Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

SA-RSR: a read-optimal data recovery strategy for XOR-coded distributed storage systems*

Xingjun ZHANG‡[1], Ningjing LIANG[1], Yunfei LIU[1], Changjiang ZHANG[1], Yang LI[2]

[1]*School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China*

[2]*Beijing Electronic Engineering General Research Institute, Beijing 100854, China*

E-mail: xjzhang@xjtu.edu.cn; l_ningjing@stu.xjtu.edu.cn; liuyunfei@stu.xjtu.edu.cn; zcj9527@stu.xjtu.edu.cn

Received May 16, 2021; Revision accepted Aug. 16, 2021; Crosschecked Feb. 28, 2022

Abstract: To ensure the reliability and availability of data, redundancy strategies are always required for distributed storage systems. Erasure coding, one of the representative redundancy strategies, has the advantage of low storage overhead, which facilitates its employment in distributed storage systems. Among the various erasure coding schemes, XOR-based erasure codes are becoming popular due to their high computing speed. When a single-node failure occurs in such coding schemes, a process called data recovery takes place to retrieve the failed node's lost data from surviving nodes. However, data transmission during the data recovery process usually requires a considerable amount of time. Current research has focused mainly on reducing the amount of data needed for data recovery to reduce the time required for data transmission, but it has encountered problems such as significant complexity and local optima. In this paper, we propose a random search recovery algorithm, named SA-RSR, to speed up single-node failure recovery of XOR-based erasure codes. SA-RSR uses a simulated annealing technique to search for an optimal recovery solution that reads and transmits a minimum amount of data. In addition, this search process can be done in polynomial time. We evaluate SA-RSR with a variety of XOR-based erasure codes in simulations and in a real storage system, Ceph. Experimental results in Ceph show that SA-RSR reduces the amount of data required for recovery by up to 30.0% and improves the performance of data recovery by up to 20.36% compared to the conventional recovery method.

Key words: Distributed storage system; Data reliability and availability; XOR-based erasure codes; Single-node failure; Data recovery

https://doi.org/10.1631/FITEE.2100242 **CLC number:** TP391.4

1 Introduction

With the rapid development of cloud computing and big data in recent years, distributed storage systems, such as Google's GFS (Ghemawat et al., 2003), HDFS (Borthakur, 2007), Microsoft's WAS (Calder et al., 2011), open source Ceph (Weil et al., 2006a), and Swift (Arnold, 2014), have been extensively applied. Considering the low reliability of the cheap commercial equipment in these systems and that storage nodes are prone to failure, some redundancy strategies are indispensable. Replication is a traditional redundancy technology and has been widely exploited in various storage systems. Replication places multiple copies of data on different nodes (in this paper, "node" refers to an independent failure domain, which can be a disk or a storage node), which is beneficial for data recovery because only one replica needs to be read and transmitted when a single-node failure occurs. However, replication consumes a massive amount of extra storage space.

A redundancy strategy has been proposed based

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (No. 62172327)

ORCID: Xingjun ZHANG, https://orcid.org/0000-0003-1434-7016

on erasure coding to save storage resource. In general, erasure codes are represented by (k, m), denoting that the original file is divided into k equally-sized data chunks and that m parity chunks are calculated by these k data chunks. Compared to replication, erasure codes usually have lower storage overhead under the same fault tolerance conditions. For example, Reed-Solomon (RS) code (4, 2) (Reed and Solomon, 1960) and 3-replication can both tolerate any two-node failure, while the former consumes half the storage space of the latter. Therefore, enterprises are gradually transferring to the use of erasure codes to provide efficient and reliable data storage services. Maximum distance separable (MDS) codes are especially popular for use in real storage systems, because they optimally balance fault tolerance and storage overhead.

RS code, which tolerates multiple node failures, is a representative MDS code and has been deployed in various storage systems, e.g., Google's GFS (Ghemawat et al., 2003) and Facebook's HDFS storage clusters (Facebook, 2018). Furthermore, array codes (Blaum et al., 1996; Tamo et al., 2011, 2013; Gad et al., 2013; Hou et al., 2019a; Hou and Lee, 2020) have been studied extensively, where each chunk is a column of elements in a two-dimensional array. Binary MDS array codes with node elements in GF(2) are an important family of array codes. Sometimes these codes are also called XOR-based codes, because they perform only binary XOR operations during encoding and decoding procedures, which can be efficiently implemented in hardware and software. Therefore, XOR-based codes are more efficient than RS code in terms of computation complexity. More importantly, recovery can work at symbol granularity for XOR-based codes (array elements are also referred to as "symbols" for representation convenience). Cauchy RS (CRS) code (Roth and Lempel, 1989) has all the advantages of RS code and performs only XOR operations. Two-fault-tolerant erasure codes, known as RAID-6 codes, have received more attention in recent decades, such as EVENODD (Blaum et al., 1995), RDP (Corbett et al., 2004), Blaum_Roth (Blaum and Roth, 1993), Liberation code (Plank, 2008), Liber8Tion code (Plank, 2009), X-Code (Xu LH and Bruck, 1999), H-Code (Wu et al., 2011), P-Code (Jin et al., 2009), and HV Code (Shen and Shu, 2014). X-Code, H-Code, P-Code, and HV Code are vertical codes, which are seldom applied in real storage systems due to the complex rules for placement of parity chunks. Furthermore, triple-fault-tolerant XOR-based erasure codes, e.g., STAR (Huang and Xu, 2008), RTP (Goel and Corbett, 2012), and TIP (Zhang et al., 2015), are also presented to offer higher reliability.

