AZ-Code: An Efficient Availability Zone Level Erasure Code to Provide High Fault Tolerance in Cloud Storage Systems

Xin Xie[1], Chentao Wu[1*], Junqing Gu[1], Han Qiu[1], Jie Li[1], Minyi Guo[1], Xubin He[2], Yuanyuan Dong[3], and Yafei Zhao[3]

[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China

[2]Department of Computer and Information Sciences, Temple University, Philadelphia, United States

[3]Alibaba Group, Hangzhou, China

*Corresponding Author: wuct@cs.sjtu.edu.cn

Abstract—As data in modern cloud storage system grows dramatically, it's a common method to partition data and store them in different Availability Zones (AZs). Multiple AZs not only provide high fault tolerance (e.g., rack level tolerance or disaster tolerance), but also reduce the network latency. Replication and Erasure Codes (EC) are typical data redundancy methods to provide high reliability for storage systems. Compared with the replication approach, erasure codes can achieve much lower monetary cost with the same fault-tolerance capability. However, the recovery cost of EC is extremely high in multiple AZ environment, especially because of its high bandwidth consumption in data centers. LRC is a widely used EC to reduce the recovery cost, but the storage efficiency is sacrificed. MSR code is designed to decrease the recovery cost with high storage efficiency, but its computation is too complex.

To address this problem, in this paper, we propose an erasure code for multiple availability zones (called AZ-Code), which is a hybrid code by taking advantages of both MSR code and LRC codes. AZ-Code utilizes a specific MSR code as the local parity layout, and a typical RS code is used to generate the global parities. In this way, AZ-Code can keep low recovery cost with high reliability. To demonstrate the effectiveness of AZ-Code, we evaluate various erasure codes via mathematical analysis and experiments in Hadoop systems. The results show that, compared to the traditional erasure coding methods, AZ-Code saves the recovery bandwidth by up to 78.24%.

Index Terms—Erasure Codes, Availability zone, Reliability, Cloud Storage, Performance Evaluation

I. INTRODUCTION

Nowadays in public cloud storage systems [28] such as Amazon AWS [3], Microsoft Azure [8], Google Cloud [21] [14] [10] [7], Alibaba Cloud, a large amount of data are stored in multiple Availability Zones (AZs), where AZ is a general configuration. Typically, an AZ is a physically isolated area in data centers, which consists of several racks of servers, or one floor of a data center, or even a whole data center. By using multiple AZs, cloud storage systems can have the following benefits,

- **High availability.** Typically, data are distributed among multiple physically isolated storage infrastructures or data centers. When one AZ fails, the remaining AZs can provide ordinary cloud services as well.
- **High reliability.** Redundant data are provided among different AZs, which can tolerate large scale of failures such as natural disasters (i.e., earthquakes, tsunamis), massive power outage, etc.

- **Low network latency.** The users connect to the nearest geographical availability zone which can decrease network latency.

Typically, data redundancy methods (i.e., replication and erasure coding) are widely utilized in availability zones (AZs) to obtain various capabilities of fault tolerance [8] [33] [23] [30]. Traditional replication or backup strategies can easily guarantee high reliability but the storage cost is extremely high. In cloud storage systems like Microsoft Azure [8], Google cloud [21], Facebook [1] and Alibaba Cloud, erasure codes (EC) are chosen to store the cold data in general. In this use case, erasure codes can support both high reliability with low monetary cost. In the last decade, popular erasure codes can be categories into two types, RS-based codes [31] [17] [36] and XOR-based codes [4] [9] [41] [38] [16] [37] [34] [5] [40] [27] [44] [12] [45]. Traditionally, RS-based codes (e.g., RS [31], LRC [17]) are generated via Galois Field computations, and XOR-based calculations are operated in XOR-based codes (e.g., STAR code [16], TIP-Code [44], Butterfly codes [24] [13]).

However, existing erasure codes have several drawbacks in availability zone (AZ) environment. First, for XOR-based codes, most XOR-based erasure codes [9] [4] only tolerate two or three node failures, which cannot meet the requirements of tolerating AZ level node errors. Some XOR-based codes can tolerate more than three nodes, but the scalability is rather poor [44] [16]. Second, For RS-based codes, although they can provide high scalability, the recovery cost (e.g., the reconstruction I/Os, the recovery bandwidth) is extremely high, which attracts many researchers to propose new solutions [22] [17] [29]. Third, Minimal Storage Regenerating code [36] [42] is efficient to reduce recovery overhead, but the computational complexity is much higher than RS-based and XOR-based codes.

To avoid the above drawbacks, in this paper, we propose a novel erasure code called AZ-Code, which is an efficient code for multiple availability zones environment. AZ-Code take advantages of both RS and MSR codes, which can provide both high reliability and scalability with low monetary and recovery cost.

The contribution of our work includes:

1) We propose AZ-Code to provide low recovery cost with high reliability for AZ environment.

2) A series of experiments show that comparing with the traditional approaches, AZ-Code achieves better performance of recovery in AZ environment.

The rest of the paper is organized as follows. In Section II, we introduce related work and our motivation. In Section III, the design of AZ-Code and corresponding encoding and decoding techniques are illustrated in detail. The evaluation is presented in Section IV and the conclusion of our work is in Section V.

II. RELATED WORK AND MOTIVATION

In this section, we discuss the requirements of erasure codes for AZ environment in cloud storage systems and give a brief introduction about the existing erasure codes. To facilitate our discussion, we summarize the symbols used in this paper in Table I.

TABLE I
SYMBOLS USED IN THIS PAPER

Symbols	Description
n	the number of nodes in a node array
r	the number of parity nodes
k	the number of data nodes ($k = n - r$)
l	code group length
s	a parameter of MSR code ($n = s^m$)
m	a parameter of MSR code ($n = s^m$)
z	the number of AZs(the group of AZ-Code)
p	the number of local parity nodes
g	the number of global parity nodes
d	the number of helper nodes
α	bits stored on a node
β	bits downloaded from one node in recovery process
τ	the total recovery bits
δ	the radio form of recovery cost
Θ	an MSR code
C_i	the nodes of MSR code
A_i	a l*l coefficient matrix
D	a vector consists of all data
D_L	a vector consists of lost data
D_S	a vector consists of surviving data
H	a parity-check matrix
H'	a decoding matrix of MSR code
H_L	a sub-matrix of H (columns corresponds to the lost data)
H_S	a sub-matrix of H (columns corresponds to the surviving data)
P	a local parity node of AZ-Code
G	a global parity node of AZ-Code
J	a decoding matrix of AZ-Code

A. Requirements of Erasure Codes in Multiple Availability Zones (AZs)

In modern large-scale cloud storage systems, large datasets are normally divided into several parts and then stored in different AZs. In general, the following aspects of erasure codes are desired in multiple AZs environment,

- *High Reliability*: Erasure codes can provide high reliability, which can tolerate concurrent AZ level failures (e.g., lost a whole AZ). In this paper *Fault Tolerance* is defined as the maximum number of r node failures that a code can tolerate to evaluate reliability.
- *High Storage Efficiency*: Typically, a node array consists of k data nodes and r parity nodes, the storage efficiency

is $k/(k + r)$. In AZ environment, we need to make the storage efficiency as high as possible.
- *Low Recovery Cost*: In data centers, node failure is common and recovery cost is a significant problem [33]. As we know, the recovery cost include the computation cost, network bandwidth, and the total number of I/Os. An erasure code with low recovery cost is a popular choice in cloud storage systems [17]. In this paper we use *Reconstruction Cost* defined as the number of nodes required to reconstruct the lost nodes to evaluate the network bandwidth of recovery cost.
- *High Scalability*: Erasure codes can provide fast scaling processes when scale up or scale out [39].

B. Existing Erasure Codes in Cloud Storage Systems

1) RS Code: *Reed Solomon Code* (RS Code) [26] was proposed by *Irving S. Reed* and *Gustave Solomon* in 1960. RS code is a kind of *Maximum Distance Separable (MDS) Codes*, which have the optimal storage efficiency. The encoding and decoding operations of RS code are based on *Galois Field*, which leads to a higher computational complexity comparing to XOR-based codes. However, due to its high scalability, RS code has been widely applied in traditional cloud storage systems. In a RS code which is delegated by RS(k, r), $n = k + r$ denotes the total number of nodes[1] participating in the erasure coding schema, k stands for the number of data nodes, and r is the number of parity nodes. Generally data is organized and encoded/decoded with the minimum coding unit **block**[2]. $RS(k, r)$ can tolerate at most r failures at the same time, and single node failure can be recovered from any k survivors. The encoding process of RS code is given in Fig.1.
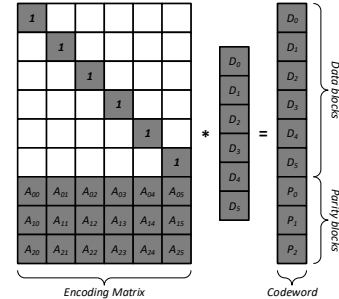

Fig. 1. The encoding process of $RS(6, 3)$. The leftmost matrix is called Encoding Matrix, which encodes data nodes ($D_0, D_1, \cdots, D_5$) into Codeword ($D_0, D_1, \cdots, D_5, P_0, P_1, P_2$).

RS Code can be applied in multiple AZs. In general, all nodes are evenly distributed among all AZs. An example shows in Fig.2, where $RS(6, 6)$ can be applied in 3AZ environment, and $RS(4, 4)$ can be applied in 2AZ environment.

The reconstruction process of RS code is shown in Fig.3. In this figure, k surviving data nodes are used to recover the

[1]In this paper, a node represents a storage medium which is physically isolated from other storage mediums.

[2]In this paper, a *block* is used to represent a *data element* or *chunk* which is the basic access unit in erasure codes [6] [32].
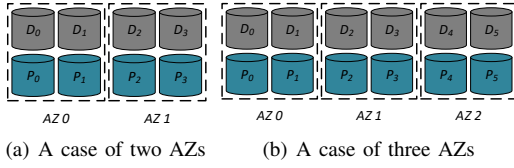
(a) A case of two AZs   (b) A case of three AZs

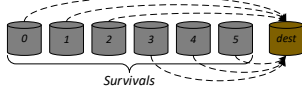Fig. 2. Two examples show an $RS(6,6)$ applied in 3AZ environment and an $RS(4,4)$ in 2AZ environment.

