IR-RBT Codes: A New Scheme of Regenerating Codes for Tolerating Node and Intra-node Failures in Distributed Storage Systems

Jianchao Bian[1]*, Shoushan Luo[1], Wei Li[1] , Yaxing Zha[1] and Yixian Yang[1,2]
[1] National Engineering Laboratory for Disaster Backup and Recovery,
Beijing University of Posts and Telecommunications,
100876, Beijing, P.R. China
[e-mail: bianjianchao@bupt.edu.cn]
[2] State Key Laboratory of Public Big Data, 550025, Guizhou, P.R. China
*Corresponding author: Jianchao Bian

Received October 23, 2017; revised May 8, 2018; revised September 7, 2018; revised October 18, 2018; accepted February 14, 2019; published October 31, 2019

Abstract

Traditional regenerating codes are designed to tolerate node failures with optimal bandwidth overhead. However, there are many types of partial failures inside the node, such as latent sector failures. Recently, proposed regenerating codes can also repair intra-node failures with node-level redundancy but incur significant bandwidth and I/O overhead. In this paper, we construct a new scheme of regenerating codes, called IR-RBT codes, which employs intra-node redundancy to tolerate intra-node failures and serve as the help data for other nodes during the repair operation. We propose 2 algorithms for assigning the intra-node redundancy and RBT-Helpers according to the failure probability of each node, which can flexibly adjust the helping relationship between nodes to address changes in the actual situation. We demonstrate that the IR-RBT codes improve the bandwidth and I/O efficiency during intra-node failure repair over traditional regenerating codes but sacrifice the storage efficiency.

Keywords: distributed storage, data security, erasure codes, regenerating codes, intra-node failure

1. Introduction

Cloud computing has changed our world and the way in which we process data. Increasing amounts of data are being stored in distributed storage systems, rather than traditional file systems or storage arrays. This huge mass of data has presented a big challenge in terms of the reliability and efficiency of the storage system.

Generally, a distributed storage system introduces redundancy to ensure reliability, the most widely used form of which is multiple replications. For example, Facebook's Hadoop cluster employs triple replication to improve data availability, but with low storage efficiency. Hence, more recently, several popular forms (e.g., HDFS-RAID, OceanStore [1]) have used erasure codes, such as Reed-Solomon (RS) codes [2], instead of replication to achieve high fault tolerance and reduce the storage overhead.

Under RS codes, the data are divided into equal units, which we called blocks or strips, such that each k blocks constitutes a data group for computing r parity blocks; these n ($n = k + r$) blocks, which encode together, constitute a stripe. It follows that any fewer than r blocks failure in a stripe can be repaired completely. Thus, RS codes are maximum distance separable (MDS) codes with high failure tolerance, but compute based on complex Galois field arithmetic.

Another family of MDS codes is array code, such as EVENODD [3], RDP (Row-Diagonal Parity) [4], and X-codes [5], which are based entirely on the XOR operation but have low fault tolerance. To achieve higher fault tolerance, strip-based codes, which are called GRID codes [6], have been presented.

However, during data reconstruction, the related method need to read and transfer the k blocks of data in each stripe for decoding; this process has high read I/O and communication overhead. To reduce the bandwidth and I/O that is required during the repair operation, a new class of erasure codes called Locally Repairable Codes (LRCs) have been proposed in [7, 8]. These codes are efficiently repairable and have lower disk I/O; however, they are non-MDS codes and require more storage space than RS codes.

In addition, for investigating the optimal bandwidth consumption of repair in a distributed storage system, which stores the data across nodes in the network, regenerating codes were introduced by Dimakis et al. [9]. Under regenerating codes, which have lower communication overhead than RS codes, the data of a failed node can be repaired by connecting to d ($d > k$) helper nodes and transferring fewer data. However, to generate the help data, the helper nodes need to read the related data, which result in an increased number of I/O requests and additional local computational overhead.

In [10, 11], the authors present a new class of codes called repair-by-transfer (RBT) codes, which compute and store the help data in the encoding phase. During a repair operation, the data collector only reads and transfers the help data from helper nodes to repair a failure that can reduce the number of I/O requests.

In recent years, researchers have done a lot of work on the actual deployment of regenerating code. Goparaju et al. introduce a scheme of systematic repair MSR codes with no restrictions on the parameters n, k, d in [12]. In [13], Sasidharan et al. construct a high-rate MSR code with small field size. Optimal cross-rack repair bandwidth for data center discussed in [14, 15].

All above regenerating codes schemes optimize the performance of single node failure, but not optimal for multiple node failures. A mutually cooperative recovery (MCR) mechanism

was presented by Hu et al. in [16], and a lower bound of repair bandwidth based on MCR was obtained. The generalization of the cooperative regenerating codes was presented in [17], which also achieve all parameters of the minimum-bandwidth point.

However, the above methods only provide device-level or node-level redundancy for tolerating device or node failure, which means that all data in the device or node are temporarily unavailable or permanently lost. The failures of disk drives can be classified into two principal types: device failures [18, 19] and sector failures [20, 21]. A node failure can be the equivalents of a device failure under an erasure code, while a sector failure is the data loss of a particular disk sector. Hence, traditional erasure codes (regenerating codes) must use device-level (node-level) redundancy to repair sector failures with high space overhead.

To tolerate both device and sector failures with efficient overhead, sector-disk (SD) codes [22, 23], PMDS codes [24] and STAIR codes [25] have been proposed. These codes employ sector redundancy to protect against sector failure with lower space overhead, but do not address the problem of communication overhead during the repair operation. In contrast, traditional regenerating codes reduce the communication overhead, but do not take into account the optimization of intra-node failure repair performance in the distributed storage system.

In this paper, we propose a new scheme of regenerating codes for protecting against both node and intra-node failures (such as sector failure) with lower I/O request and network bandwidth overhead. We also propose algorithms for assigning the intra-node redundancy and RBT-Helpers according to the failure probability of each node.

The rest of the paper is organized as follows. In Section 2, we introduce the background of regenerating codes. In Section 3, the code model is described, in which any regenerating code that satisfies a specified property can improve the performance of intra-node failure repair. Algorithms that assign helpers for each node are also introduced in Section 3. In Section 4, numerical analysis is carried out to evaluate the performance of the code model. A conclusion is drawn in Section 5.

2. Preliminaries

2.1 Notation and Terminology

We consider a distributed storage system, in which the storage space of each node has been logically segmented into a set of strips of the same size. Before being stored in the system, the data are first to split into data blocks (which are the same as strips). Then, each set of k data blocks is encoded to generate r parity blocks, and all n blocks are stored in n nodes. We refer to each of the n strips that encode together as a stripe, as illustrated in **Fig. 1**.

