the storage overhead also decreases.

size in a stripe decreases, and the storage overhead decreases as well. When $m = 2$, the Z codes can achieve the same normalized repair bandwidth as the FMSR codes with the same storage overhead, while the Z codes can save more repair bandwidth as the m value increases.

V. Experimental Evaluations

We measure the coding performance of Z codes in a single machine with 3.5GHz Intel Core i7-4770K CPU.

We empirically evaluate the in-memory encoding and repairing speeds of several codes, which are measured by the amount of data encoded or rebuilt per second based on the following equation:

$$\text{Encoding Speed} = \frac{\text{Size of data to encode}(M)}{\text{Time cost}}$$

$$\text{Repairing Speed} = \frac{\text{Size of data to reconstruct}(\frac{M}{k})}{\text{Time cost}}$$

The baseline codes for the Z codes are the RS, CRS and FMSR codes. The RS and CRS codes are two most commonly used erasure codes with first-rank coding speed. Existing repair-bandwidth-efficient codes are constructed over the RS or CRS codes, so the performances of the former are usually worse than those of the latter [5]. Our encoding/repairing speed evaluation method is exactly the same as that of the SD codes [12] and STAIR codes [13]. For the RS and FMSR codes, their generator matrices are generated over the Galois field GF(2^8), so encoding and repairing are over the same field. The fast Galois Field arithmetic library GF-complete provided by Jim Plank et al. [14] is leveraged for fast Galois field arithmetic.

The experiments cover the situations where $m = 2, 3, 4$, which are the most common cases of failure tolerance. For each code, we test all combinations of (m,k)s with the restrictions of $r < 7000$ to avoid excessively large generator

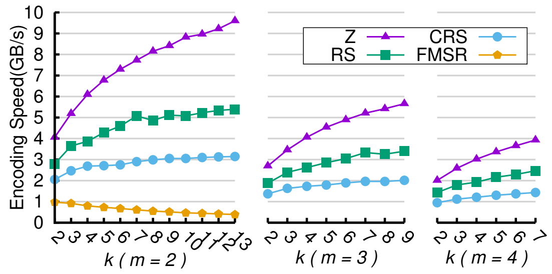


Fig. 9: Encoding speed of the Z codes, CRS codes, RS codes and FMSR codes for a 1-GB file.

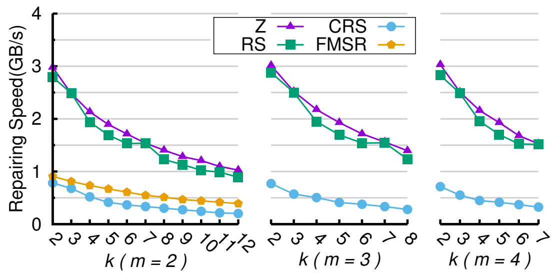


Fig. 10: Repairing speed of the Z codes, CRS codes, RS codes and FMSR codes for a 1-GB file.

matrices. We encode/repair a region of data with size up to 1 GB so that the size of a block would not be too small. Each test is run 50 times with the average speed as the final result. Encoding speeds are plotted in Fig. 9 where encoding speeds of these four codes have the following relationship:

$$\text{Z codes} > \text{RS codes} > \text{CRS codes} > \text{FMSR codes}$$

Actually, The RS codes and Z codes share the same encoding complexity on account of the fact that each parity block is a combination of k data blocks, which can be observed from the construction of these two codes. Hence, Z codes gain the fastest encoding speed benefiting from the fast bit-wise XOR.

Fig. 10 plots the speed of repairing a single data chunk of 1-GB. We only measure the speed of repairing a data chunk for the RS, CRS and Z codes, since repairing a parity chunk is exactly the same as the process of re-encoding it. For all codes, their repairing speeds exhibit a decreasing trend as k increases, since more data blocks are involved in reconstructing the same size of data, incurring more computation. Although repair bandwidth consumptions of Z codes and RS codes are different, they perform the similar repairing performance. Z codes offer a speed on the order of GB/s, which suggests that computation would no longer be the performance bottleneck when applying the Z codes in a practical storage system.

VI. Conclusions

We propose a family repair-bandwidth-efficient erasure codes, called the Z codes for distributed storage systems. The Z codes not only achieve the optimal repair bandwidth for repairing a single data node's failure, but also attain the minimum storage property as the RS and CRS codes. The Z codes have many other beneficial properties that make them suitable for distributed storage systems. The Z codes have comparable encoding and repairing performances with the RS codes, and significantly outperform the CRS codes and FMSR codes. Reducing the number of blocks in a chunk (r) and finding new constructions to enable the Z codes to tolerate more concurrent failures are the directions of our future research on the Z codes.

Acknowledgement

The authors would like to thank Jian Li for helping to implement the Z codes in a practical storage system. This work is supported in part by National Basic Research Program of China (973 Program) (2011CB302301); National Natural Science Foundation of China (61025008, 61232004, 61173043); National High Technology Research and Development Program of China (863 Program) (2013AA013203); National Key Technology R&D Program of China (2011BAH04B02); The Fundamental Research Funds for the Central Universities (2013TS043); Natural Science Foundation of Hubei Province (2015CFB192).

References

[1] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *Information Theory, IEEE Transactions on*, vol. 56, no. 9, pp. 4539–4551, 2010.

[2] K. Rashmi, P. Nakkiran, J. Wang, N. B. Shah, and K. Ramchandran, "Having your cake and eating it too: Jointly optimal erasure codes for i/o, storage, and network-bandwidth," in *Proc. of USENIX FAST*, 2015, pp. 81–94.

[3] D. S. Papailiopoulos, J. Luo, A. G. Dimakis, C. Huang, and J. Li, "Simple regenerating codes: Network coding for cloud storage," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 2801–2805.

[4] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *USENIX ATC*, 2012.

[5] H. C. Chen, Y. Hu, P. P. Lee, and Y. Tang, "Nccloud: a network-coding-based storage system in a cloud-of-clouds," *Computers, IEEE Transactions on*, vol. 63, no. 1, pp. 31–44, 2014.

[6] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: Minimizing i/o for recovery and degraded reads," in *Proc. of USENIX FAST*, 2012.

[7] I. Tamo, Z. Wang, and J. Bruck, "Zigzag codes: Mds array codes with optimal rebuilding," *Information Theory, IEEE Transactions on*, vol. 59, no. 3, pp. 1597–1616, 2013.

[8] J. S. Plank, "T1: erasure codes for storage applications," in *Proc. of USENIX FAST*, 2005, pp. 1–74.

[9] S. Pawar, N. Noorshams, S. El Rouayheb, and K. Ramchandran, "Dress codes for the storage cloud: Simple randomized constructions," in *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 2338–2342.

[10] C. F. Van Loan, "The ubiquitous kronecker product," *Journal of computational and applied mathematics*, vol. 123, no. 1, pp. 85–100, 2000.

[11] R. Tanner, "A recursive approach to low complexity codes," *Information Theory, IEEE Transactions on*, vol. 27, no. 5, pp. 533–547, Sep 1981.

[12] J. S. Plank, M. Blaum, and J. L. Hafner, "Sd codes: Erasure codes designed for how storage systems really fail," in *Proc. of USENIX FAST*, 2013.

[13] M. Li and P. P. Lee, "Stair codes: a general family of erasure codes for tolerating device and sector failures in practical storage systems." in *FAST*, 2014, pp. 147–162.

[14] J. S. Plank, K. M. Greenan, and E. L. Miller, "Screaming fast galois field arithmetic using intel simd instructions," in *FAST-2013: 11th Usenix Conference on File and Storage Technologies, San Jose*, 2013.