Fig. 3. The reconstruction of single node failure in $RS(6,3)$. Data of 6 surviving nodes should be transmitted to the failure node named *dest*.

lost data, which causes great latency and seriously affects the performance of cloud storage systems. To solve this problem, many methods have been proposed (introduced in Section II-C).

2) LRC Code: Local Reconstruction Codes (LRC) [17] and *Locally Repairable Codes* [25] are similar codes, and applied in Windows Azure Storage and Facebook cloud, respectively. Since both of them are based on a similar idea, here we mainly discuss *Local Reconstruction Codes*, and the abbreviation LRC is used to refer to it. LRC is a kind of non-MDS code.

A LRC can be represented by $LRC(k,z,r)$, where k, z, r are denoted by the number of data nodes, the number of local parity nodes, the number of parity nodes, respectively. Besides, k data nodes are divided into z groups, and one local parity node is added to each group. Fig.4 shows an example.
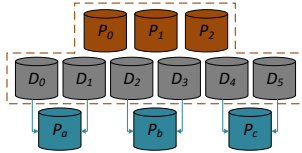

Fig. 4. An example of $LRC(6,3,3)$. P_0,P_1 and P_2 are global parities of data nodes $(D_0, D_1, \cdots, D_5)$. P_a, P_b and P_c are local parities, which are generated via data node groups (D_0, D_1), (D_2, D_3) and (D_4, D_5) respectively.
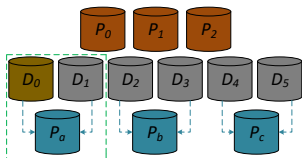

Fig. 5. The reconstruction of single failure in $LRC(6,3,3)$. The failure node D_0 is marked as brown. Only data of (D_1, P_a) in green box need to be transmitted to D_0.

It is obvious that LRC sacrifices more storage redundancy than RS Code for extra local parities, which completes single failure recovery in the local group and improves the performance of failure recovery. Specially, when LRC is applied in multiple AZs (shown in Fig.6), each group is equivalent to an

AZ, and global parity nodes are uniformly assigned to each AZ.

For reconstruction of a lost data node, the failure node only need to obtain the data of nodes in the same group. An example shows in Fig.5.
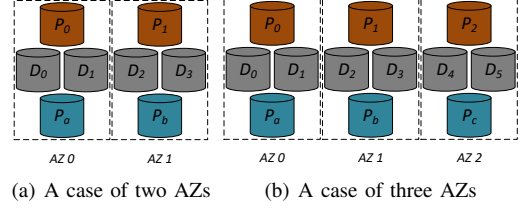

(a) A case of two AZs   (b) A case of three AZs

Fig. 6. Two examples show an $LRC(6,3,3)$ applied in 3AZ environment and an $LRC(4,2,2)$ applied in 2AZ environment.

3) XOR-based Codes: In the last two decades, several XOR-based codes are proposed for cloud storage systems, typical XOR-based codes include STAR [16], Triple-STAR [37], HDD code [35], TIP-Code [44], which can tolerate concurrent node failures of any triple nodes. Under multiple AZ environment, GRID code [19] is a feasible solution for a large amount of node failures. However, due to the limitation of multi-dimensional XOR-based parity generation, the scalability issue [18] is a big obstacle in cloud storage systems.

C. Optimization Methods for Reconstruction

In recent years, a number of methods to improve the performance of Erasure Code have been proposed. For example, Partial-Parallel-Repair [22] and Pipeline-Repair [20] reduce the network latency, but do not reduce the total amount of data transmitted in cloud storage system. Other methods like Parity-Check Matrix [43] reduce the complexity of decoding, but do not improve the network transmission.

Minimum-Storage Regenerating (MSR) code [11] [15] [36] [42] [24] [13] is one special case of regenerating code with the property of Maximum Distance Separable (MDS) codes. It is expressed as the tuple (n,k,l) which means n data nodes and parity nodes, k data nodes and sub-packetization with size l. MSR code is sufficient for reducing the recovery cost, but its construction is too complex and the computation cost is high in AZ environment.

D. Our Motivation

We summarize existing erasure codes in AZ environment in Table II which shows that the existing erasure codes are insufficient for AZ environment. RS code can provide high reliability but its recovery cost is also high which has been the major issue in AZ environment. LRC code can reduce the recovery cost to some extent but increase the storage cost. MSR code can achieve optimal recovery cost without increasing the storage cost but its complexity is too high.

In summary, existing erasure codes are insufficient in AZ environment, which motivates us to propose a new code construction called AZ-Code.

TABLE II
SUMMARY ON VARIOUS ERASURE CODES IN AZ ENVIRONMENT

Name	Reliability	Scalability	Storage Efficiency	Recovery Cost	Computational Complexity
RS Code	high	high	high	high	high
LRC Code	high	high	low	high	medium
MSR Code	high	medium	high	low	very high
GRID Code	high	low	high	medium	low
AZ-Code	high	high	low	low	medium

III. AZ-CODE

In this section, we first give the construction of the Availability Zone Level Erasure Code (AZ-Code) and then introduce the construction of local parities and global parities, encoding and decoding processes of AZ-Code. We also summarize the properties of AZ-Code at the end of the section.

A. Overview of AZ-Code

Generally we present AZ-Code as $AZ(k, z, p, g)$ where k means the number of data nodes, z means the number of AZ. AZ-Code divides k data nodes into z AZs with k/z data nodes in each AZ. It generates p local parity nodes by MSR code method for each AZ to reduce the recovery cost, and computes g global parity nodes from all k data nodes by RS code to provide high reliability. Let n be the total number of nodes (data + parity), thus $n = k + z * p + g$.

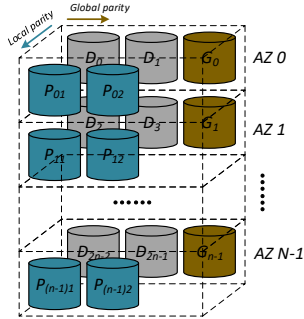The Figure 7 shows an example of AZ-Code with multiple AZs.

Fig. 7. The construction of AZ-Code. Local parities are stored in the corresponding AZ and global parities are evenly distributed among multiple AZs.

1) Local Parity: We use MSR codes [24] [13] [11] [15] [36] [42] (MSR code is defined in a matrix form and introduced in Section III-C) to generate the local parity nodes. Typically, the local parity nodes are generated from the local data nodes and stored in a same AZ area.

2) Global Parity: We use RS code [26] to generate the global parities which are generated from all data nodes and evenly distributed among all AZs.

B. Case Studies of Multiple AZs

1) AZ-Code for Two AZs (2AZ-Code): We show an example of $AZ(4, 2, 2, 2)$ with 2AZs in Figure 8(a). Each AZ has one global parity node, two local parity nodes and two data nodes.
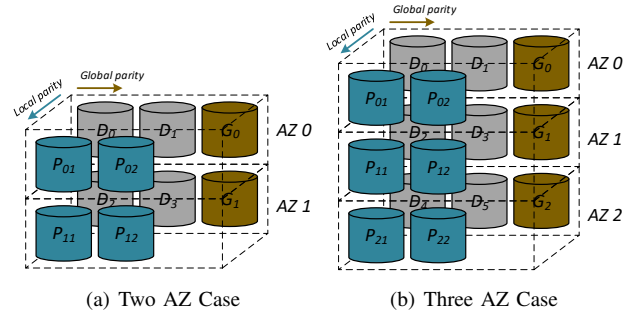

(a) Two AZ Case (b) Three AZ Case

Fig. 8. Two cases of AZ-Code.

- **Encoding Process**: local parity nodes are encoded by $MSR(4, 2, 4)$ and global parity nodes are encoded by $RS(4, 2)$.
- **Decoding Process**: 1) We prefer to use the local parity nodes to recover the lost data when the number of failure nodes is small. For example, when D_0 in Fig.8(a) fails, we use D_1, P_{01} and P_{02} to recover it. 2) When the number of failure nodes is large, we use both local and global parities to decode data. For example, when D_0, D_1, P_{01} fail, we use one local parity node P_{02}, two global parity nodes G_0, G_1 and two data nodes D_2 and D_3 to recover them. The recovery algorithm is shown in Section III-F.

2) AZ-Code for Three AZs (3AZ-Code): We show an example of $AZ(6, 3, 2, 2)$ with 3AZs in Figure 8(b). Here each AZ has one global parity node, two local parity nodes and two data nodes.

- **Encoding Process**: the local parities are encoded by $MSR(4, 2, 4)$ and the global parities are encoded by $RS(6, 3)$.
- **Decoding Process**: the same as two AZ case.

C. Construction of Local Parities

In this section, we introduce one kind of MSR codes [42] which is used to generate the local parity nodes. First we give the mathematical construction and then use one special case to explain the encoding/decoding process.