For encoding, each of the k data blocks is split into w symbols (also known as sector), and each stripe is encoded independently. Hence, the smallest granularity of the data in the system is a symbol, of which the size is dependent on the finite-field arithmetic. For simplicity, our discussion focuses on a single stripe. The node failure is mapped to the block failure (also known as strip failure) in the stripe and the intra-node failure can be mapped to a symbol failure (also known as sector failure) in the strip.

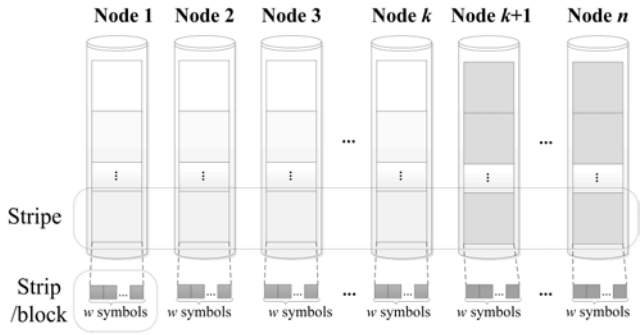


Fig. 1. A stripe of the storage system under regenerating codes

2.2 Regenerating Codes

Regenerating codes are associated with the parameter set (n, k, d, w, β). This means that each block stores w symbols in one stripe, and the data collector connects any d blocks and downloads β symbols of help data from each block during the repair operation; thus, the network bandwidth is $\gamma = d\beta$.

As shown in [9], the repair operation can be interpreted as multicasting over an information flow graph to characterize the tradeoff between storage and repair bandwidth. The authors use the max-flow-min-cut theorem to obtain two extremal points, which are termed the minimum-storage regenerating (MSR) and minimum-bandwidth regenerating (MBR) points. The codes that attain the MSR point achieve optimal storage overhead and lower network bandwidth overhead, which called MSR codes. Let B be the original data symbols that are stored in one stripe. The parameters w and β satisfy

$$(w_{MSR}, \gamma_{MSR}) = \left(\frac{B}{k}, \frac{Bd}{k(d-k+1)} \right). \tag{1}$$

In this paper, we assume $\beta = 1$, which represents that the help data consist of one symbol from each helper node; thus, $w = d - k + 1$. However, if we can construct an MSR code with $\beta = 1$, then we can construct an MSR code for any larger value of β.

The encoded nw symbols are represented as an $(n \times w)$ code matrix C, in which the ith row c_i^t represents the w symbols that are stored in the ith block. The code matrix is computed by $C = \Psi M$, where Ψ is the encoding matrix and M is the message matrix.

When repairing the failure block f, the data collector chooses any d nodes from the other $(n - 1)$ blocks to be helpers; the set which consists of these d helpers is denoted as H. Each helper in set H needs to read all w symbols and compute the help data for failure block f; the help data are computed by

$$\theta_{hfH} = \phi_{fH}^t c_h, \tag{2}$$

where ϕ_{fH}^t is the repair vector of failure block f with the helper set H, θ_{hfH} denotes the help data from block h, and c_h is the vector of the w symbols that are stored in block h.

Then, the data collector can use the help data to repair the failure block f; more detail can be found in [26].

2.3 Repair-by-transfer Model

Regenerating codes reduce the communication overhead during the repair operation, but increase the read I/O, as they need to read all w symbols in each of the d helpers. If we have previously computed help data for block f during the encoding process, helpers can only read

and transfer the help data to repair block *f*. The failure block *f* is said to utilize the repair-by-transfer (RBT) model, and if all the blocks can be repaired by the RBT model, we called this coding scheme an RBT code.

The MSR codes that are transformed into the RBT codes must satisfy two properties [10]:

(i) The help data from a helper are unrelated to the choice of the remaining $(d-1)$ helpers.

(ii) Any w of the $(n-1)$ the repair vectors are linearly independent.

Hence, the help data from block *h* for block *f* is unrelated to *H*, and Equation (2) can be rewritten in the form

$$\theta_{hf} = \phi_f^t c_h . \tag{3}$$

We assume that the set *F* consists of all the failure blocks that are helped by block *h*, block *h* computes the help data for all blocks in set *F* with

$$\theta_{hF} = \phi_F c_h , \tag{4}$$

and block *h* stores θ_{hF} and replaces the original data c_h. We called block *h* the RBT-Helper of the blocks in set *F* and θ_{hF} the RBT-Help data. If all the blocks have enough RBT-Helpers, then we have transformed the MSR codes to the RBT codes.

3. The Code Model Description

In this section, we will build a new regenerating coding scheme based on MSR codes with optimal intra-node failure repair performance. First, we provide the coding scheme and prove the feasibility. Then, we design the algorithms for the intra-node redundancy assignment and RBT-Helper selection.

3.1 Repair by Transfer and Intra-node Failure Repair

The recently proposed MSR and RBT codes are designed to tolerate total data loss in the node and need to connect to *d* blocks when repairing intra-node failure. We can employ intra-node redundancy symbols (abbreviated as IR symbols), which are computed by *w* symbols that are stored in one block, to tolerate the failure inside the node, but this will lead to a decline in storage utilization.

The RBT-Help data are linear combinations of *w* symbols, which are stored in the helper block, such as θ_{hF}, which is computed with c_h by Equation (4). If we can use the RBT-Help data θ_{hF} to tolerate the intra-node failure in block *h*, the read I/O overhead during repair the block failure of the MSR codes will be reduced and better performance on intra-node failure repair will be achieved.

To make the RBT-Help data tolerant of intra-node failure, the MSR codes must satisfy two properties in [10] and the repair vector of the MSR codes must satisfy property 1.

Property 1: Any *w* rows of $\begin{pmatrix} I \\ \phi_F \end{pmatrix}$ are linearly independent, where *I* is the $w \times w$ identity matrix.

As mentioned above, one block stores the *w*-symbol data c_h and $|F|$-symbol RBT-Help data θ_F, and $\begin{pmatrix} c_h \\ \theta_F \end{pmatrix}$ can be computed by

$$\begin{pmatrix} I \\ \phi_F \end{pmatrix} \cdot c_h = \begin{pmatrix} c_h \\ \theta_F \end{pmatrix} . \tag{5}$$

We assume that e ($e < |F|$) symbols are lost. We can construct the residual matrix by deleting the rows of $\begin{pmatrix} I \\ \phi_F \end{pmatrix}$ that correspond to the failed symbols.

By property 1, any $w \times w$ submatrix of the residual matrix is invertible; we denote such submatrix as D. Hence, we can reconstruct c by using the inverse matrix of D and w surviving symbols, similar to the decoding operation of RS codes with $\begin{pmatrix} I \\ \phi_F \end{pmatrix}$ as the encoding matrix.

Hence, any MSR code that satisfies property 1 and the two properties in [10] can use the RBT-Help data to obtain IR symbols that can tolerate intra-node failure; we call such codes IR-RBT codes. In **Fig. 2**, a stripe model of IR-RBT codes is shown for n=12, k=6, d=10, and w=5.