MDS codes can greatly reduce the consumption of storage space; however, their recovery performance is far poorer than that of replication. For example, in a (k, m) MDS coded system, k chunks will be retrieved from surviving nodes to reconstruct any single failure chunk, while in a triple-replicated system, failure chunk can be recovered by downloading any single surviving replica. This k-factor data to be read and transmitted will result in a long recovery time, which may dramatically affect system service performance. Single-node failure in distributed storage systems accounts for more than 99.75% of all node failures (Schroeder and Gibson, 2007). Therefore, current research aims to optimize data recovery efficiency of a single-node failure.

Two optimization ideas are available to optimize data recovery of an erasure-coded storage system. The first idea is to design new erasure codes with a low recovery cost, which can theoretically reduce the amount of data required during recovery. Such codes include local reconstruction code (Huang et al., 2012) deployed by Microsoft in the WAS cloud storage system, Facebook's locally repairable code (Sathiamoorthy et al., 2013) applied in the HDFS-Xorbas system, and shingled erasure code (SHEC) (Miyamae et al., 2014; RedHat, 2018) exploited in the Ceph system. The improved recovery performance is obtained at the expense of extra storage space for these codes, which is not desired in real systems. In contrast, regenerating codes (RGCs) (Jiekak et al., 2013; Hou et al., 2019b, 2019c; Ye et al., 2020) are erasure codes that are developed from network coding, which can greatly reduce repair bandwidth without increasing storage overhead. However, they are rarely implemented in real systems because they lack several key desirable properties that are crucial for practical systems (Pamies-Juarez et al., 2016; Vajha et al., 2018).

Another idea is to optimize the repair processes of existing codes rather than trying to construct new codes. Some techniques that reduce the amount of data required have emerged for existing MDS codes, especially for XOR-based codes. For example, the

minimum amount of data needed for RDP, EVENODD, and X-Code in recovering a single-node failure has been derived (Wang et al., 2010; Xiang et al., 2011), but these conclusions and recovery methods cannot be extended to other erasure codes because they have inherently different structures. Some general optimization algorithms have been proposed to recover any MDS XOR-based code, but these approaches are highly complex or easily fall into a local optimum solution. In addition, none of these techniques have been tested on real-world storage systems. To overcome these shortcomings, we focus on researching the issue of optimal recovery for any MDS XOR-based erasure code, and implement the read-optimal recovery approach in a real-world storage system to demonstrate its efficiency.

2 Background and related works

2.1 Background: XOR-based erasure codes

We consider a distributed storage system consisting of n nodes, which are partitioned into k nodes that store data and m ($m=n-k$) nodes that store parity information. The data nodes are denoted as $\boldsymbol{D}_0, \boldsymbol{D}_1, \cdots, \boldsymbol{D}_{k-1}$, and the parity nodes are represented as $\boldsymbol{P}_0, \boldsymbol{P}_1, \cdots, \boldsymbol{P}_{m-1}$. A file written to the erasure-coded storage system is first divided into multiple data blocks. Each data block can be encoded simultaneously to obtain k data chunks and m parity chunks. The resultant set of n chunks is called a stripe. For XOR-based erasure codes, a chunk is composed of w symbols. The i^{th} symbol in the j^{th} data chunk of a stripe is labeled as d_{wj+i}, where w is the number of symbols in each chunk. Similarly, the i^{th} symbol in the j^{th} parity chunk is labeled as p_{wj+i}. The size of a symbol is typically multiples of the sector size (e.g., 4096 bytes) and depends on the specific storage system implementation. An XOR-based erasure code can also be parameterized as (k, m, w). In storage systems, the encoding and decoding processes of each stripe are independent of each other, so we just consider a single stripe.

Fig. 1 depicts an example stripe of one XOR-based erasure code. In Fig. 1, the stripe comprises $k=4$ data chunks and $m=2$ parity chunks. Each chunk consists of $w=3$ symbols. Symbols d_0–d_2 are data symbols in the 0^{th} data chunk belonging to data node $\boldsymbol{D}_0$, while p_0–p_2 are parity symbols in the 0^{th} parity chunk belonging to parity node $\boldsymbol{P}_0$. For the sake of discussion, the notations of XOR-based erasure codes are listed in Table 1.

An XOR-based erasure code can be represented by the product of a bit matrix and a vector. The bit matrix is called a generator matrix in coding theory and can be divided into two parts. The first part is a $kw \times kw$ identity matrix, which can be considered to be a $k \times k$ matrix, whose element is a $w \times w$ bit matrix. The second part, known as a coding distribution matrix (CDM), is composed of an $mw \times kw$ matrix and determines the generation of parities. Specifically, parity symbol p_i ($0 \le i < mw-1$) is calculated from the data symbols whose corresponding columns are not zero in the i^{th} row of the CDM. Therefore, the CDM is able to define a unique XOR-based erasure code.