1) Overview of MSR Code: MSR code is expressed as a tuple(n, k, l), where n denotes the total number of data and parity nodes, k denotes the number of data nodes, and l means the minimum coding unit or sub-packetization [42]. Suppose that each node has α bits and when a node fails, the storage system attempts to repair their content by contacting d helper nodes[3] and downloading β bits from each helper node. The total recovery bits τ satisfies the equation $\tau = d * \beta$. It shows that [11] the recovery cost δ has the following equation,

$$\delta = \frac{\tau}{\alpha} = \frac{d * \beta}{\alpha} = \frac{d}{d - k + 1} \quad (k \le d \le n - 1) \quad (1)$$

δ is inversely proportional to d in Equation 1, so we can reduce the recovery cost by allowing more helper nodes to repair the

---

[3]helper nodes are the nodes which provide the data to reconstruct the lost data

lost nodes. When one node fails, d is set to $n-1$ and δ achieves the maximum value $\frac{n-1}{n-k}$. When multiple nodes fail, d is usually set to k and δ reaches the minimum value of k. At this time MSR code is equivalent to any MDS code like RS code. We show this relationship in the following formula,

$$k \leq \delta \leq \frac{n-1}{n-k} \qquad (2)$$

2) MSR Code Construction: Let Θ be a (n,k,l) MSR code with nodes $C_i \in F^{ln}, i = 1, 2, ..., n$ (data nodes and parity nodes are unified by C_i), where each C_i is a column vector with coordinates $C_i = \{c_{i1}, c_{i2}, ..., c_{il}\}^T$. To further reduce the construction cost, we use the Parity Check Matrix (PCM) [43] to construct the MSR part of the AZ-Code. The construction of code Θ is defined by the following r parity check equations,

$$\Theta = \{(C_1, C_2, ...C_n) | \sum_{i=1}^{n} A_{ti} * C_i = 0, \quad t = 0, 1, 2, ..., r-1\} \qquad (3)$$

In Equation 3, $A_{t,i} \in F$ is a $l * l$ matrix corresponding to $l * 1$ column vector C_i. The whole coefficient matrix H with the size of $(r * l) * (n * l)$ consists of the $r * n$ submatrix $A_{t,i}$. $A(a, b)$ is defined as the element in row a, column b of submatrix A, where $0 \leq a, b \leq l-1$. The specific structure of $A_{t,i}$ is shown as below.

Construction via Local Parities *Let $s, m \in F$ be positive integers with the constraint,*

$$s \leq r \leq sm, \quad n = sm, \quad l = s^m$$

We rewrite a to S-ary form, $a = (a_1, a_2, ..., a_m)$ and define an equation $a(v, u) := (a_1, ..., a_{v-1}, a_v = u, a_{v+1}, ..., a_m)$. Let F be a finite field with the size $|F| \geq n$. The parameter λ_i and γ follow the constraints,

$$\lambda_i \in F, i = 1, 2, ..., n.$$
$$\lambda_i \neq \lambda_j \quad if \ i \neq j.$$
$$\gamma \in F, \gamma \neq 0, 1.$$

$A_{t,i}(0 \leq t \leq r-1, 1 \leq i \leq n)$ *can be transformed to* $A_{t,(v-1)s+u+1}(0 \leq v \leq m-1, 0 \leq u \leq s-1)$. *And then, the elements $A_{t,(v-1)s+u+1}(a, b)$ are constructed by following function,*

$$A_{t,(v-1)s+u+1}(a, b) = \begin{cases} \lambda_{(v-1)s+u+1}^t & if \ a_v < u, a = b \\ \gamma \lambda_{(v-1)s+u+1}^t & if \ a_v > u, a = b \\ \lambda_{(v-1)s+w+1}^t & if \ a_v = u, a(v, w) = b \\ & with \ w = 0, 1, ..., s-1 \\ 0 & otherwise \end{cases}$$

For each submatrix $A_{t,i}$ which is well defined, we can get the whole coefficient matrix H.

The coefficient matrix H has a property that any $r * r$ sub-matrix of H is invertible (Here each submatrix $A_{t,i}$ is treated as a single element.). Due to space limitations, we don't provide the detailed proof, which can be found in the literature

[42]. This property guarantees that MSR code can tolerate any r nodes failure. Here we show the detailed construction of an MSR code in Fig.9 with the following configuration,

$$n = 4, \ k = 2, \ r = s = 2, \ m = 2, \ l = 4.$$

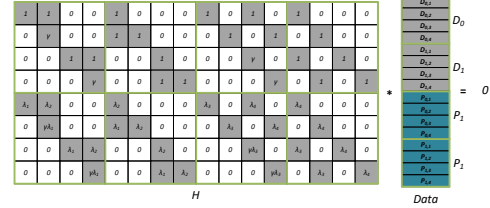We use this configuration to demonstrate the encoding and recovery processes of MSR code in the next sections.

Fig. 9. The detailed construction of MSR code with ($n = 4, k = 2, s = m = 2, l = 4$). H is a *coefficient matrix* based on finite field F. According to Equation 3, the MSR code is defined by total $r * l = 8$ parity check equations.

3) The Encoding Process of MSR Code: We demonstrate a brief construction in Fig.10, where each element $A_{i,j}$ is a $2 * 2$ matrix. In this condition we ignore the details in the matrix $A_{i,j}$, so there are total two parity check equations.

The encoding process of MSR code is based on PCM method and shown in Fig.11. According to the two parity check equations, we split the coefficient matrix A into H_L and H_S, which corresponds to the lost and surviving data nodes, respectively. Because H_L is invertible, we get the following equation,

$$H_L * D_L = H_S * D_S \qquad (4)$$

We can change Equation 4 to Equation 5 as below via matrix transformation (multiply H_L^{-1} on the both sides of Equation 4),

$$D_L = H_L^{-1} * H_S * D_S \qquad (5)$$

Therefore, we can reconstruct the lost parity nodes via Equation 5.
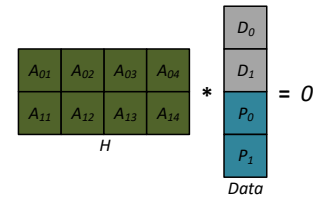

Fig. 10. The brief construction of MSR code with ($n = 4, k = 2, s = m = 2, l = 4$) following Equation 3. G is a *coefficient matrix* based on the finite field F. Each element $A_{i,j}$ is a $2 * 2$ *matrix*. The dot product of the data vector the corresponding row in G is equal to zero.

D. Construction of Global Parities

We use RS code [31] to generate the global parities,

Construction via Global Parities *Let H be the parity check matrix, data node is $(D_0, D_1, ..., D_{k-1})$ and parity node is $(G_0, G_1, ..., G_{r-1})$. Then we define the code equation,*

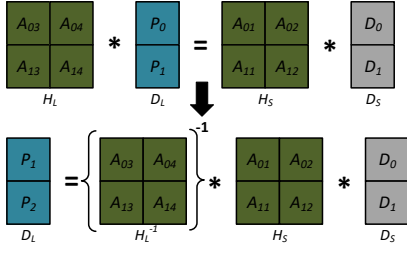$$H * (D_0, ..., D_{k-1}, G_0, ..., G_{r-1})^T = 0$$

Fig. 11. The encoding process of MSR code ($n = 4, k = 2, s = m = 2, l = 4$). D_L represents the lost data vector and D_S represents the surviving data vector. H_L and H_S are the coefficient matrix of D_L and D_S respectively.

With each $r * r$ sub-matrix H' of H is invertible.

E. Encoding with AZ-Code

The encoding process of AZ-Code is simple. We summarize it into the following steps,

1) Use the data nodes in the same AZ to calculate the local parity with MSR code and store them in corresponding AZ.
2) Use the whole data nodes in all AZs to calculate the global parity with RS code.
3) Store global parity evenly across multiple AZs.

F. Decoding with AZ-Code

1) Decoding with Local Parities: MSR code using different decoding methods for various numbers of node failures.

When single node fails, MSR code contacts with the other $d = n - 1$ help nodes to recover the lost data together. Recovery method is described as below,

Assume that data C_q of the code Θ fail. And C_q can be regarded as $C_{(v_q-1)s+u_q+1}$.

Step1: We construct a new coefficient matrix H' based on the original coefficient matrix H by taking a part of elements from each submatrix $A_{t,i}(0 \leq t \leq r-1, 1 \leq i \leq n)$. The new submatrix $A'_{t,i}$ is constructed by the following function,

$$A'_{t,i} = \begin{cases} A_{t,i}(a,b)|a,b \in \{a_{v_k} = u_k\} & if\ i \neq q \\ A_{t,i}(a,b)|a \in \{a_{v_k} = u_k\}, b \in \{0,1,...,l\} & if\ i = q \end{cases}$$

Step2: The data obtained from other $n-1$ data nodes (C_i) to recover the lost data is defined as C'_i. The data set C'_i has the following format,

$$C'_i = \{c_{i,a}|a_{v_k} = u_k, a = 0, 1, ..., l-1\}$$
$$(i = 0, 1, ..., n-1\ and\ i \neq q.)$$

For the convenience of representation we define that $C'_q = C_q$.

Step3: We get the new decoding equation,

$$\begin{bmatrix} A_{0,1} & \cdots & A_{0,n} \\ \vdots & \ddots & \vdots \\ A_{r-1,1} & \cdots & A_{r-1,n} \end{bmatrix} * \left[C'_0, C'_1, ..., C'_{n-1}\right]^T = 0 \quad (6)$$

And the lost data C_q can be recovered via Equation 6.

We show the detailed recovery process under single node failure case in Fig.12 and Fig.13. And assume that data node D_0 fails. According to the recovery method for single node failure, we choose the $1, 3, 5, 7$ rows from the original coefficient matrix A to construct a new matrix A'. The data nodes D_1, P_0, P_1 only need to provide their first and third sub-nodes in a sub-packetization. Then we get the decoding equation shown in Fig.13. At last we use the PCM method, which is the same as the encoding process, to recover the lost data node D_0.
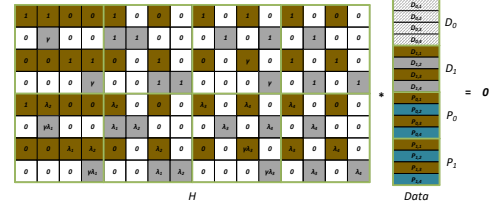

Fig. 12. The decoding process of MSR code ($n = 4, k = 2, s = m = 2, l = 4$). H is the new coefficient matrix. And D_0 is the lost data node. The rows $1, 3, 5, 7$ of coefficient matrix A and the sub-nodes $1, 3$ of sub-packetization in the data node D_0, P_1, P_2 are selected to recover D_0, which are marked as brown.
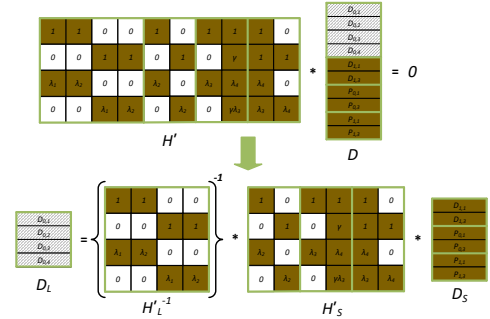

Fig. 13. The decoding process of MSR code with($n = 4, k = 2, s = m = 2, l = 4$). H' is the new coefficient matrix constructed by original matrix H. D_L represents the lost data vector and D_S represents the surviving data vector. H'_L and H'_S are the coefficient matrices of D_L and D_S respectively.