The first w lines are encoded data of MSR codes. The first k columns are data strips and the rest of the columns are parity strips. As mentioned above, the IR symbols are generated by Equation (3) with w symbols in each strip. As illustrated in **Fig. 2**, strips 0 to 7 employ 2 IR symbols each; therefore, these strips can tolerate two symbol failures each, such as the data loss of $c_{4,3}$, $c_{4,4}$ in strip 4. The rest of the strips each have only 1 IR symbol; therefore, these strips can tolerate 1 symbol failure, such as error of $c_{10,3}$ in strip 10.

In contrast, the first IR symbols $\theta_{*,11}$ in strips 0 to 11 are also the RBT-Help data for strip 11. When strip failure occurs in the strip 11, the data collector can choose any 10 of these 11 strips to be helpers and only read these first IR symbols, thereby imposing low read I/O.

Only strips 0 to 6 and strip 11 have RBT-Help data $\theta_{*,10}$ for strip 10, which means that strip 10 does not have enough RBT-Helpers to be repaired by the RBT model. During the repair, the data collector must choose 2 extra strips from strips 7 to 9 to compute help data. However, except strips 10 and 11, no other strips have RBT-Helpers.

The number of deployed IR symbols determines the capacity of the intra-node failure tolerance and the performance of the read I/O; we will discuss this in more detail in Section 3.3.

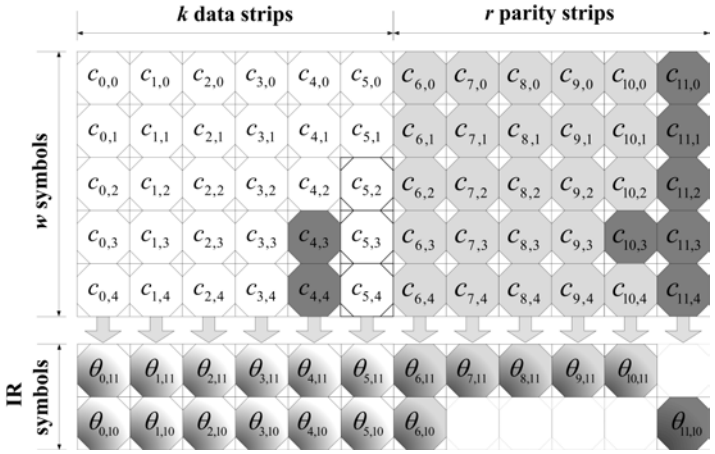


Fig. 2. A stripe model of IR-RBT codes for n=12, k=6, d=10, w=5

3.2 An Example IR-RBT Code

In this section, we give an example IR-RBT code that is based on the product-matrix-MSR (PM-MSR) codes. When we choose the Vandermonde matrix to be the encoding matrix Ψ, the PM-MSR codes satisfy Property 1. We define $d = 2k - 2$ and the encoding matrix Ψ as

$$\Psi = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{d-1} \\ 1 & \alpha_2 & \alpha_2^2 & \cdots & \alpha_2^{d-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \cdots & \alpha_n^{d-1} \end{pmatrix}. \tag{6}$$

Under the PM-MSR codes, we have $\Psi = [\phi \quad \Lambda\phi]$, more detail on the encoding can be found in [26]. We have $d = 2w$, and the repair matrix ϕ is given by

$$\phi = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{w-1} \\ 1 & \alpha_2 & \alpha_2^2 & \cdots & \alpha_2^{w-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \cdots & \alpha_n^{w-1} \end{pmatrix}, \tag{7}$$

where

$$\Lambda = \begin{pmatrix} 1 & & & & \\ & \alpha_1^w & & & \\ & & \alpha_2^w & & \\ & & & \ddots & \\ & & & & \alpha_n^w \end{pmatrix}. \tag{8}$$

The repair matrix ϕ is also a Vandermonde matrix; thus, any w rows of $\begin{pmatrix} I \\ \phi \end{pmatrix}$ are linearly independent. Because ϕ_F is submatrix of ϕ, any w rows of $\begin{pmatrix} I \\ \phi_F \end{pmatrix}$ are linearly independent.

Hence, under a PM-MSR code, if we employ the Vandermonde matrix as the encoding matrix, we can use the repair matrix as the generator matrix of intra-node redundancy; then, the intra-node redundancy can not only be utilized for tolerating intra-node failure but also serve as the RBT-Help data for other blocks.

3.3 IR Symbol Deployment Policies

In this section, we will discuss the IR symbol deployment policies, to determine how many IR symbols each stripe needs.

As the number of deployed IR symbols increases, the system can achieve better intra-node failure tolerance and read I/O performance during repair node failure. We can divide this process into 3 stages:

(i) Basic intra-node failure tolerance

If each block only has one IR symbol, the blocks that have basic intra-node failure tolerance can tolerate one intra-node failure. Hence, in this case, the number of IR symbols in each stripe is n, and only $\lfloor n/d \rfloor$ blocks can be repaired by the RBT model at most; in other words, each block can use one RBT-Helper, on average.

(ii) Degraded read

In the storage system, many repair operations occur in reading requests from the user.

During a read operation, the system may not be able to read all the raw data directly, and some data need to be repaired, this type of reading operation is called degrade reads. Therefore, it is important to ensure the *k* data blocks can be repaired by the RBT model. The system needs to assign *d* RBT-Helpers to each of the *k* data blocks, so *dk* IR symbols must be employed. Each block can tolerate *dk/n* intra-node failures on average.

(iii) All-RBT repair model

If we ensure that every node can be repaired by the RBT model (which is called the all-RBT repair model), as with RBT codes, the system needs to assign *d* RBT-Helpers for each block. This means that *dn* IR symbols are employed in each stripe. In this case, the system achieves optimal read I/O overhead during the repair operation and each block can protect against *d* intra-node failures, on average.

As mentioned above, if more IR symbols are deployed, more RBT-Help data can be used, and the read I/O overhead can be reduced, but with lower storage utilization. However, in practice, the storage space of the system is limited; thus, the actual situation must be taken into account when setting the IR symbols.

We set the storage utilization of the system as δ, which has a lower limit $\delta_{\min}$, and x represents the number of IR symbols in one stripe. Thus, we have

$$\delta = \frac{kw}{nw + x}, \tag{9}$$

and the maximum number of IR symbols in one stripe can be computed as follows:

$$x_{\max} = \frac{kw}{\delta_{\min}} - nw. \tag{10}$$

Hence, the system can set the value of x according to the desired repair performance, and the value must be less than $x_{\max}$. However, a limited number of IR symbols may not be able to ensure that all nodes have enough RBT-Help data for repair by the RBT model. Thus, developing effective distribution mechanisms for achieving optimal repair performance under limited resources is important. We will discuss this in the next section.