Fig. 2 shows the CRS code encoding process with $k=4$, $m=2$, and $w=3$. In Fig. 2, the encoding equations representing the generation of all parity symbols are as follows:

$$\begin{cases} p_0 = d_0 \oplus d_3 \oplus d_6 \oplus d_9, \\ p_1 = d_1 \oplus d_4 \oplus d_7 \oplus d_{10}, \\ p_2 = d_2 \oplus d_5 \oplus d_8 \oplus d_{11}, \\ p_3 = d_0 \oplus d_5 \oplus d_6 \oplus d_7 \oplus d_{10}, \\ p_4 = d_1 \oplus d_3 \oplus d_5 \oplus d_8 \oplus d_{10} \oplus d_{11}, \\ p_5 = d_2 \oplus d_4 \oplus d_6 \oplus d_9 \oplus d_{11}. \end{cases} \tag{1}$$

We can observe that if there is only one symbol

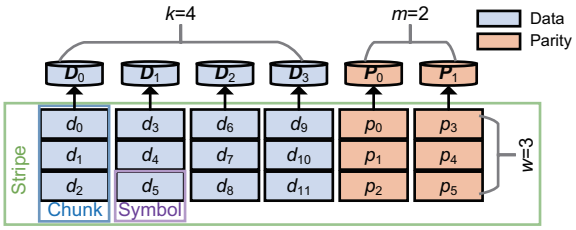

Fig. 1 A stripe of one XOR-based erasure code when $k=4$, $m=2$, and $w=3$

Table 1 Notations used in this paper

Notation	Description
k	Number of data chunks in each stripe
m	Number of parity chunks in each stripe
w	Number of symbols in each data/parity chunk
$\mathcal{F}$	Index of the failure data node
$\boldsymbol{D}_j$	The j^{th} data node in one stripe
d_{wj+i}	The i^{th} symbol in the j^{th} data chunk of a stripe
$\boldsymbol{P}_j$	The j^{th} parity node in one stripe
p_{wj+i}	The i^{th} symbol in the j^{th} parity chunk of a stripe
$\oplus$	XOR operation

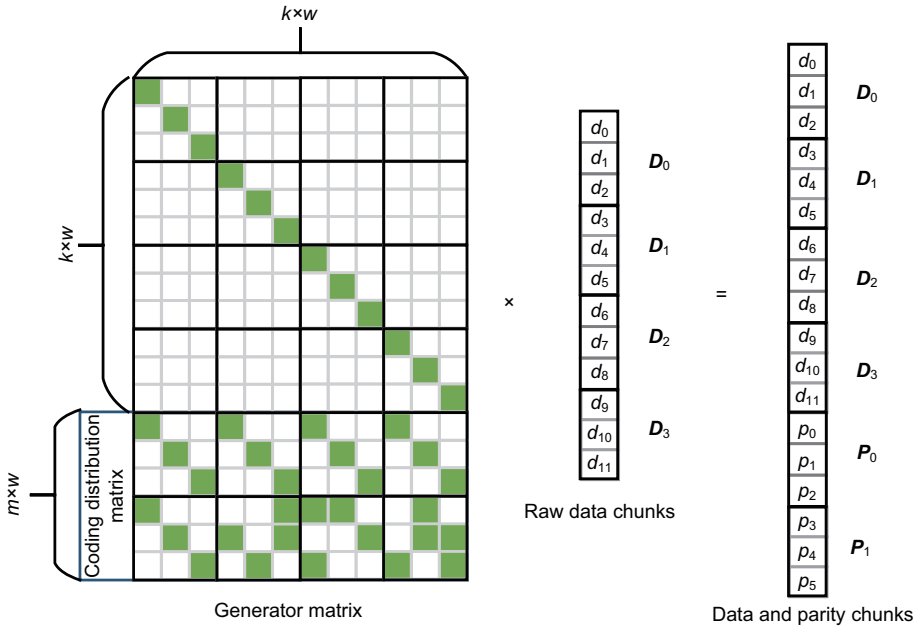

Fig. 2 Encoding process of CRS (4, 2, 3)

that experiences failure in one of the equations in Eq. (1), XOR of other surviving symbols can reconstruct this failure symbol, and more than one equation is available for recovering the same symbol (e.g., $d_0 = p_0 \oplus d_3 \oplus d_6 \oplus d_9$ and $d_0 = p_3 \oplus d_5 \oplus d_6 \oplus d_7 \oplus d_{10}$ when d_0 fails).

We focus on recovering failure symbols using the least amount of data instead of constructing a new erasure code. Our recovery approach can be applied to any XOR-based erasure code, and we conduct our experiments on three representative erasure codes, that is, CRS code, Blaum_Roth code, and Liber8Tion code. These codes have been implemented in the open-source library Jerasure-2.0, and readers can consult the manual (Pamies-Juarez et al., 2016) for details. CRS code is constructed based on a Cauchy matrix, has flexible configuration about (k, m, w), and has a dense generator matrix. Blaum_Roth code and Liber8Tion code ($w = 8$) are minimum density codes for RAID-6. Examples of CDMs for Blaum_Roth code and Liber8Tion code are shown in Fig. 3.