When multiple nodes fail, The decoding process of the MSR code is the same as the encoding process and shown in Fig.14. Assume that D_0 and D_1 are lost data, and P_0, P_1 are surviving data nodes. Then we can recover the D_0 and D_1 by using Equation 5.

2) Decoding with Global Parities: The global parities recovery method is shown as below,

Assume that any r nodes fail, we rewrite the data set D and parity set G as surviving data set $D_S = (D_{S0}, D_{S1}, ..., D_{Sk-1})$ and lost data set $D_L = (D_{L0}, D_{L1}, ..., D_{Lr-1})$. then we choose the column corresponding to the surviving data from H and get H_S, the remaining column make up H_L. We use the following equation to recover the lost data.

$$D_L * H_L = D_S * H_S$$
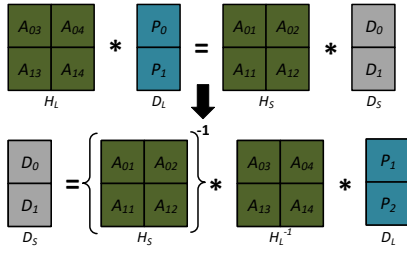$$D_L = (H_L)^{-1} * H_S * D_S.$$

Fig. 14. The decoding process of MSR code with($n = 4, k = 2, s = m = 2, l = 4$). D_L represents the lost data vector and D_S represents the surviving data vector. H_L and H_S are the coefficient matrices of D_L and D_S respectively.

3) Hybrid Decoding with Global and Local Parities: we use the hybrid recovery method to decode the lost data in the condition that local or global parities can't recover the lost data independently.

Given an AZ-Code $AZ(k, z, p, g)$, we define that $m = k/z+p$ (represents the total number of data nodes in one AZ) and a data set $D = (C_0, C_1, ..., C_{k-1})$ which generate local and glocal parities. $(G_0, G_1, ..., G_{g-1})$ represents the global parity nodes with coefficient matrix $R_{g \times n}$ and $(P_0, P_1, ..., P_{p-1})$ represents the local parity nodes with coefficient matrix $A_{p \times m}$. In order to unify the matrix size, we make the following identity transformation,

$$R_{i,j} = diag(r_{i,j}, r_{i,j}, ..., r_{i,j})_{l \times l}$$

Step1: We choose the local parities belong to the first AZ as an example (Generally we select an AZ which contains the most lost data nodes) and get $g + p$ parity check equations,

$$\{\sum_{i=0}^{k-1} R_{t,i} * C_i + \sum_{i=k}^{k+g-1} R_{t,i} * G_{i-k} = 0 | t = 0, 1, ..., g - 1\},$$

$$\{\sum_{i=0}^{k/z-1} A_{t,i} * C_i + \sum_{i=k/z}^{m-1} A_{t,i} * P_{i-k/z} = 0 | t = 0, 1, ..., p - 1\}$$

Step2: We use a new coefficient matrix $J_{(g+p) \times (k+g+p)}$ to represent these $g + p$ equations,

$$J_{i,j} = \begin{cases} R_{i,j} & if \ 0 \le i \le g - 1 \ and \ 0 \le j \le k + g - 1 \\ A_{i,j} & if \ g \le i \le g + p - 1 \ and \ 0 \le j \le k/z - 1 \\ A_{i,j-k-g} & if \ g \le i \le g + p - 1 \ and \\ & k + g \le j \le k + g + p - 1 \\ 0 & otherwise \end{cases}$$

Step3: Then we get the equation with the PCM format,

$$J * \left[C'_0, ..., C'_{k-1}, G_0, ..., G_{g-1}, P_0, ..., P_{p-1} \right]^T = 0 \quad (7)$$

If any $g + p$ nodes belongs to $AZ(k, z, p, g)$ fails, we can recover them by Equation 7. The parameters should be properly chosen to guarantee that the corresponding sub-matrices are invertible.

Algorithm 1 shows the decoding strategy of AZ-Code. Firstly, we scan all AZs and use the local parities to decode

the lost data. Then the remaining lost data, which cannot be recovered by the local parities only, are reconstructed through global parities. Finally, we use a hybrid recovery method to decode the lost data.

Algorithm 1 Reconstruction Algorithm of AZ-Code

Require:
  Configuration of AZ-Code $AZ - Code(k, z, p, g)$;
  A lost data position vector $F[k]$;
  A lost data number k;
  A global parity set G;
  A local parity set L;
  A surviving data set D_S;
  $Local\_Parity\_Decoding(D)$: using data set D to recover lost data by local parities;
  $Global\_Parity\_Decoding(D)$: using data set D to recover lost data by global parities;
  $Combination\_Decoding(D)$: using data set D to recover lost data through the combination decoding method via local and global parities;
Ensure: original lost data set D_L
  Initial $D_L = \emptyset$
  for all $F[i]$ belongs to the same AZ_j and $F[i] \notin G$ **do**
    if $size(F[i] \in AZ_j) \le p$ **then**
      $D_{temp} = D_{Si} \in AZ_j \bigcup L_i \in AZ_j$;
      $D_L += Local\_Parity\_Decoding(D_{temp})$;
      $k = k - size(L[i] \in AZ_j)$.
    end if
  end for
  if $p < k \le g$ **then**
    $D_{temp} = (D_{Si} \in \bigcup_j AZ_j) \bigcup G$;
    $D_L += Global\_Parity\_Decoding(D_{temp})$.
  else
    if $g < k \le g + p$ **then**
      $D_{temp} = (D_{Si} \in \bigcup_j AZ_j) \bigcup G$;
      $D_L += Combination\_Decoding(D_{temp})$.
    else $\{k > g + l\}$
      ERROR: Exceed the capability of fault tolerance.
    end if
  end if

G. Proof of Correctness

Actually, $AZ(k, z, p, g)$ tolerates up to $g+p$ arbitrary nodes failures and AZ-Code can recover up to $g+p*z$ nodes in some special cases.

We assume that $g + p$ arbitrary nodes fail. The locations of failure nodes can be divided into two cases.

1) The total number of lost data/parity nodes are less than or equal to g. This case is shown in Fig.16(a).
2) The total number of lost data/parity nodes are more than g. This case is shown in Fig.16(b).

In case 1, we can recover all data nodes by the global parities and then we can reconstruct the local parity nodes.

In case 2, we use the combination decoding method to recover the lost data because local or global parities cannot recover the lost data independently. Note that we should choose the AZ which has the most number of failure nodes when we use the combination decoding method. We take $AZ(6, 3, 2, 3)$ as an simple example, which is shown in Fig.16.

According to Fig.16(a), we use G_0, G_1, G_2 to recover D_0, D_1, D_2, and use D_0, D_1 to reconstruct the P_0, P_1.

According to Fig.16(b), we use the the hybrid decoding method to recover the lost data and the recovery process is shown in Fig.17 and Fig.18. Columns $1, 2, 7, 10, 11$ are utilized to establish the matrix H_L and the remaining columns to build up H_S. Then we can use the equation $D_L = H_L^{-1} * H_S * D_S$

TABLE III
COMPARISONS AMONG AZ-CODE, RS CODE, LRC CODES IN TERMS OF
STORAGE EFFICIENCY, CAPABILITY OF FAULT TOLERANCE AND
RECONSTRUCTION COST

EC	Storage Efficiency	Fault Tolerance	Reconstruction Cost (single data node failure)
$AZ(k,z,p,g)$	$\frac{k}{k+z*p+g}$	$g+p$	$\frac{k/z+p-1}{p}$
$RS(k,r)$	$\frac{k}{k+r}$	r	k
$LRC(k,z,g)$	$\frac{k}{k+z+g}$	$g+1$	k/z

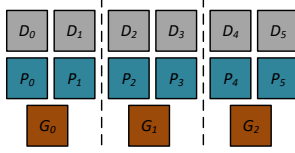which is introduced in Section III-F to recover the lost data nodes.

Fig. 15. The construction of AZ-Code ($k = 6, z = 3, p = 2, g = 3$). Data nodes are marked gray, local parities are marked blue and global parities are marked brown. Local parities are stored in their corresponding AZs and global parities are evenly distributed among the three AZs.

H. Properties of AZ-Code

We analyze the properties of $AZ(k,z,p,g)$ from the following aspects and show the formulation in table.III.

- *High Reliability*: Fault tolerance is the quantification of the reliability abilities. The fault tolerance capabilities of AZ-Code is $g + p$, so we can increase the number of global and local parities to provide high reliability in AZ environment.
- *Low Storage Efficiency*: Storage efficiency of AZ-Code is $k/(k + z * p + g)$, so we can increase k (or reduce $g/p/z$) to get better storage efficiency. For MDS code the optimal storage efficiency is $k/(k + p + g)$, so AZ-Code is near optimal.
- *Low Recovery Cost via Local Parities*: Recovery cost include the computation cost, network bandwidth, total number of I/Os. the reconstruction cost is equivalent to network bandwidth here and also affects total number of I/Os. Usually the reconstruction cost should be as low as possible. According to the equation in Table III, the reconstruction cost of AZ-Code is lower than RS Code
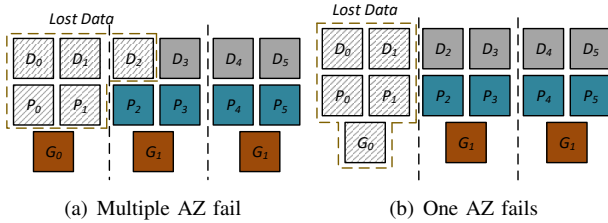

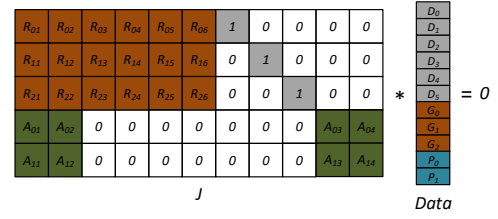Fig. 16. Two cases of total 5 nodes failure in AZ-Code ($k = 6, z = 3, p = 2, g = 3$).