3.4 IR Symbol and RBT-Helper Assignment

In this section, we will present 2 algorithms for assigning the limited number of IR symbols and the helping relationship between the blocks based on the failure probabilities of the blocks, to achieve optimal repair performance. We assume that block *i* may be unavailable with probability μ_i ($0 \le \mu_i \le 1$) when the data in block *i* is temporarily unavailable or permanently lost. The failure probability μ is obtained according to the actual operation of the system, such as counting the failure times or reconstruction times of each node in unit time, and the failure probability is calculated on this basis.

Then, Algorithm 1 is presented to compute the number of IR symbols in each block. First, Algorithm 1 assigns one IR symbol to each block (Line 2), to ensure that each block can tolerate at least one symbol failure. Then, more IR symbols are assigned to the block, which has the lower failure probability μ, so that the failure block can be repaired by RBT-Helpers with higher probability.

Algorithm 1 Algorithm for computing the number of IR symbols of each block

1. Set sum_num_ir = *x*
   //Total number of IR symbols to be allocated in a stripe, which initial value is *x*, computed by equation (10)
2. Set num_ir[*i*] = 1 with *i* = *n* to 1
   // The number of IR symbols in each block，which initial value is 1

3. Set num_rbt_helpers = 0 for each block
4. Let E be the set and the initial value be empty set
5. for sum_num_ir = $x - n$ to 1 do
6. for $i = n$ to 1 do
7. if num_ir[i] < $\lfloor(x - \text{sum\_num\_ir})/n\rfloor+1$ then
8. Let the block i in the set E
9. end if
10. end for
11. Let μ_j be the min μ of blocks in the set E
12. Set num_ir [j] = num_ir[j]+1
13. end for

Under Algorithm 1, the number of IR symbols for each block is determined. Then, we need to define the helping relationship under the repair operation. One option is an even distribution: block i can be the RBT-Helper for block {i+1, ..., i+ num_ir[i]} mod n, which is called the Cyclic(CYC) scheme, where num_ir[i] is the number of IR-symbols of block i.

However, the CYC scheme does not consider the differences among the failure probabilities of the blocks. Algorithm 2 is designed to select the RBT-Helpers for each block according to the failure probability: blocks with low failure probabilities are chosen to be helpers previously, while blocks with high failure probability are assigned more helpers.

Algorithm 2 Algorithm for RBT-Helper assignment

1. Set num_rbt_helper = 0 for each block
2. Let H be the set of blocks that can be RBT-Helpers
3. Set H consist of all n blocks
4. Let H_i be the set of RBT-Helpers for block i
5. Set initial value of H_i is an empty set, which $i \in \{1, 2, \ldots, n\}$
6. Let φ_i [H_i] be the probability average of the RBT-Helpers that block i can connect from H_i
// Computed by equation (11)
7. Set Δ_i be the increments of φ_i [*] with different RBT-Helpers sets
8. Let F_i be the set of failure blocks that block i can help with IR symbols
9. Set F_i consist of all n blocks without block i
10. for sum_num_rbt_help = x to 1 do
//Total number of the RBT-Help symbols in a stripe, which number is the same as IR symbols
11. Let block h be a block in set H with the minimum value of μ
12. compute $\Delta_i = \varphi_i[H_i|h] - \varphi_i[H_i]$ for each block in F_h
13. Let block f be a block in set F_h with the maximum value of Δ
14. Add block h in set H_f
15. Remove block f from F_h
16. if num_ir [h] > 1 then
17. Set num_ir [h] = num ir [h]-1
18. else
19. Remove block h from the set H
20. end if
21. end for

Algorithm 1 assigns IR symbols for each block, these IR symbols will become RBT-Help data during repair operation, called RBT-Help symbols in Algorithm2. First, the algorithm assigns the RBT-Help symbols one by one in the block that has the lowest failure probability (Line 9). When each RBT-Help symbol is assigned, the other blocks calculate the probability average of the number of RBT-Helpers that can be connected if the block obtains this RBT-Help symbol (Line 12). $\varphi_i\left[H_i\right]$ can be computed by

$$\begin{aligned}\varphi_i\left[H_i\right] &= hP_{i,h}\left[H_i\right]+(h-1)P_{i,h-1}\left[H_i\right]+\cdots+P_{i,1}\left[H_i\right] \\ &= \sum_{j=1}^{h} jP_{i,j-1}\left[H_i\right]\end{aligned}, \tag{11}$$

where $|H_i| = h$. $P_{i,j}[H_i]$ is the probability of block i connecting to j RBT-Helpers for repair when the other (h - j) RBT-Helpers are unavailable. The $P_{i,j}[H_i]$ can be computed as follows:

$$\begin{aligned}P_{i,j}[H_i] &= \mu_i(1-\mu_{h_1})(1-\mu_{h_2})\cdots(1-\mu_{h_j})\mu_{h_{j+1}}\cdots\mu_{h_h} \\ &+\mu_i\mu_{h_1}(1-\mu_{h_2})\cdots(1-\mu_{h_{j+1}})\mu_{h_{j+2}}\cdots\mu_{h_h}+ \\ &\mu_i\mu_{h_1}\mu_{h_2}(1-\mu_{h_3})\cdots(1-\mu_{h_{j+2}})\mu_{h_{j+3}}\cdots\mu_{h_h}+\cdots,\end{aligned} \tag{12}$$

where $H_i = \{h_1, h_2, ..., h_h\}$. Let $H_{i,m}$ be a subset of H_i, which denotes the set of connected helpers, where $1 \le m \le c_h^j$. We can rewrite (12) as:

$$P_{i,j}[H_i] = \mu_i \cdot \sum_{m=1}^{C_h^j} \prod_{l\in H_{i,m}} (1-\mu_{h_l}) \prod_{l\in(H_i-H_{i,m})} \mu_{h_l}. \tag{13}$$

The assignment is determined by increments of the probability average of the RBT-Helpers that can be connected by each block, which is denoted as Δ. Then, the algorithm assigns the RBT-Help symbols in the other blocks in ascending order of failure probability. Through Algorithm 2, the coding scheme achieves the maximum probability by using more RBT-Helpers during the repair operation.

In the case of a degraded read, there will be more repair operations of the original data blocks. Then, Algorithm 2 can only assign RBT-Helpers for the original data blocks to ensure that the original data is repaired, with a high probability of using more RBT-Helpers.

The above is the normal situation, but in the initialization stage of the system, due to the lack of actual operation data support, it is temporarily unable to allocate according to the failure probability μ.

At this point, the distribution of IR-symbols can follow the principle of uniform distribution, such as each block has one IR symbol, so that all blocks have the basic intra-node failure tolerance as mentioned in 3.3. The repair relationships among nodes can be determined by CYC scheme temporarily.

In the subsequent operation of the system, the failure times of each node is counted to calculate the failure probability, and the encoded data can be dynamically adjusted. For example, the system can use algorithm 1 to add or modify the number of IR-symbols in some nodes, and algorithm 2 to adjust the repaired relationship between nodes to achieve better performance.