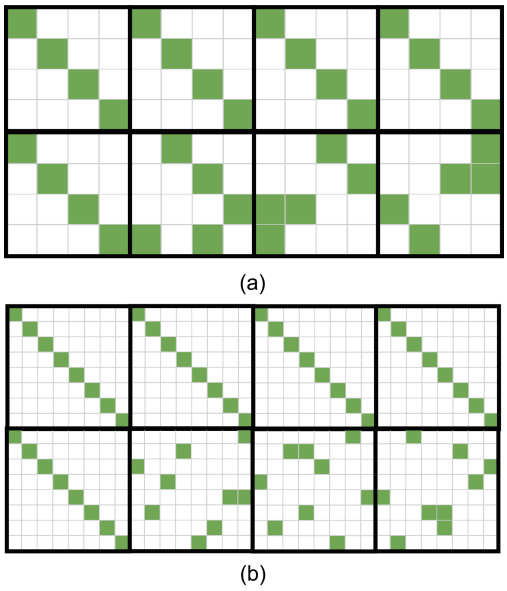

Fig. 3 CDMs for Blaum_Roth code (4, 2, 4) (a) and Liber8Tion code (4, 2, 8) (b) when $k = 4$ and $m = 2$

2.2 Related works

2.2.1 Conventional recovery

For storage systems that employ XOR-based erasure codes, conventional recovery methods will use the first surviving parity node to recover the failure data node. Assume that the $\mathcal{F}^{\text{th}}$ data node fails and that other nodes are all alive. The i^{th} failed symbol $d_{\mathcal{F}w+i}$ can be recovered by XOR-summing the i^{th} parity symbol of $\boldsymbol{P}_0$ and all the other surviving data symbols corresponding to the non-zero columns in the i^{th} row of the CDM. Therefore, recovering each erased data symbol requires the k symbols be read for any XOR-based erasure code.

If a parity node fails, it simply needs to perform a new encoding process to reconstruct the erased parity symbols. To the best of our knowledge, there is no research indicating that when a parity node fails, the number of symbols to be read can be further reduced. Therefore, existing optimization methods focus mainly on data node failures and this also applies in our approach.

2.2.2 Hybrid recovery

Xiang et al. (2011) first investigated the optimal recovery problem related to a single-node failure for RDP code (Corbett et al., 2004). They derived the lower bound of the number of symbols needed for any single failure recovery of the RDP code and proposed an efficient recovery method. Wang et al. (2010) conducted related and independent studies for EVENODD, X-Code, and STAR code. The authors demonstrated that the lower bound of the number of symbols of X-Code is $3p^2-2p+5$. Later, the tight lower bound of the number of symbols of X-Code was proved by Xu SL et al. (2014), and the load balancing recovery among different nodes was considered. Liang et al. (2020) extended this idea to liberation codes and evaluated their recovery scheme on a real distributed storage system. All the studies mentioned here follow the idea of hybrid recovery from Xiang et al. (2011), where symbols on different parity nodes are interchangeably employed to reconstruct the failed data symbols. Most of all, this is fundamental to other optimization approaches, e.g., the enumeration algorithm (Khan et al., 2012) and hill-climbing algorithm (Zhu et al., 2014).

Example 1 The CDM of RDP (4, 2, 4) is shown in Fig. 4. RDP code is also represented by a $(p-1)\times(p+1)$ two-dimensional array, where $w=p-1$, $n=p+1$, and p is a prime. A two-dimensional array of RDP (4, 2, 4) is plotted in Fig. 5. In Fig. 5, the parity symbols are generated by the symbols with the same color and shape. We take this RDP code as an example to illustrate how conventional recovery and hybrid recovery work. Fig. 6 shows the symbols used in these two approaches. In Fig. 6a, the erased symbols are labeled "×," and the surviving symbols to be read are marked with "◯." Conventional recovery uses parity node $\boldsymbol{P}_0$ to recover symbols on $\boldsymbol{D}_1$, so all surviving data symbols and parity symbols on $\boldsymbol{P}_0$ are read and the total number of symbols read is $wk=16$. In Fig. 6b, the

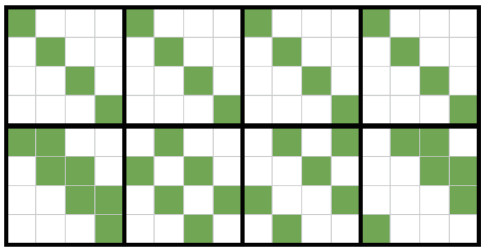

Fig. 4 CDM of RDP code when $k=4$, $m=2$, and $w=4$

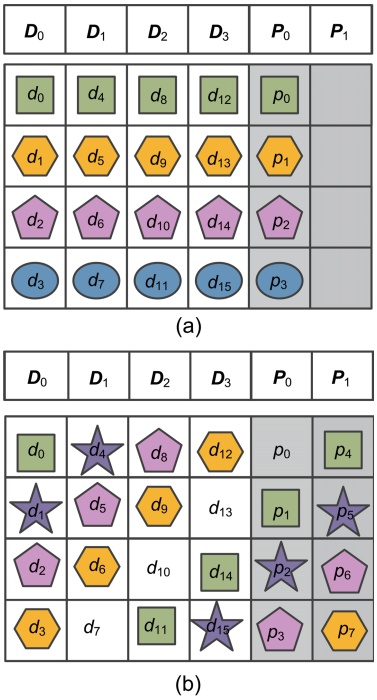

Fig. 5 Encoding of RDP code when $k=4$, $m=2$, and $w=4$: (a) $\boldsymbol{P}_0$ parity layout; (b) $\boldsymbol{P}_1$ parity layout