Fig. 17. The combined recovery process of AZ-Code ($k = 6, z = 3, p = 2, g = 3$) and RS code ($n = 9, k = 6$).
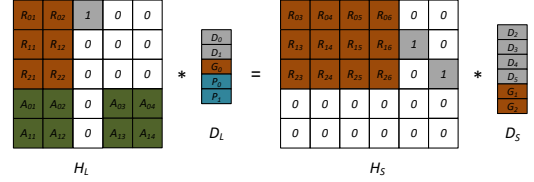

Fig. 18. The combined recovery process of AZ-Code ($k = 6, z = 3, p = 2, g = 3$) and RS code ($n = 9, k = 6$).

and LRC with the same code size in single node failure condition. As the number of failure nodes increases, the gap between AZ-Code, RS code and LRC decrease and eventually eliminated.

- *High Scalability*: AZ-Code has four dimensions k, z, p, g to adapt to the various configuration of data centers.

The properties of AZ-Code with $z = 2$ and $z = 3$ are summarized in Table IV. AZ-Code generally performs well on three attributes. We can change the code configuration to get excellent performance in one properties. For example, if we choose $AZ(6,3,2,3)$, we can get low reconstruction cost.

IV. EVALUATION

In this section, we conduct a series of experiments to demonstrate the efficiency of AZ-Code under AZ environment.

TABLE IV
SUMMARY ON VARIOUS CONFIGURATIONS OF AZ-CODE WITH TWO AZS
AND THREE AZS

Configuration	Storage Efficiency	Fault Tolerance	Reconstruction Cost (single data node failure)
$AZ(4,2,2,2)$	0.4	4	1.5
$AZ(8,2,2,2)$	0.57	4	2.5
$AZ(8,2,2,4)$	0.50	6	2.5
$AZ(12,2,2,2)$	0.67	4	3.5
$AZ(12,2,2,4)$	0.60	6	3.5
$AZ(12,2,4,6)$	0.55	8	3.5
$AZ(6,3,2,3)$	0.4	5	1.5
$AZ(12,3,2,6)$	0.5	8	2.5
$AZ(12,3,2,9)$	0.44	11	2.5
$AZ(18,3,2,6)$	0.60	8	3.5
$AZ(18,3,2,9)$	0.55	11	3.5
$AZ(18,3,2,12)$	0.50	17	3.5

8

A. Evaluation Methodology

We select RS code and LRC codes (which are widely used in AZ environment), as well as several XOR-based codes to compare with AZ-Code in our evaluations. Since 3AZ is a general configuration in cloud storage systems, we set $z = 3$ in our evaluation. We also use both mathematical analysis and experiments to demonstrate the efficiency of AZ-Code.
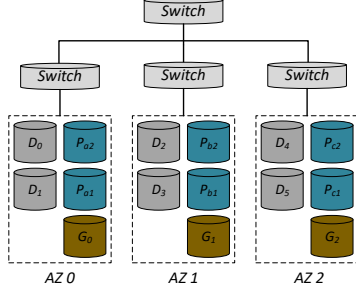

Fig. 19. The environment of experiments.

TABLE V
DETAILS OF THE EVALUATION PLATFORM

Description	DELL R730 Server
CPU	Intel Xeon 3.0GHz
NIC	1Gbps
Memory	32GB
Disk	8TB HDD
OS	Linux 3.19
Platform	Hadoop HDFS 3.0.3

1) Metrics and Methods for Mathematical Analysis: We use **Storage Efficiency**, **Fault Tolerance** and **Reconstruction Cost** which defined in Section II-A as the metrics. The number of nodes in a cluster limits the total code size of AZ-Code. Therefore, we make the following constraints for $AZ(k, z, p, g)$.

$$k \leq 18, \ z = 2 \ or \ 3, \ p = 2 \ or \ 3, \ g \leq 12$$

The mathematical analysis is divided into two aspects. Firstly, we analyze the influence of four parameters on the three metrics in Section II-A. In this part, we only discuss the reconstruction cost under the single node failure condition, and find out the differences of AZ-Code under various values of parameters. Secondly, we compare the reconstruction cost of AZ-Code with LRC/RS codes under different faulty conditions. We also compare the reconstruction cost of AZ-Code with existing coding schemes (RS, LRC, MSR, XOR-based codes) under three situations, such as single data node failure, double and multiple data nodes failures.

2) Metrics and Methods for Experiments: We use **recovery time**, which consists of computation time, I/O overhead and transmission time, as the metric in our experiments. Several code configurations with the same n and k are chosen to explore the impact of other parameters.

The environment of our experiments is shown in Fig.19 and Table V. We set up a Hadoop system [2] with one NameNode and three DataNodes (to simulate three availability zones) to evaluate the performance of the erasure codes. Four DELL R730 servers based on Linux ext4 filesystem are deployed as these nodes and connected with a gigabit Ethernet. Each DataNode stores multiple chunks (to simulate the nodes in AZs), and the number of chunks is adapted to the EC configuration and each chunk contains exactly 1GB data. Each chunk consists of many blocks which is the basic unit of encoding and decoding processes in our experiments, and we change the block size to simulate various possible failure conditions in the real AZ environment.

We implement AZ-Code in Intel ISA-L of HDFS and write a test program to obtain the recovery time, computation time, I/O overhead and transmission time. In the test program we randomly generate some error chunks and retrieve data from three DataNodes to recover the error chunks.

Our experiments include four parts. In the first three parts, we evaluate the full recovery time of three ECs with different configurations under single, double and multiple failure node(s) situations. Besides, we evaluate the proportion of three components in recovery time, which is mentioned above, under single and multiple failure node(s) situations.

B. Numerical Results of Mathematical Analysis

In this section, we show the mathematical analysis of AZ-Code and compare it with LRC and RS codes.

1) AZ-Code Analysis: In Fig.20 we summarize the performance of $AZ(k, z, p, g)$ in terms of various configurations in storage efficiency, fault tolerance, and reconstruction cost in single data node failure situation.

- **Storage Efficiency**: Storage efficiency ($\frac{k}{k+z*p+g}$) is determined by four parameters k, z, p and g. So we vary k from 4 to 18 and g from 2 to 12. Since z and p both have only two choices, we can get totally four conditions. The results are shown in Fig.20. We can see that storage efficiency is direct proportion to k and inversely proportional to z, p and g.
- **Reconstruction Cost**: Reconstruction cost is determined by k, z and p and defined as $(k/z + p - 1)/p$. We vary k from 4 to 18, p and z from 2 to 3 and hold g to constant 12. The results are shown in Fig.20(g) and Fig.20(h).
- **Fault Tolerance**: Fault tolerance, which is defined as $g + p$, is determined by g and p. So we vary g from 2 to 12 and p from 2 to 3 with two configuration: $z = 2, k = 12$ and $z = 3, k = 18$. The results are shown in Fig.20(e) and Fig.20(f).

Next we consider how to select the parameters of AZ-Code for 2AZ and 3AZ environment. We choose storage efficiency and reconstruction cost as the metrics. We use the storage efficiency of 0.25 and fault tolerance of 3 as the threshold which is the value of three replication strategy. We keep those code sets of parameters that yield equal or higher storage efficiency and fault tolerance threshold. Then we plot the storage efficiency and the reconstruction cost of the remaining code sets in Fig.21(b) (3AZ-Code) and Fig.21(a) (2AZ-Code). Each individual point represents one set of coding parameters.
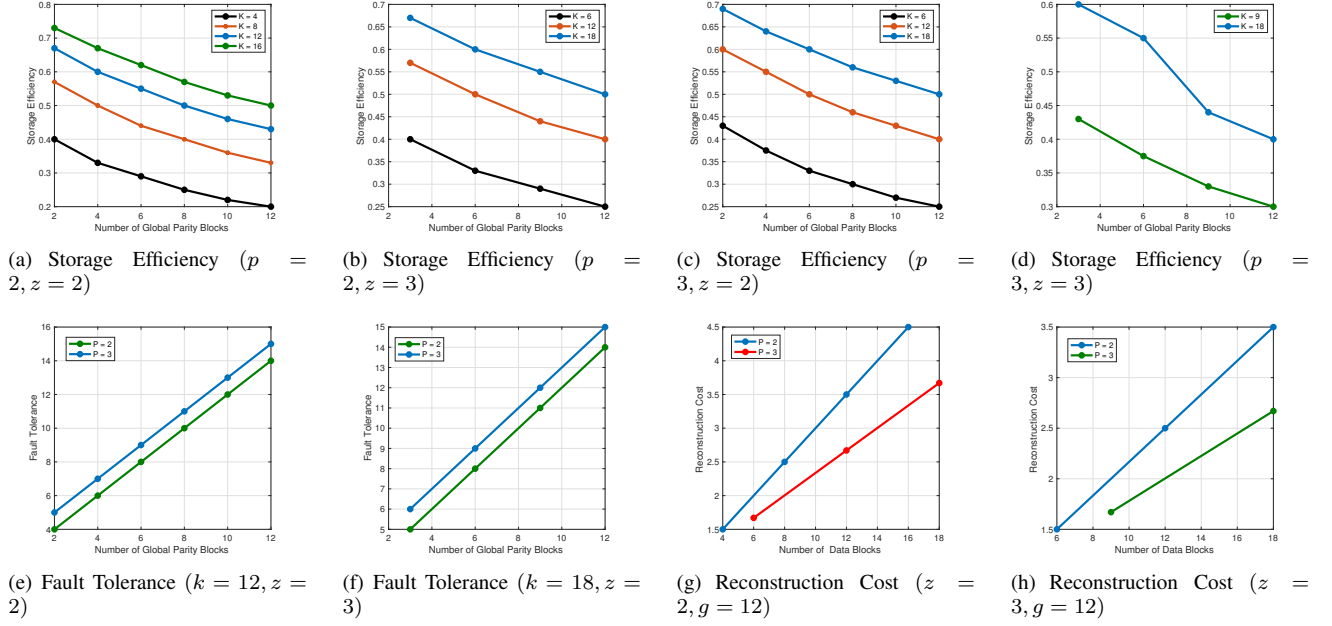
Fig. 20. The impact of different parameters on storage efficiency, fault tolerance and reconstruction cost.