4. Results and Discussion

In this section, we compare the proposed method with other typical erasure codes in terms of intra-node failure repair and storage utilization. Then, the effect of the RBT-Helper assignment algorithms will be analyzed.

4.1 Evaluation Setting

We consider a distributed storage system with 12 nodes. To tolerate node and intra-node failure, we employ the IR-RBT codes with parameters n = 12, k = 6 for encoding data prior to storage in the system. The employed arithmetic is Galois Field arithmetic over 8-bit symbols,

which is termed GF(2^8). Since $2k - 2 \leq d \leq n - 1$, we have d = 10, 11 and w = 5, 6, correspondingly.

In order to compare validly, we will compare IR-RBT codes with these schemes under the same parameters, include the (n,k,d,w) MSR and RBT codes, (n,k) RS and STAIR codes.

Although RS codes and STAIR codes are very different from regenerating codes, w can be set as the number of symbols in one block in a stripe which value is consistent with w in the scheme of IR-RBT codes.

In addition, the comparison scheme also includes MCR codes. Since MSR codes are the optimal storage scheme of regenerating codes, both RBT codes and IR-RBT codes are based on MSR codes. Therefore, for the validity of experimental comparison, we also adopt the optimal storage scheme of MCR codes to make the comparison, which called minimum storage collaborative regenerating codes (MSCR codes)[17] with parameters (n,k,d,t), The parameter t represents the number of nodes for cooperative repair, the value is $t = n - d$.

4.2 Single Intra-node Failure Repair Performance

A comparison of the amounts of data read and transfer from all helpers in one stripe during the single intra-node failure (symbol failure) repair operation is shown in **Fig. 3** and **Fig. 4**.

To tolerate single intra-node failure, the number of IR-symbols x is set to n, which means that each block has 1 IR-symbol. Because of the addition of IR symbols, IR-RBT codes can repair symbol failure using other symbols in the failure block with no bandwidth consumption. IR-RBT codes only need to read w symbols in the failure block, 6 symbols in this case, and no data transfer from other blocks. IR-RBT codes achieve better performance on reading I/O and bandwidth.

Although STAIR codes are optimized for sector failure (same as symbol failure) repair, they also need to connect k blocks to read and transfer k symbols from other blocks. However, RS codes must decode all the data blocks with the highest bandwidth overhead in these schemes, which mean RS codes need to connect k blocks, read and transfer kw symbols, 36 symbols in this example.

MSR, RBT codes need to repair the entire block, therefore, the bandwidth overheads are the same as those for repairing a block failure. It means that MSR and RBT codes must connect d blocks to read $d\alpha$ and $d\beta$ separately, but both only transfer $d\beta$ symbols. Hence, MSR codes must read 66 symbols in 11 helpers to compute help data; thus, they have the highest read I/O in these schemes. RBT codes are optimized for reading I/O based on MSR codes, which read less data than MSR and RS codes.

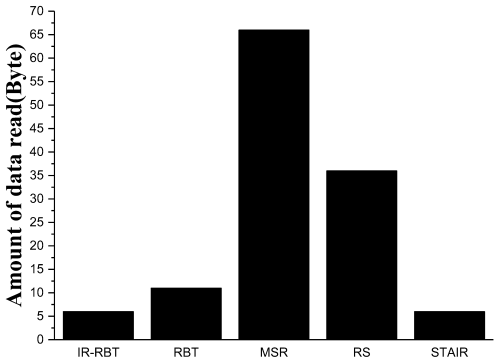


Fig. 3. Total amounts of data read from all helpers under repair operation of the single intra-node failure for n=12, k=6, d=11 and w=6.

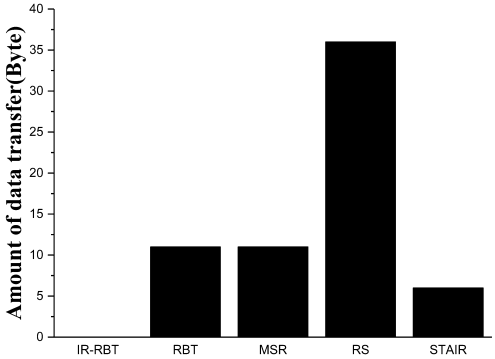


Fig. 4. Amounts of data transfer from all helpers under the repair operation of the single intra-node failure for n=12, k=6, d=11 and w =6.

Fig. 5 shows a comparison of the amounts of data read from the system during single intra-node failure repair operation for different values of k, with $d = 2k - 1$ and $n = d + 1$. We can see that IR-RBT codes and STAIR codes read the least data from the system, and the amount grows slowly with k.

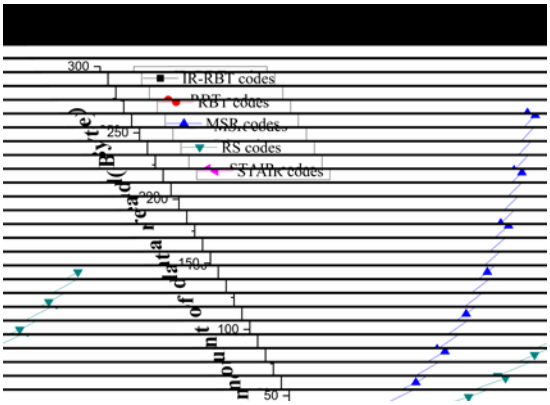


Fig. 5. Comparison of data read under the repair operation for various values of k, with $d = 2k - 1$ and $n = d + 1$.

Because of IR-RBT codes and STAIR codes only need to read k and w symbols, which $w = d - k + 1 = k$. The data read under RBT codes also are the linear relationship with k, because $d\beta = 2k - 1$.

As mentioned above, RS codes need to read $kw = k^2$ symbols, and MSR codes need to read $dw = k(2k-1) = 2k^2 - k$ symbols. Hence, the MSR codes need to read more data from helpers and the growth rate is fast; the growth rate for the RS codes is slightly lower than that for the MSR codes.

When the change of k considered, compared with the related schemes, the IR-RBT codes achieve optimal intra-node failure repair performance.

4.3 Multiple Intra-node Failure Repair Performance

In the above section, single intra-node failure repair performance was compared. When multiple nodes occur intra-node failure, the repair performance will be changed, we will discuss in this section.

We consider 2 nodes occur a single intra-node failure, **Fig. 6** and **Fig. 7** show comparisons of the performance of reading I/O and bandwidth during the repair operation. Because of IR-symbols, IR-RBT codes only need to read 2*w* data from these 2 nodes without transferring data.

Like repairing single intra-node failure, another scheme needs to repair the data of total 2 blocks. Among them, the performance of RS codes has not changed, all the data will be decoded during repair.