symbols read by parity nodes $\boldsymbol{P}_0$ and $\boldsymbol{P}_1$ are labeled "◯" and "□," respectively. The symbol that is read by more than one equation to recover failure symbols is referred to as an overlapping symbol. Overlapping symbols are marked using both "◯" and "□." One of the optimal recovery schemes to recover $\boldsymbol{D}_1$ for hybrid recovery is as follows:

$$\{d_1,d_{15},p_2,p_4\}\rightarrow d_4,$$
$$\{d_2,d_8,p_3,p_6\}\rightarrow d_5,$$
$$\{d_2,d_{10},d_{14},p_2\}\rightarrow d_6,$$
$$\{d_3,d_{11},d_{15},p_3\}\rightarrow d_7.$$

The number of symbols read is $wk-n_{\text{overlap}}=12$, where n_{overlap} is the number of overlapping symbols.

The core idea of hybrid recovery is the idea of "maximizing the number of overlapping symbols."

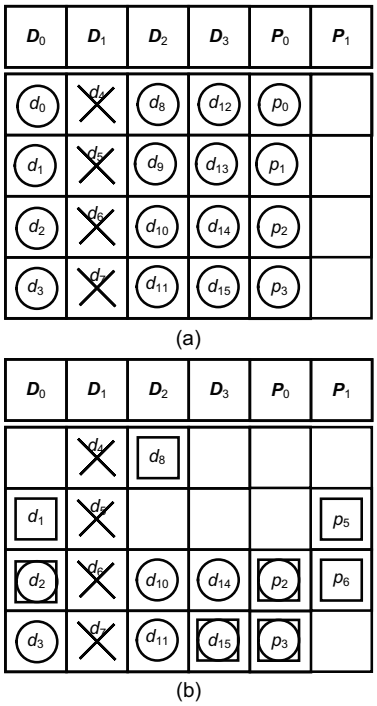

Fig. 6 The conventional (a) and hybrid (b) recoveries for RDP code when $k = 4$, $m = 2$, and $w = 4$

This approach has the advantages of finding the optimal amount of data read in advance and providing an optimal recovery scheme quickly. However, only a few XOR-based erasure codes with regular structures have been deduced about the lower bound of the number of symbols read, such as RDP, EVENODD, and X-Code. It is difficult to extend the optimal recovery conclusion of hybrid recovery to other codes, especially erasure codes with $m \geq 3$.

2.2.3 Enumeration recovery

To solve the optimal recovery problem of general XOR-based erasure codes, Khan et al. (2012) suggested modeling the problem of minimizing the number of symbols read as a combinatorial optimization problem and obtained the global optimized solution by enumeration.

For the sake of description, the notion of a recovery equation is defined as follows:

Definition 1 (Recovery equation) A set of symbols that are XOR-summed to a zero vector is referred to as a recovery equation.

According to the definition of the recovery equation, we can conclude that any single symbol can be reconstructed as long as the remaining symbols survive in a recovery equation.

Example 2 We use CRS $(4, 2, 3)$ in Fig. 2 to explain the recovery of the enumeration method. Let F contain all the failure symbols. Each symbol $f_i \in F$ owns a recovery equation set E_i. An equation e belongs to E_i only if $e \cap F = f_i$. This means that any recovery equation in E_i can recover f_i because its remaining symbols survive. For example, we assume that $\boldsymbol{D}_1$ fails, so $F = \{d_3, d_4, d_5\}$, and $E_1 = \{e_0, e_1\}$ is the recovery equation set of $f_1 = d_4$, where $e_0 = \{p_1, d_1, d_4, d_7, d_{10}\}$ and $e_1 = \{p_5, d_2, d_4, d_6, d_9, d_{11}\}$. Suppose that we can list all recovery equations for E_0–E_{w-1}. The problem is formulated as follows: select one equation from E_i to recover f_i and minimize the number of symbols in the union of the selected equations. In fact, the number of recovery equations is greater than mw, because the sum of two or even more recovery equations in Eq. (1) is still zero; i.e., $\{d_0, d_1, d_4, d_5, d_6, p_1, p_3\}$ is also a recovery equation obtained by adding the second and fourth equations in Eq. (1). In fact, there are up to $2^{mw} - 1$ candidate equations, so this approach is time-consuming, especially for large m or w.

2.2.4 Hill-climbing recovery

Zhu et al. (2014) employed a heuristic algorithm to tackle the problem mentioned. First the algorithm reduces the search space from $\binom{2^{mw-1}}{w}$ to $\binom{mw}{w}$ by using only mw recovery equations in the CDM. Then, it exploits a hill-climbing technique to search out a proximate optimum solution.