(a) Storage Efficiency ($p = 2, z = 2$)

(b) Storage Efficiency ($p = 2, z = 3$)

(c) Storage Efficiency ($p = 3, z = 2$)

(d) Storage Efficiency ($p = 3, z = 3$)

(e) Fault Tolerance ($k = 12, z = 2$)

(f) Fault Tolerance ($k = 18, z = 3$)

(g) Reconstruction Cost ($z = 2, g = 12$)

(h) Reconstruction Cost ($z = 3, g = 12$)

TABLE VI
IMPROVEMENT OF AZ-CODE OVER RS/LRC CODES UNDER DIFFERENT FAILURE SCENARIOS

Failure Scenarios	Erasure Codes	Block Size					
		1MB	2MB	4MB	8MB	16MB	32MB
Single Node Failure	$LRC(6,3,6)$	23.94%	23.92%	23.35%	22.08%	24.20%	23.47%
	$RS(6,9)$	74.73%	74.43%	74.68%	74.56%	74.57%	74.27%
	$LRC(12,3,6)$	34.81%	34.70%	34.72%	34.60%	34.66%	34.56%
	$RS(12,9)$	78.17%	78.24%	78.16%	78.19%	78.05%	78.14%
	$LRC(18,3,6)$	34.38%	33.53%	33.09%	31.11%	30.30%	34.25%
	$RS(18,9)$	78.13%	77.87%	77.78%	77.02%	76.89%	78.16%
Double Node Failures	$LRC(6,3,6)$	35.50%	35.43%	35.14%	34.28%	35.58%	34.83%
	$RS(6,9)$	52.92%	52.73%	53.02%	52.86%	52.88%	52.07%
	$LRC(12,3,6)$	45.34%	45.28%	45.23%	45.10%	45.00%	45.12%
	$RS(12,9)$	58.72%	58.77%	58.60%	58.58%	58.33%	58.54%
	$LRC(18,3,6)$	42.80%	42.31%	42.19%	41.12%	41.56%	43.82%
	$RS(18,9)$	57.08%	56.79%	56.74%	55.93%	56.39%	57.94%
Multiple Node Failures	$LRC(6,3,6)$	5.61%	5.17%	4.76%	5.03%	4.87%	3.94%
	$RS(6,9)$	5.94%	5.19%	4.90%	5.04%	4.97%	3.89%
	$LRC(12,3,6)$	4.28%	4.23%	4.00%	3.77%	3.38%	3.87%
	$RS(12,9)$	4.28%	4.31%	4.08%	3.81%	3.42%	3.84%
	$LRC(18,3,6)$	8.85%	8.36%	8.65%	7.79%	9.77%	10.68%
	$RS(18,9)$	8.83%	8.36%	8.68%	7.78%	9.71%	10.67%

Each parameter set represents a certain trade-off between the storage efficiency and the reconstruction cost.

Different parameters can result in different storage efficiency. However, we only pay attention on the one with the lower reconstruction cost. Therefore, we outline the lower bound of the trade-off points. The lower bound curve characterizes the fundamental trade-off between storage efficiency and reconstruction cost for AZ-Code.

2) Comparisons among Different Erasure Codes: We select the following code sets in our analysis. In these evaluations, we set k as $(6, 12, 18)$ respectively and $z = 3$ (3AZ environment).

- RS-based codes:
  - **AZ-Code**: $AZ(k,3,2,3)$ with the best storage efficiency and $AZ(k,3,2,6)$ with high fault tolerance.

- **RS code** [26]: $RS(k,9)$ with the same total nodes n of $AZ(k,3,2,3)$ and $RS(k,8)$ with the same fault tolerance of $AZ(k,3,2,6)$.
- **LRC** [17] [25]: $LRC(k,3,6)$ with the same total nodes n of $AZ(k,3,2,3)$ and $LRC(k,3,7)$ with the same fault tolerance of $AZ(k,3,2,6)$.
- **MSR** [11] [15] [36] [42] [24] [13]: $MSR(k,6)$ with $r = 6$.

- XOR-based codes:

  - **EVENODD** [4]: A kind of XOR-base code which can tolerate concurrent failures of any double nodes, in which the configuration is $k + 2$.
  - **STAR** [16]: A kind of XOR-base code which can
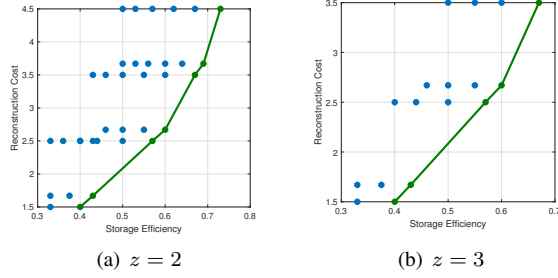
(a) $z = 2$ (b) $z = 3$

Fig. 21. The impact of different parameters on storage efficiency, fault tolerance and reconstruction cost.

tolerate concurrent failures of any triple nodes, in which the configuration is $k + 3$.

First, we choose the AZ-Code sets which are located in the lower bound curve in Fig.21(b) (3AZ-Code) to compare RS and LRC. In this situation, we define the same parameters k and n of these three codes. For example, $AZ(12, 3, 2, 3)$, $RS(12, 9)$ and $LRC(12, 3, 6)$ are involved in our Comparisons, where k and n are set to the same values ($k = 12$, $n = 21$). Then we choose another code sets with the same capability of fault tolerance and the same value of k. For example, $LRC(12, 3, 7)$, $AZ(12, 3, 2, 6)$ and $RS(12, 8)$ have the same $k = 12$ as well as the same fault tolerance 8. We use the reconstruction cost to evaluate the recovery performance. The results are presented in Fig.22, which illustrate that AZ-Code have the best performance in all conditions. For example, $AZ(6, 3, 2, 3)$ saves 75.00% reconstruction cost compared to $RS(6, 6)$.

Typical erasure codes (including AZ-Code) under faulty conditions are illustrated in Fig.23. We can see that AZ-Code has the least reconstruction cost under all three situations compared with other EC.

C. Experimental Results of Single Node Failure

In this section, we compare the recovery time of RS code, LRC codes and AZ-Code with the same parameter k under single data node failure condition. Three different code sets are chosen to evaluate the performance. In all three experiments, AZ-Code performs better than RS code and LRC codes under any block size from 1MB to 32MB.

- $AZ(6, 3, 2, 3)$, $RS(6, 9)$ and $LRC(6, 3, 6)$: Experiment results are shown in Fig.24(a). We can find out that $AZ(6, 3, 2, 3)$ is up to 74.73% faster than $RS(6, 9)$ and 24.20% faster than $LRC(6, 3, 6)$.
- $AZ(12, 3, 2, 3)$, $RS(12, 9)$ and $LRC(12, 3, 6)$: Experiment results are shown in Fig.24(b). The figure illustrates that $AZ(12, 3, 2, 3)$ is up to 78.24% faster than $RS(12, 9)$ and up to 34.81% faster than $LRC(12, 3, 6)$.
- $AZ(18, 3, 2, 3)$ and $RS(18, 9)$ and $LRC(18, 3, 6)$: Experiment results are shown in Fig.24(c). We can see that $AZ(18, 3, 2, 3)$ is up to 78.12% faster than $RS(18, 9)$ and up to 34.38% faster than $LRC(18, 3, 9)$.

The higher reliability of RS code requires a larger amount of data to be transmitted, and this makes it perform much worse

than AZ-Code. Though the computational complexity of AZ-Code grows when k increases, the advantage of transmission overhead for AZ-Code becomes more prominent. Therefore, AZ-Code can still achieve the best performance.

In the above three experiments, the performance of AZ-Code is almost the same when the block size changes. The optimization ratio reaches the peak when the number of data nodes (k) is 12.

D. Experimental Results of Double Node Failures

In these three experiments, we compare the recovery time of AZ-Code with RS code and LRC codes in double nodes failure condition, respectively.

- $AZ(6, 3, 2, 3)$, $RS(6, 9)$ and $LRC(6, 3, 6)$: Experimental results are shown in Fig.25(a). We find that $AZ(6, 3, 2, 3)$ is up to 53.02%, 35.58% faster than $RS(6, 9)$ and $LRC(6, 3, 6)$, respectively. Various block size slightly affect the performance of three codes.
- $AZ(12, 3, 2, 3)$, $RS(12, 9)$ and $LRC(12, 3, 6)$: Experimental results are shown in Fig.25(b). It illustrates that $AZ(12, 3, 2, 3)$ is up to 58.76% faster than $RS(12, 9)$ and up to 45.34% faster than $LRC(12, 3, 6)$. The optimization ratio between AZ-Code and LRC codes increases sharply when k grows. It is because LRC saves the transmission overhead under faulty conditions.
- $AZ(18, 3, 2, 3)$ and $RS(18, 9)$ and $LRC(18, 3, 6)$: Experiment results are shown in Fig.25(c). We can see that $AZ(18, 3, 2, 3)$ is up to 57.08% faster than $RS(18, 9)$ and up to 42.80% faster than $LRC(18, 3, 9)$.

From these experiments, we observe that the performance of AZ-Code achieves the peak when $k = 12$. Compared with the single node failure situation, the optimization ratio between AZ-Code and RS code decreases under double node failures situation. And the average optimization ratio between AZ-Code and LRC codes increases. It is because LRC codes have one and only one local parity in each group, which can speed up the decoding process when single node failure occurs, but cannot speed up the process when two or more node failures occur.

E. Experimental Results of Multiple Node Failures

In these three experiments, we test the recovery time of AZ-Code, RS code and LRC codes when five failure nodes need to be recovered.