MSR and RBT code is optimal for single node failure, the overhead under this case is twice to a single failure. MSCR codes are optimal for multiple node failures, reduce the bandwidth overhead a lot, but MSCR codes need to have high read I/O.

Hence, IR-RBT codes achieve better performance during multiple intra-node failure repair.

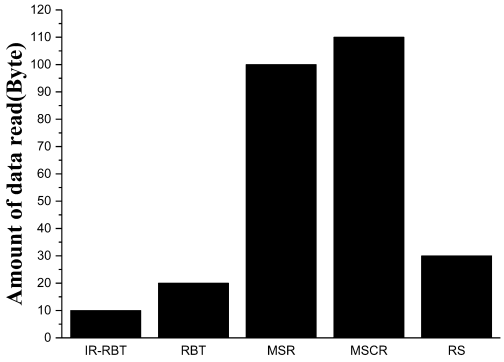


Fig. 6. Total amounts of data read from all helpers under the repair operation of 2 intra-node failures in different nodes for n=12, k=6, d=10, t=2 and w=5.

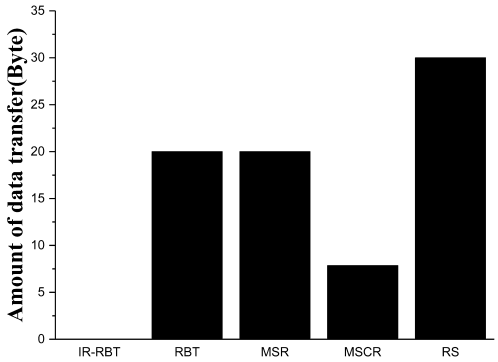


Fig. 7. Amounts of data transfer from all helpers under the repair operation of 2 intra-node failures in different nodes for n=12, k=6, d=10, t=2 and w =5.

4.4 Storage Utilization

As mentioned above, the IR-RBT codes employ IR symbols to tolerate intra-node failure, which reduces storage utilization. In this section, we will discuss the extreme cases of storage utilization in the different repair models that are mentioned in Section 3.3, to determine how much the storage utilization drops under the IR-RBT codes. To facilitate the discussion, the failure probabilities of the blocks are set to the same value, and the IR symbols and RBT-Helpers are uniformly distributed.

4.4.1 Basic Intra-node Failure Tolerance

In this case, each block has a basic ability to tolerate one intra-node failure and $x = n$. Hence, we have

$$\delta = \frac{kw}{n(w+1)} = \frac{k}{n} \cdot \frac{w}{w+1}. \tag{14}$$

It is proved in [27] that there does not exist an exact repair① MSR code for $\beta = 1$ and $d < 2k - 3$. We have $2k-2 \le d \le n-1$ and $w = d - k + 1$, so the value range of storage utilization can be given by

$$\frac{k}{n} \cdot \left(1 - \frac{1}{k}\right) \le \delta \le \frac{k}{n} \cdot \left(1 - \frac{1}{n-k+1}\right). \tag{15}$$

Compared to MSR codes, the storage utilization of IR-RBT codes decreases by $1/k$ at most, but each block can tolerate one symbol failure and connect to one RBT-Helper on average during the repair operation, with lower read I/O.

①Regenerating codes have 2 types, functional regeneration and exact-regeneration. By exact-regeneration, the data repaired by the system was consistent with the data stored on the node originally. The schemes we discuss in this article are exact-regenerating codes.

4.4.2 Degraded Read

Under the degraded read case, the system only assigns enough RBT-Helpers for k data blocks to ensure that the original data blocks are repaired with lower read I/O; this means that dk IR symbols must be employed. Since $x = dk$,

$$\delta = \frac{kw}{nw + dk} = \frac{k}{n} \cdot \frac{1}{1 + \frac{kd}{nw}}, \tag{16}$$

$$\frac{k}{n} \cdot \frac{1}{1 + 2 \cdot \frac{k}{n}} \le \delta \le \frac{k}{n} \cdot \frac{1}{1 + \frac{k-1}{n-k} \cdot \frac{k}{n}}. \tag{17}$$

In this case, δ is reduced to $\frac{1}{1 + 2 \cdot \frac{k}{n}}$ of the MSR code at most; however, each of the k data blocks can be repaired by the RBT model.

4.4.3 All-RBT Repair Model

Under the All-RBT repair model, all blocks can be repaired by the RBT model, as with the RBT codes. Since $x = nd$, the storage utilization is

$$\delta = \frac{kw}{n(w+d)} = \frac{k}{n} \cdot \frac{1}{1 + \frac{d}{w}}, \tag{18}$$

$$\frac{k}{n} \cdot \frac{1}{3} \le \delta \le \frac{k}{n} \cdot \frac{1}{2 + \frac{k-1}{n-k}}. \tag{19}$$

In this case, IR-RBT codes achieve the optimal bandwidth overhead as RBT codes, but δ is reduced to 1/3 of the RBT codes at most. The IR-RBT codes improve the repair performance of the intra-node failure by sacrificing part of the storage space; however, they also optimize

the performance of reading I/O.

Fig. 8 shows the changes in storage utilization in different cases. We fix n = 12, k = 6 and vary the number of IR symbols with d = 10 or 11. When x=0, the IR-RBT codes transform to MSR codes, which the storage utilization is k/n = 0.5. Under the same parameters, the storage utilization of RBT codes and MSCR codes also is k/n = 0.5.

According to **Fig. 8**, the best case is $x = n$, in which the storage utilization is 0.42 and 0.41 for d = 11 and 10, and each block only can tolerate one intra-node failure and use one RBT-Helper during repair. When $x = dk$, each of the k original data blocks can be repaired by the RBT model and each block can tolerate dk/n intra-node failures, while the storage utilization drops to 0.26 and 0.25. In the worst case, all blocks can be repaired by the RBT model and the storage utilization is reduced to 0.17 and 0.16 (almost 0.35 and 0.33 of MSR/RBT codes), but the best read I/O overhead during block failure repair is achieved.

Although the IR-RBT codes have low storage utilization when maintaining the same I/O performance as RBT codes, they can adjust the helping relationship between blocks at any time without modifying the original data, whereas the RBT codes need to encode the data again. More importantly, the IR-RBT codes have better repair performance of intra-node failure.

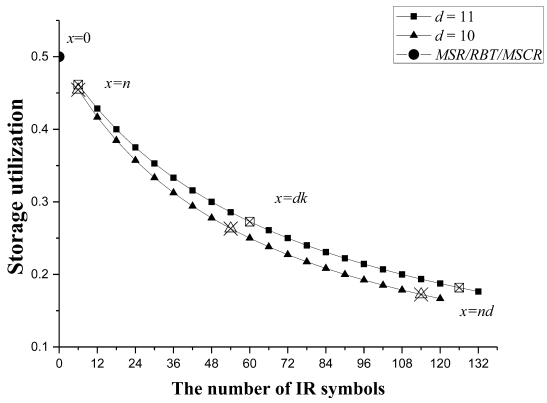


Fig. 8. The storage utilization of IR-RBT codes for various numbers of IR symbols.