Example 3 Using CRS $(4, 2, 3)$ in Fig. 2 , we explain the principle of hill-climbing recovery next. Assume that $\boldsymbol{D}_1$ fails. Let S_i be the set of parity symbols from parity node $\boldsymbol{P}_i$, X be the set of w symbols used to recover failure symbols now, N_s be the number of symbols read now, and Y be the collection symbols to replace the elements in X. Initially, $S_0 = \{p_0, p_1, p_2\}$, $S_1 = \{p_3, p_4, p_5\}$, $X = S_0$, $N_\mathrm{s} = 12$, and $Y = S_1$. Then, it starts to update X. In every update process, it tries to use $y_j \in Y$ $(0 < j < 3)$ to replace one element $x_i \in X$ $(0 < i < 3)$; if the replacement is valid (can recover all failed symbols) and N_s is reduced, it will conduct the replacement, and vice versa. In the first update, supposing that we replace p_1 by p_5 and remove p_5 from Y, we obtain $X = \{p_0, p_5, p_2\}$ and $Y = \{p_3, p_4\}$. We have $N_\mathrm{s} = 10$ after this replacement. Next we continue the second update. We replace p_2 by p_3, while N_s cannot be reduced further, so

we stop the iteration process. Finally, the searched parity symbol set is $\{p_0, p_5, p_2\}$.

From the analysis above, we can determine that the complexity of hill-climbing recovery is greatly reduced compared to enumeration recovery. However, the algorithm omits a lot of important recovery equations and it easily falls into a local optimum solution.

3 Motivation

In general, enumeration recovery has an extremely high time complexity, because minimizing the number of symbols to be read for single data-node failure of XOR-based erasure codes is an NP-hard problem (Khan et al., 2012; Zhu et al., 2014). For an erasure code (k,m,w), the enumeration algorithm will select w from $2^{mw}-1$ recovery equations and guarantee that the total number of symbols required for recovery is minimum. Thus, the maximum search space of the enumeration algorithm can be $\binom{2^{mw}-1}{w}$. Therefore, it is not practicable to apply the enumeration algorithm to real-world storage systems.

Hill-climbing recovery can obtain a suboptimal symbol-reading scheme in polynomial time and the time complexity is $O(m\times w^3)$. However, the algorithm omits several important recovery equations that generate the optimal solutions and uses greedy thoughts in the running process, which has one major shortcoming: it easily falls into a local optimum solution.

In this study, we propose a random search recovery algorithm that is based on simulated annealing (SA). The algorithm has a low time complexity and overcomes the problem of easily being trapped in local optima in the search process.

4 Detailed design of SA-RSR

Based on this motivation, we propose a random search recovery algorithm named SA-RSR, which can be decomposed into two steps: parity symbol grouping and recovery solution search.

4.1 Parity symbol grouping

SA-RSR proposed in this study draws on the idea of grouping the recovery equations in the enumeration algorithm. SA-RSR significantly reduces time complexity, which is beneficial in facilitating its implementation in real storage systems.

It is easily observed that each recovery equation contains only a parity symbol from Eq. (1). Hence, a parity symbol can represent a recovery equation, and grouping parity symbols is equivalent to grouping the corresponding recovery equations. Parity symbol grouping partitions mw parity symbols into w groups, where parity symbols in one group participates in the recovery of the same failure data symbol. Details of parity symbol grouping are shown in Algorithm 1. In summary, the goal of SA-RSR is to select w symbols from mw parity symbols to recover w failed data symbols.

To facilitate the analysis, the recovery group is defined as follows:

Definition 2 (Recovery group) The collection of parity symbols whose corresponding recovery equations can be used to recover the same failure data symbols is known as a recovery group.

Next, we give a detailed description of the parity symbol grouping algorithm.

AddToG(g, d_j, p_i): Adding the parity symbol p_i to the recovery group $g_{j-\mathcal{F}w} \in G$ indicates that the recovery equation corresponding to the parity symbol p_i can be used to recover data symbol d_j.

We initialize G to NULL (step 1), traverse the rows of CDM $\boldsymbol{M}$ (step 2), and check whether the corresponding columns of the failure data symbols on the row is one (steps 3 and 4). If $[M]_{i,j}=1$, the recovery equation corresponding to the i^{th} parity symbol p_i can recover the j^{th} failure data symbol d_j and p_i is added to the recovery group $g_{j-\mathcal{F}w}$ (step 5).

Example 4 We take Fig. 2 as an example. When data node $\boldsymbol{D}_0$ fails, three data symbols will

Algorithm 1 Parity symbol grouping

Input:
$\mathcal{F}$: index of the failure data node;
$\boldsymbol{M}$: CDM of the erasure code;
m: number of the parity chunks in a strip;
w: number of the symbols in a data/parity chunk
Output:
G: a set of recovery groups for the failure data symbols

```
1: Initialize G = NULL
2: for i = 0 to mw − 1 do
3:     for j = Fw to Fw + w − 1 do
4:         if [M]_{i,j} == 1 then
5:             AddToG(g, d_j, p_i)
6:         end if
7:     end for
8: end for
```

be erased, namely, d_0, d_1, and d_2. For the row that corresponds to the parity symbol p_0, we note that $[M]_{0,0}=1$, which indicates that the recovery equation corresponding to p_0 can recover the failure data symbol d_0. Thus, we can add p_0 to g_0. Other parity symbols are also processed in this way. After parity symbol grouping, we can obtain the following results:

$$\begin{cases} g_0:\{p_0,p_3\}, \\ g_1:\{p_1,p_4\}, \\ g_2:\{p_2,p_5\}. \end{cases} \tag{2}$$