- $AZ(6, 3, 2, 3)$, $RS(6, 9)$ and $LRC(6, 3, 6)$: The related experimental results are shown in Fig.26(a). We can find out that $AZ(6, 3, 2, 3)$ is up to 5.94% faster than $RS(6, 9)$ and 5.61% faster than $LRC(6, 3, 6)$. The figure illustrates that with the increase of block size, the optimization ratio decrease slowly and the maximum value is produced when the block size is 1MB.
- $AZ(12, 3, 2, 3)$, $RS(12, 9)$ and $LRC(12, 3, 6)$: Experiment results are shown in Fig.26(b). It illustrates that $AZ(12, 3, 2, 3)$ is up to 4.28% faster than $RS(12, 9)$ and up to 4.28% faster than $LRC(12, 3, 6)$.

(a) The same total nodes ($n = 15, k = 6$)

(b) The same total nodes ($n = 21, k = 12$)

(c) The same total nodes ($n = 27, k = 18$)

(d) The same fault tolerance (8) $k = 6$

(e) The Same Fault Tolerance(8) $k = 12$
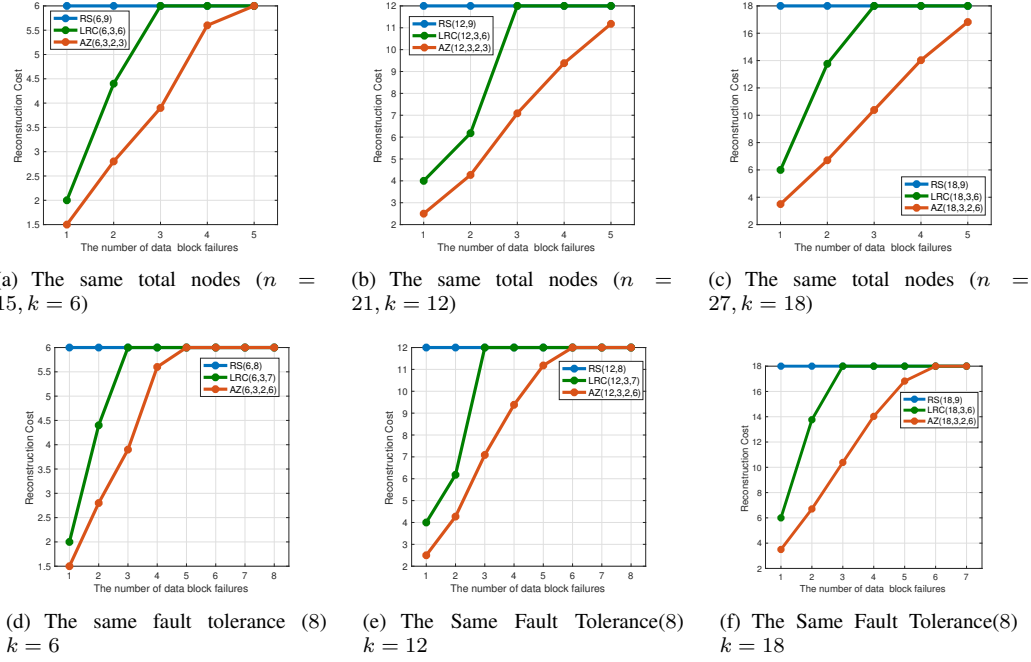
(f) The Same Fault Tolerance(8) $k = 18$

Fig. 22. Comparisons of reconstruction cost among AZ-Code, LRC and RS code with the same number of nodes (n) and levels of fault tolerance(8)
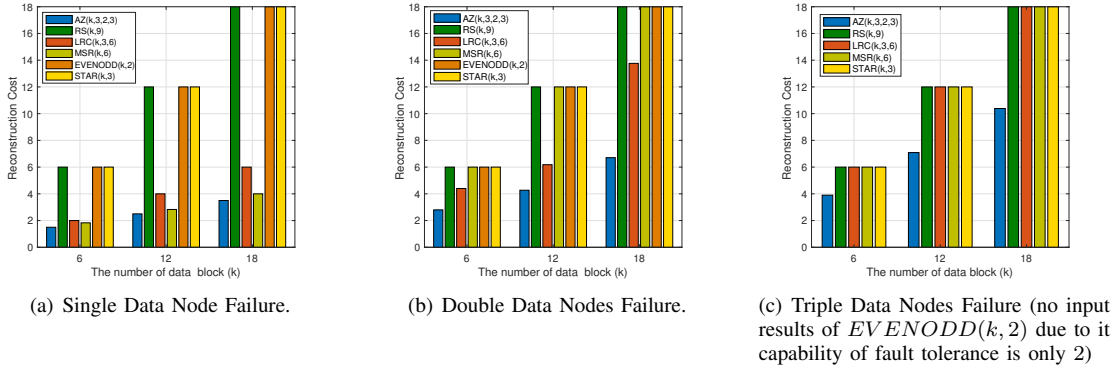

(a) Single Data Node Failure.

(b) Double Data Nodes Failure.

(c) Triple Data Nodes Failure (no input results of $EVENODD(k, 2)$ due to it capability of fault tolerance is only 2)

Fig. 23. Comparisons of reconstruction cost among AZ-Code, RS code, LRC and XOR-based code in single data node failure, double data nodes failure and triple data nodes failure scenarios.

- $AZ(18, 3, 2, 3)$, $RS(18, 9)$ and $LRC(18, 3, 6)$: Experiments results are shown in Fig.26(c). We can see that $AZ(18, 3, 2, 3)$ is up to 9.71% faster than $RS(18, 9)$ and up to 9.77% faster than $LRC(18, 3, 9)$.

In these three experiments, we can see that the performance of the three codes tends to be the same when the number of node failures grows. It also shows that our AZ-Code always achieves a better performance under any number of node failures situations. And the optimization ratio achieves a peak when $k = 18$.

F. Experimental Results of Recovery Time

In these two experiments, we separately measure the three components of recovery time, which are transmission time, I/O overhead and computation time. Fig.26(b) and Fig.27(b) illustrate the distribution of three recovery time components.

We can find out that the transmission time is about $8.71\times$ of computation time and $5.45\times$ of I/O overhead for AZ-Code. Therefore, transmission time plays a more significant impact on recovery time, that's why AZ-Code focus on integrating MSR code to reduce the recovery network bandwidth.

G. Analysis

The improvements of AZ-Code over RS/LRC codes are listed in Table VI. From the table, we discover that AZ-Code can obtain a better decoding performance in all experiments. There are several reasons to achieve these gains. First, AZ-Code has less network transmission overhead because of the usage of MSR code. Secondly we use local parities to recover the lost data as far as possible, that's why AZ-Code obtains a much better recovery performance. Thirdly the calculation complexity of AZ-Code can be well controlled by adjusting
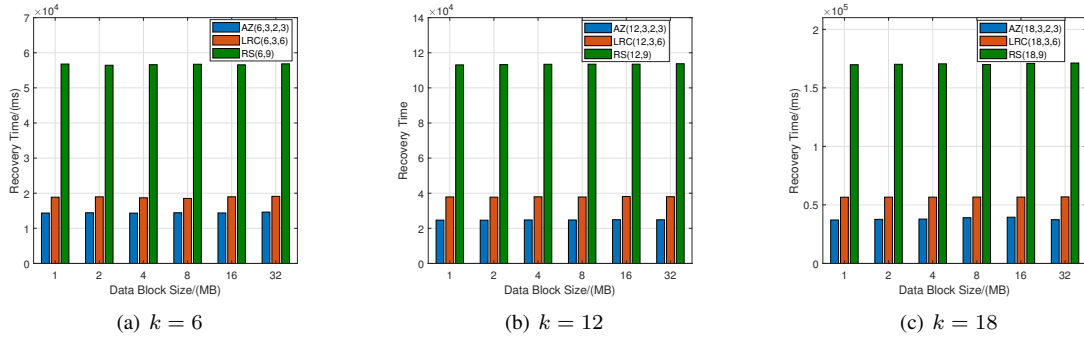
(a) $k = 6$ (b) $k = 12$ (c) $k = 18$

Fig. 24. Comparisons of recovery time among AZ-Code, RS code and LRC codes under single node failure.

(a) $k = 6$ (b) $k = 12$ (c) $k = 18$
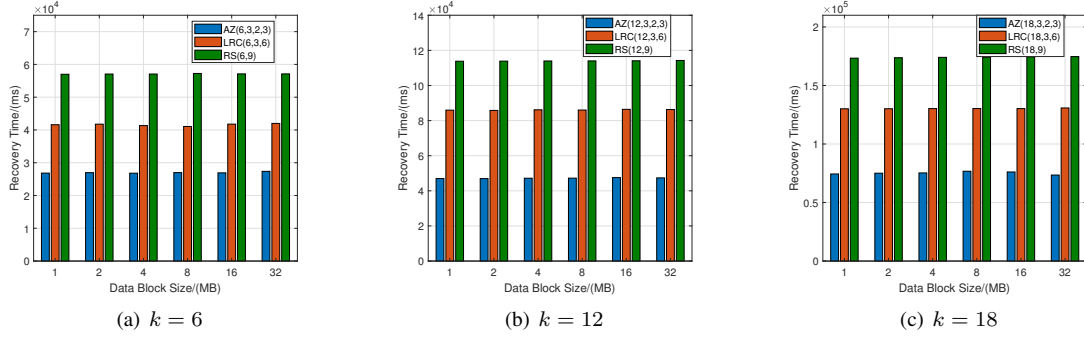
Fig. 25. Comparisons of recovery time among AZ-Code, RS code and LRC codes under single node failure.