4.5 RBT-Helper Assignment

To analyze the effect of the RBT-Helper assignment algorithms, we set the lower limit of the storage utilization $\delta_{\min}$ = 0.3, and the failure probability of each block, which is chosen randomly, is listed in **Table 1**. In the following sections, the codes are evaluated under the above parameters.

As discussed earlier in Section 3.3, employing RBT-Helper to repair data can greatly reduce the read I/O, so we can measure the read I/O performance by the number of RBT-Helpers that are used during the repair operation. In other words, the more RBT-Helpers that are used during the repair process, the smaller the read I/O is.

Table 1. Failure probability of each block

	Block 1	**Block 2**	**Block 3**	**Block 4**	**Block 5**	**Block 6**
μ	0.008	0.011	0.002	0.005	0.001	0.012
	Block 7	**Block 8**	**Block 9**	**Block 10**	**Block 11**	**Block 12**
μ	0.015	0.004	0.01	0.017	0.014	0.006

Due to storage utilization, not all blocks can repair by RBT model in the IR-RBT coding scheme. To make the system use more RBT-Helpers with higher probability during the repair operation, finite RBT-Helper assignment becomes very important. In Section 3.3, we proposed Algorithm 2. Unlike the CYC scheme, Algorithm 2 assigns RBT-Helpers according to the node failure probability to achieve better performance.

Fig. 9 compares the numbers of RBT-Helpers that each block is assigned by Algorithm 2 and the CYC scheme. As shown in **Fig. 9**, blocks 6, 7, 10, and 11, which have larger failure probabilities, are assigned more RBT-Helpers under the IR-RBT coding scheme. Under the CYC scheme, the number of RBT-Helpers in each block is the same, which ensures that blocks with low failure probability can also use RBT-Helpers.

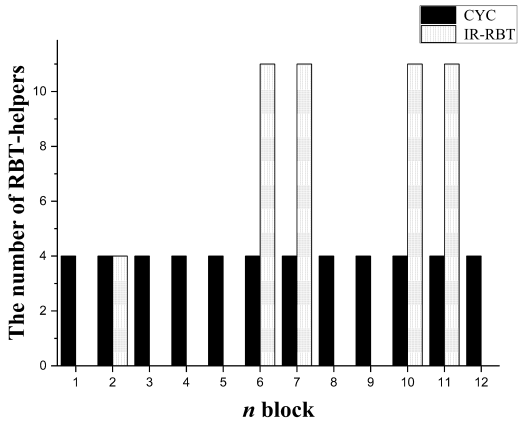


Fig. 9. The numbers of RBT-Helpers for different RBT-Helper assignments.

In **Fig. 10,** we compare the effects of the two schemes in terms of the probabilities of different numbers of RBT-Helpers that the failure blocks can connect during the repair operation. According to **Fig. 10**, the IR-RBT scheme can connect 11 RBT-Helpers with the highest probability; as long as $4 < d < 11$, this probability decays as d is reduced, but increases by a large value when d becomes less than 5. The reason for this behavior is that the IR-RBT scheme assigns 10 RBT-Helpers to blocks 6, 7, 10, and 11, 4 RBT-Helpers to block 2. These 5 blocks can use 4 RBT-Helpers for repair, but only blocks 6, 7, 10, and 11 can connect more than 4 RBT-Helpers.

Although the CYC scheme has a greater probability of using 4 RBT-Helpers for repair, it is impossible to connect more than 4 RBT-Helpers nodes, because each node is assigned only 4 RBT-Helpers under the CYC scheme. In addition, the CYC scheme cannot assign the RBT-Helpers flexibly according to the actual situation, especially when the probability differences among the blocks are large.

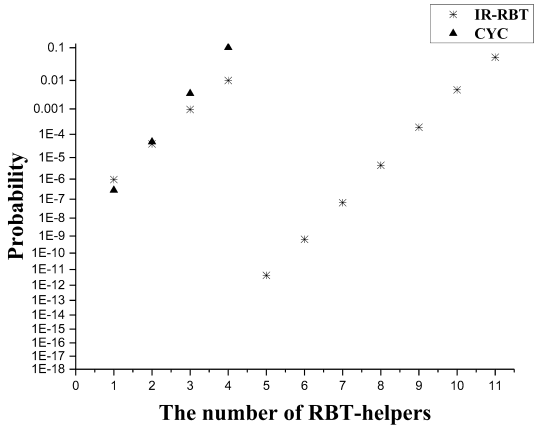


Fig. 10. The probabilities of different numbers of RBT-Helpers that failure nodes can connect

In contrast, the IR-RBT scheme assigns RBT-Helpers more reasonably based on the failure probability, so that the system can use more RBT-Helpers with higher probability during the repair operation. The IR-RBT scheme can assign the RBT-Helpers for the original data blocks prior to repair in the case of the degraded read model. In addition, the IR-RBT scheme can adjust the RBT-Helper assignment without affecting the original data when the blocks' access situation changes.

5. Conclusion

In this paper, new regenerating codes that take into account both intra-node failure repair and read I/O are presented. The IR-RBT codes employ IR symbols to tolerate the intra-node failure and serve as the RBT help data for other blocks. Any MSR code that satisfies the specified property can be converted into an IR-RBT code. In this work, we also design 2 algorithms for assigning the IR symbols and RBT-Helpers according to the failure probability, which can change the number of IR symbols to achieve the desired repair performance and adjust the assignment of RBT-Helpers for each block to address changes in block access.

Our results also indicate that the IR-RBT codes achieve better performance on intra-node failure repair and read I/O than related coding schemes, but sacrifice the storage utilization. Furthermore, the IR-RBT codes can flexibly adjust the repaired relationship between blocks according to the actual situation of the system, while other RBT codes must re-encode the data.

References

[1] Kubiatowicz J, Bindel D, Chen Y, Czerwinski S, Eaton P, Geels D, Gummadi R, Rhea S, Weatherspoon H and Wells C, "Oceanstore: An architecture for global-scale persistent storage," in *Proc. of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.190–201, 2000. Article (CrossRef Link).

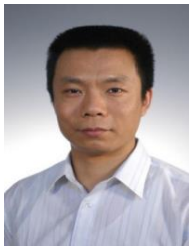
[2] Reed IS and Solomon G, "Polynomial codes over certain finite fields," *Journal of The Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp.300–304, 1960. Article (CrossRef Link).

[3] Blaum M, Brady J, Bruck J and Menon J, "Evenodd: an optimal scheme for tolerating double disk failures in raid architectures," *International Symposium on Computer Architecture*, pp. 245–254, 1994. Article (CrossRef Link).