Furthermore, a parity symbol probably belongs to multiple groups simultaneously. For example, when data node $\boldsymbol{D}_1$ fails, data symbols d_3, d_4, and d_5 will be erased. p_4 will be simultaneously added to g_0 and g_2 due to $[M]_{4,3}=1$ and $[M]_{4,5}=1$. The final recovery groups are as follows:

$$\begin{cases} g_0:\{p_0,p_4\}, \\ g_1:\{p_1,p_5\}, \\ g_2:\{p_2,p_3,p_4\}. \end{cases} \tag{3}$$

If one parity symbol is selected per group and the same symbol is chosen more than once, some failure data symbols may not be recovered. For example, if p_4, p_1, and p_4 are chosen from g_0, g_1, and g_2 respectively, d_3 and d_5 will be unrecoverable because two equations in Eq. (1) are not enough to uniquely solve three unknown variables. SA-RSR addresses this scenario to ensure that one parity symbol appears at most once in the current search procedure.

4.2 Recovery solution search

After the grouping operation of Algorithm 1, we can obtain w recovery groups, and the corresponding recovery equations in each group are responsible for repairing a specific failure data symbol. For example, in Eq. (2), g_0 consists of p_0 and p_3 and their corresponding recovery equations can accomplish the recovery of data symbol d_0.

Definition 3 (Recovery solution) A set of w parity symbols that are selected from distinct recovery groups and the corresponding recovery equations that can repair all the failure symbols is known as a recovery solution G_{r}.

Example 5 According to Eq. (2), if we select p_0 from g_0, p_1 from g_1, and p_2 from g_2 to recover d_0, d_1, and d_2 respectively, we will obtain the recovery solution $G_{\mathrm{r}}=\{p_0,p_1,p_2\}$. G_{r} is obtained by selecting one element from g_i $(0\le i<w)$ to recover all failure symbols.

The recovery solution search problem can be considered as a combinatorial optimization problem. The set of feasible solutions is really massive if the erasure code parameters are a little larger. Let us use CRS (10, 4, 16) as an example, where CRS (10, 4, 16) has the same k and m parameter settings as the RS code used in HDFS-RAID in Facebook (Facebook, 2018). There are, on average, about 2.43×10^{20} alternative recovery solutions when one data node fails for CRS (10, 4, 16) and the number will increase to 2.53×10^{23} if m increases from 4 to 5. Therefore, it is difficult to obtain the best solution in such a huge research space by employing a brute-force method. We choose an SA-based technique, and SA is a stochastic optimization algorithm based on a Monte-Carlo iteration strategy. Compared to the hill-climbing technique, accepting the deteriorating solution with a certain probability and decreasing the probability over time can help SA step out of stagnation effectively.

Before explicitly presenting our detailed algorithm, we introduce the notion of a recovery sequence and provide a sketch of our algorithm. A recovery sequence consists of two parts, where the second part, having mw elements, indicates which symbol is selected out of mw parity symbols to recover the given failed symbol, and the first part, having kw elements, reveals the surviving data symbols that participate in the recovery of the failed symbol.

Definition 4 Define a recovery sequence as $R=\{r_0,r_1,\cdots,r_{(k+m)w-1}\}$. If parity symbol p_i is selected to recover a failed symbol, $r_{kw+i}=1$, and $r_j=1$ $(j\in\{e_0,e_1,\ldots,e_l\})$ for those surviving data symbols $\{d_{e_0},d_{e_1},\ldots,d_{e_l}\}$ that belong to the recovery equation corresponding to the parity symbol p_i; otherwise, $r_j=0$ $(j\notin\{e_0,e_1,\cdots,e_l\}\cup\{kw+i\})$.

Example 6 Taking Eq. (2) as an example, if p_3, p_1, and p_2 are chosen to recover d_0, d_1, and d_2 respectively, the recovery sequences are as follows:

$$\begin{cases} R_0=\{000001110010000100\}, \\ R_1=\{000010010010010000\}, \\ R_2=\{000001001001001000\}. \end{cases} \tag{4}$$

We use S to represent a symbol-reading scheme to recover all failure data symbols. S is calculated

using the following equation:

$$S = R_0|R_1|\cdots|R_{w-1}, \tag{5}$$

where "|" is the OR operation. If the j^{th} $(0 \le j < kw)$ position in the i^{th} $(0 \le i < w)$ recovery sequence R_i is 1, S_j will be set to 1. It indicates that we need to read data symbol d_j if S_j=1 $(0 \le j < kw)$ and read parity symbol p_{j-kw} if S_j=1 $(kw \le j < (k+m)w)$. Based on Eq. (5), our objective function is to minimize the number of 1's in S:

$$\min \sum_{i=0}^{(k+m)w-1} S_i. \tag{6}$$

Given the failure data symbols $\{d_{\mathcal{F}w}, d_{\mathcal{F}w+1}, \ldots, d_{\mathcal{F}w+w-1}\}$, we first obtain a set of recovery groups $G = \{g_0, g_1, \ldots, g_{w-1}\}$, and g_i is responsible for recovering $d_{\mathcal{F}w+i}$, e.g., $g_0 \to d_{\mathcal{F}w}$ and $g_1 \to d_{\mathcal{F}w+1}$. First, we initialize the current recovery solution G_{r} by appending a random symbol belonging to g_i. Then we update G_{r} through multiple iterations based on the idea of SA. Finally, we stop the algorithm until the convergence, obtain the recovery scheme S corresponding to G_{r}, and read the smallest number of symbols.