(a) $k = 6$ (b) $k = 12$ (c) $k = 18$
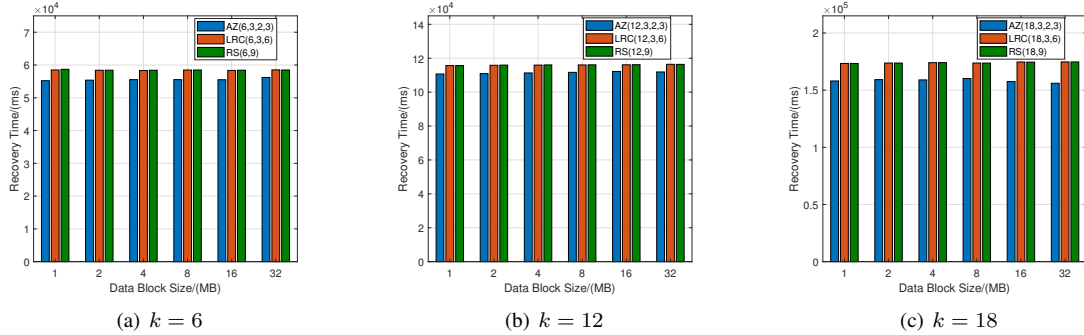
Fig. 26. Comparisons of recovery time among AZ-Code, RS code and LRC codes under single node failure.

the parameters to construct short parity chain. Finally, AZ-Code use RS code to provide high flexibility, so it is well adapted to the multiple AZs environment.

V. CONCLUSION

In this paper, we propose Availability Zone Level Erasure Code (AZ-Code) to improve the recovery performance in AZ environment. Based on MSR code and LRC, we introduce the construction of AZ-Code and design the encoding and decoding algorithm. Compared with LRC, AZ-Code use MSR code to construct the local parities, which results in faster recovery speed and higher scalability. To demonstrate the effectiveness of AZ-Code, we conduct experiments in different conditions. Comparing to LRC and RS codes with similar reliability and storage efficiency, AZ-Code has the following advantages: 1) reducing the recovery time up to 78.24% compared with RS code and 34.81% compared with LRC when a single node fails; 2) reducing the recovery time up to 58.77% compared with RS code and 45.34% compared with LRC codes when double nodes fail; 3) better scalability than RS codes.
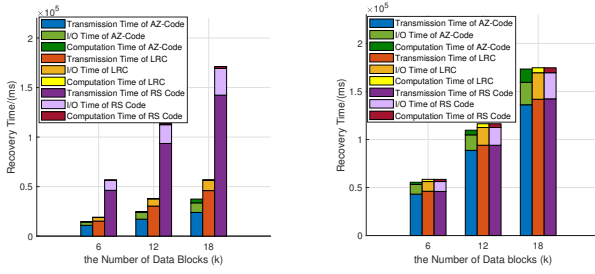
(a) Detailed recovery time of full system recovery in terms of Single node failure

(b) Detailed recovery time of full system recovery in terms of Multiple nodes failure

Fig. 27. Detailed recovery time of full system recovery, which includes the CPU computation time, network transmission time and I/O processing time in storage devices.

REFERENCES

[1] "Facebooks erasure coded hadoop distributed file system (hdfs-raid)," https://github.com/facebook/hadoop-20.

[2] "Hadoop distributed file system (hdfs)," http://hadoop.apache.org.

[3] I. Bermudez, S. Traverso, M. Mellia, and M. Munafo, "Exploring the cloud from passive measurements: The amazon aws case," in *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 230–234.

[4] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 192–202, 1995.

[5] J. Bloemer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme," International Computer Science Institute, Tech. Rep. TR-95-048, August 1995.

[6] Y. Cassuto and J. Bruck, "Cyclic lowest density MDS array codes," *IEEE Transactions on Information Theory*, vol. 55, no. 4, pp. 1721–1729, 2009.

[7] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[8] M. Copeland, J. Soh, A. Puca, M. Manning, and D. Gollob, "Microsoft azure and cloud computing," in *Microsoft Azure*. Springer, 2015, pp. 3–26.

[9] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-Diagonal Parity for double disk failure correction," in *Proc. of the USENIX FAST'04*, San Francisco, CA, March 2004.

[10] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

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

[12] Y. Fu and J. Shu, "D-code: An efficient raid-6 code to optimize i/o loads and read performance," in *Parallel and Distributed Processing Symposium*, 2015, pp. 603–612.

[13] E. E. Gad, R. Mateescu, F. Blagojevic, C. Guyot, and Z. Bandic, "Repair-optimal mds array codes over gf(2)," in *2013 IEEE International Symposium on Information Theory*, July 2013, pp. 887–891.

[14] S. Ghemawat, H. Gobioff, and S.-T. Leung, *The Google file system*. ACM, 2003, vol. 37, no. 5.

[15] Y. Hu, X. Zhang, P. P. Lee, and P. Zhou, "Generalized optimal storage scaling via network coding," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 956–960.

[16] C. Huang and L. Xu, "STAR: An efficient coding scheme for correcting triple storage node failures," *IEEE Transactions on Computers*, vol. 57, no. 7, pp. 889–901, 2008.

[17] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, S. Yekhanin *et al.*, "Erasure coding in windows azure storage." in *Usenix annual technical conference*. Boston, MA, 2012, pp. 15–26.

[18] Y. Jiang, C. Wu, J. Li, and M. Guo, "Bdr: A balanced data redistribution scheme to accelerate the scaling process of xor-based triple disk failure tolerant arrays," in *Computer Design (ICCD), 2016 IEEE 34th International Conference on*. IEEE, 2016, pp. 72–79.

[19] M. Li *et al.*, "GRID codes: Strip-based erasure code with high fault tolerance for storage systems," *ACM Trans. on Storage*, vol. 4, no. 4, p. Article 15, Jan. 2009.

[20] R. Li, X. Li, P. P. Lee, and Q. Huang, "Repair pipelining for erasure-coded storage," in *Proceedings of the 2017 USENIX Annual Technical Conference (USENIX ATC17)*, 2017, pp. 567–579.

[21] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: insights from google compute clusters," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, pp. 34–41, 2010.

[22] S. Mitra, R. Panta, M.-R. Ra, and S. Bagchi, "Partial-parallel-repair (ppr): a distributed technique for repairing erasure coded storage," in *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2016, p. 30.

[23] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang *et al.*, "f4: Facebooks warm blob storage system," in *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*. USENIX Association, 2014, pp. 383–398.

[24] L. Pamies-Juarez, F. Blagojevi, R. Mateescu, C. Gyuot, E. E. Gad, and Z. Bandi, "Opening the chrysalis: On the real repair performance of MSR codes," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*. Santa Clara, CA: USENIX Association, 2016, pp. 81–94.

[25] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," *IEEE Transactions on Information Theory*, vol. 60, no. 10, pp. 5843–5855, 2014.

[26] J. S. Plank, "A tutorial on reed–solomon coding for fault-tolerance in raid-like systems," *Software: Practice and Experience*, vol. 27, no. 9, pp. 995–1012, 1997.

[27] J. S. Plank, M. Blaum, and J. Hafner, "SD codes: Erasure codes designed for how storage systems really fail," in *Proc. of the USENIX FAST'13*, San Jose, CA, February 2013.

[28] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," in *IEEE International Conference on Cloud Computing*. Springer, 2009, pp. 626–631.

[29] 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 *FAST*, 2015, pp. 81–94.

[30] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A hitchhiker's guide to fast and efficient data reconstruction in erasure-coded data centers," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 331–342, 2015.

[31] I. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial & Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[32] Y. Saito, S. Frolund *et al.*, "FAB: Building distributed enterprise disk arrays from commodity components," in *Proc. of the ASPLOS'04*.

[33] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: Novel erasure codes for big data," in *Proceedings of the VLDB Endowment*, vol. 6, no. 5. VLDB Endowment, 2013, pp. 325–336.

[34] D. Tang, X. Wang, S. Cao, and Z. Chen, "A new class of highly fault tolerant erasure code for the disk array," in *Proc. of the PEITS'08*, Guang Zhou, China, August 2008.

[35] C. Tau and T. Wang, "Efficient parity placement schemes for tolerating triple disk failures in RAID architectures," in *Proc. of the AINA'03*, Xi'an, China, March 2003.

[36] M. Vajha, V. Ramkumar, B. Puranik, G. Kini, E. Lobo, B. Sasidharan, P. V. Kumar, A. Barg, M. Ye, S. Narayanamurthy *et al.*, "Clay codes: Moulding mds codes to yield an msr code," in *Proc. 16th USENIX Conference on File and Storage Technologies, Oakland, CA, USA*, vol. 2018, 2018, pp. 139–154.

[37] Y. Wang, G. Li, and X. Zhong, "Triple-Star: A coding scheme with optimal encoding complexity for tolerating triple disk failures in RAID," vol. 8, no. 3, pp. 1731–1472, 2012.

[38] C. Wu *et al.*, "HDP code: A Horizontal-Diagonal parity code to optimize I/O load balancing in RAID-6," in *Proc. of the DSN'11*, 2011.

[39] C. Wu and X. He, "Gsr: A global stripe-based redistribution approach to accelerate raid-5 scaling," in *2012 41st International Conference on Parallel Processing*, Sep. 2012, pp. 460–469.

[40] C. Wu, S. Wan, X. He, Q. Cao, and C. Xie, "H-Code: A hybrid MDS array code to optimize partial stripe writes in RAID-6," in *Proc. of the IPDPS'11*, Anchorage, Alaska, May 2011.

[41] L. Xu and J. Bruck, "X-Code: MDS array codes with optimal encoding," *IEEE Transactions on Information Theory*, vol. 45, no. 1, pp. 272–276, 1999.

[42] M. Ye and A. Barg, "Explicit constructions of optimal-access mds codes with nearly optimal sub-packetization," *IEEE Transactions on Information Theory*, vol. PP, no. 99, pp. 1–1, 2017.

[43] Y. Zhang, C. Wu, J. Li, and M. Guo, "Pcm: A parity-check matrix based approach to improve decoding performance of xor-based erasure codes," in *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2015, pp. 182–191.

[44] ——, "Tip-code: A three independent parity code to tolerate triple disk failures with optimal update complextiy," in *Ieee/ifip International Conference on Dependable Systems and Networks*, 2015, pp. 136–147.

[45] Y. Zhu *et al.*, "On the speedup of single-disk failure recovery in xor-coded storage systems: Theory and practice," in *Proc. of the MSST'12*, San Diego, CA, April 2012.