[4] Corbett P, English B, Goel A, Grcanac T, Kleiman S, Leong J and Sankar S, "Row-diagonal parity for double disk failure correction," in *Proc. of Usenix Conference on File and Storage Technologies*, pp.1–1, 2004.Article (CrossRef Link).
[5] Xu L and Bruck J, "X-code: MDS array codes with optimal encoding," *Information Theory IEEE Transactions* on, vol. 45, no. 1, pp.272–276, 1999. Article (CrossRef Link).
[6] Li M, Shu J and Zheng W, "Grid codes: strip-based erasure codes with high fault tolerance for storage systems," *Acm Transactions on Storage*, vol. 4, no. 4, pp.1–22, 2009. Article (CrossRef Link).
[7] Sathiamoorthy M, Asteris M, Papailiopoulos D, Dimakis AG, Vadali R, Chen S and Borthakur D, "Xoring elephants: novel erasure codes for big data," *the VLDB Endowment*, vol. 6, no. 5, pp. 325-336, 2013. Article (CrossRef Link).
[8] Huang C, Simitci H, Xu Y, Ogus A, Calder B, Gopalan P, Li J and Yekhanin S, "Erasure coding in windows azure storage," in *Proc. of the 2012 USENIX Conference on Annual Technical Conference*, pp.2–2, 2012. Article (CrossRef Link).
[9] Dimakis AG, Godfrey PB,Wu Y,Wainwright MJ and Ramchandran K, "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 56, no. 9, pp.4539–4551, 2010. Article (CrossRef Link).
[10] Rashmi KV, Nakkiran P, Wang J, Shah NB and Ramchandran K, "Having your cake and eating it too: Jointly optimal erasure codes for I/O, storage and network-bandwidth," in *Proc. of the 13th USENIX Conference on File and Storage Technologies*, pp.81–94, 2015. Article (CrossRef Link).
[11] Shah NB, Rashmi KV, Kumar PV and Ramchandran K, "Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff," *IEEE Transactions on Information Theory*, vol. 58, no. 3, pp.1837–1852, 2012. Article (CrossRef Link).
[12] S. Goparaju, A. Fazeli and A. Vardy, "Minimum Storage Regenerating Codes for All Parameters," *IEEE Transactions on Information Theory*, vol. 63, no. 10, pp. 6318-6328, 2017. Article (CrossRef Link).
[13] B. Sasidharan, M. Vajha and P. V. Kumar, "An explicit, coupled-layer construction of a high-rate MSR code with low sub-packetization level, small field size and d < (n − 1)," in *Proc. of the 2017 IEEE International Symposium on Information Theory (ISIT)*, pp. 2048-2052, 2017. Article (CrossRef Link).
[14] Y. Hu, P. P. C. Lee and X. Zhang, "Double Regenerating Codes for hierarchical data centers," in *Proc. of the 2016 IEEE International Symposium on Information Theory (ISIT)*, pp. 245-249, 2016. Article (CrossRef Link).
[15] Z. Shen, P. P. C. Lee, J. Shu and W. Guo, "Cross-Rack-Aware Single Failure Recovery for Clustered File Systems," *IEEE Transactions on Dependable and Secure Computing*, pp. 1-1, 2017. Article (CrossRef Link).
[16] Y. Hu, Y. Xu, X. Wang, C. Zhan and P. Li, "Cooperative Recovery of Distributed Storage Systems from Multiple Losses with Network Coding," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 2, pp. 268-276, February 2010. Article (CrossRef Link).
[17] Shum, K.W. and J. Chen, "Cooperative Repair of Multiple Node Failures in Distributed Storage Systems," *International Journal of Information & Coding Theory*, **3**(4), 2016. Article (CrossRef Link).
[18] Schroeder B and Gibson GA, "Understanding disk failure rates: What does an MTTF of 1,000,000 hours mean to you," *ACM Transactions on Storage (TOS)*, vol. 3, no. 3, 2007. Article (CrossRef Link).
[19] Pinheiro E,Weber WD and Barroso LA, "Failure trends in a large disk drive population," in *Proc. of the 5th USENIX Conference on File and Storage Technologies*, pp.2-2, 2007. Article (CrossRef Link).
[20] Bairavasundaram LN, Goodson GR, Pasupathy S and Schindler J, "An analysis of latent sector errors in disk drives," in *Proc. of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp.289–300, 2007. Article (CrossRef Link).
[21] Schroeder B, Damouras S and Gill P, "Understanding latent sector errors and how to protect against them," *Acm Transactions on Storage*, vol. 6, no. 3, pp.1-23, 2010. Article (CrossRef Link).

[22] Plank JS and Blaum M, "Sector-disk (SD) erasure codes for mixed failure modes in raid systems," *Acm Transactions on Storage*, vol. 10, no. 1, pp.4, 2014. Article (CrossRef Link).
[23] Plank JS, Blaum M and Hafner JL, "SD codes: Erasure codes designed for how storage systems really fail," in *Proc. of the 11th USENIX Conference on File and Storage Technologies*, pp.95–104, 2013. Article (CrossRef Link).
[24] Blaum M, Hafner JL and Hetzler S, "Partial-mds codes and their application to raid type of architectures," *IEEE Transactions on Information Theory*, vol. 59, no.7, pp.4510–4519, 2012. Article (CrossRef Link).
[25] Li M and Lee PPC, "STAIR codes: a general family of erasure codes for tolerating device and sector failures," *Acm Transactions on Storage*, vol. 10, no. 4, pp.1–30, 2014. Article (CrossRef Link).
[26] Rashmi KV, Shah NB and Kumar PV, "Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp.5227–5239, 2010. Article (CrossRef Link).
[27] Shah NB, Rashmi KV, Kumar PV and Ramchandran K, "Interference alignment in regenerating codes for distributed storage: Necessity and code constructions," *IEEE Transactions on Information Theory*, vol. 58, no.4, pp.2134–2158, 2010. Article (CrossRef Link).

Jianchao Bian is a Ph.D. Candidate in the School of cyberspace security at Beijing University of Posts and Telecommunications. His main research interests include information security, coding theory, storage security and reliability, disaster backup and recovery.



Shoushan Luo is presently working as a Professor and Ph.D. Supervisor in the School of cyberspace security at Beijing University of Posts and Telecommunications. His main research interests include information security, cryptography, secure multi-party computation.

Wei Li is a Ph.D. Candidate in the School of cyberspace security at Beijing University of Posts and Telecommunications. His main research interests include information security, cryptography, network security.

Yaxing Zha received his Ph.D. degree from Beijing University of Posts and Telecommunications in 2018. His main research interests include information security, cryptography, storage security and reliability, disaster backup and recovery.

Yixian Yang is presently working as a Professor and Ph.D. Supervisor in the School of cyberspace security at Beijing University of Posts and Telecommunications. His main research interests include information security, cryptography, secure multi-party computation.