The recovery solution search algorithm (Algorithm 2) is built on the following three functions:

1. generate_solution(G_{r}): Given a recovery solution G_{r}, the function returns a symbol-reading scheme S for recovering the failure data symbols.

2. replace_symbol(G_{r}, $p_{\text{tmp\_sid}}$, $g_{\text{tmp\_gid}}$): Given the parity symbol $p_{\text{tmp\_sid}}$ and the recovery group $g_{\text{tmp\_gid}}$, the function replaces the symbol that belongs to $g_{\text{tmp\_gid}}$ with $p_{\text{tmp\_sid}}$ and returns a new recovery solution $G_{\text{r}_{\text{new}}}$.

3. cal_symbol_num(S): Given a symbol-reading scheme S, the function returns the total number of symbols to be read to recover the failure data symbols.

We initialize the temperature T as T_0 and current recovery solution G_{r} as NULL (step 1). For each failure data symbol, we randomly select a parity symbol from the corresponding recovery group $g \in G$ to construct G_{r} (steps 2–8). Note that the selection process requires multiple attempts to ensure that no parity symbol is repeated (steps 2–6). According to G_{r}, we can generate the symbol-reading scheme S (step 9). In the iteration of each outer loop, we generate a series of new symbol-reading schemes at the current temperature T and accept new schemes based on the idea of SA. At temperature T, we randomly select a symbol from all parity symbols, which is referred to as $p_{\text{tmp\_sid}}$ (step 12), and calculate the corresponding recovery groups g_{s} (step 13). If $p_{\text{tmp\_sid}}$ does not belong to G_{r}, we perform the following update operation. $p_{\text{tmp\_sid}}$ may be added

Algorithm 2 Recovery solution search

Input:
$\mathcal{F}$: index of the failure data node
$\boldsymbol{M}$: CDM of the erasure code
k: number of data chunks in a stripe
m: number of parity chunks in a stripe
w: number of symbols in a data/parity chunk
G: a set of recovery groups for the failure data symbols generated by Algorithm 1, and g_i is the i^{th} element of G
G_{r}: current recovery solution for the failure data symbols
T_0: initial temperature
T_{end}: termination temperature
K: temperature attenuation coefficient $(K < 1)$
N_{e}: number of external cycle iterations
N_{i}: number of internal cycle iterations
Output:
S: an optimal symbol-reading scheme for recovering failure data symbols

1: Initialize $T = T_0$ and G_{r} =NULL
2: **for** $i = 0$ to $w - 1$ **do**
3: x=one element selected randomly from g_i
4: **while** x is in G_{r} **do**
5: x=one element selected randomly from g_i
6: **end while**
7: G_{r}=G_{r}+x
8: **end for**
9: S = generate_solution(G_{r})
10: **for** $i = 0$ to N_{e} **do**
11: **for** $j = 0$ to N_{i} **do**
12: Select an element $p_{\text{tmp\_sid}}$ from mw parity symbols randomly
13: Calculate groups g_{s} to which $p_{\text{tmp\_sid}}$ belongs
14: **if** $p_{\text{tmp\_sid}} \notin G_{\text{r}}$ **then**
15: Select a group $g_{\text{tmp\_gid}} \in g_{\text{s}}$ randomly
16: $G_{\text{r}_{\text{new}}}$ = replace_symbol($G_{\text{r}}, p_{\text{tmp\_sid}}, g_{\text{tmp\_gid}}$)
17: S_{new} = generate_solution($G_{\text{r}_{\text{new}}}$)
18: Δ = cal_symbol_num(S) − cal_symbol_num(S_{new})
19: **if** $\Delta > 0$ or $\exp(\Delta/T) >$ rand()/rand_max **then**
20: $G_{\text{r}} = G_{\text{r}_{\text{new}}}$ and $S = S_{\text{new}}$
21: **end if**
22: **end if**
23: **end for**
24: $T = TK$
25: **if** $T < T_{\text{end}}$ **then**
26: break
27: **end if**
28: **end for**
29: **return** S

to multiple recovery groups, and thus we randomly select one recovery group $g_{\text{tmp\_gid}}$ from them (step 15). We can obtain a new recovery solution $G_{\text{r}_{\text{new}}}$ by replacing the parity symbol of $g_{\text{tmp\_gid}}$ with $p_{\text{tmp\_sid}}$ (step 16). The recovery scheme S_{new} corresponding to $G_{\text{r}_{\text{new}}}$ is generated (step 17). If S_{new} reads fewer symbols, we update S with S_{new}; otherwise, we accept the deteriorating solution S_{new} with a certain probability (steps 19–21). We will repeat this process N_{e} times at each temperature T. T will decrease according to a certain attenuation coefficient K until T is lower than the termination temperature T_{end} or the number of external iterations reaches the default value N_{e} (steps 24–28). Finally, it will return the scheme S whose number of symbols to be read is the smallest.

Example 7 We consider the recovery groups shown in Eq. (3) as an example. First, suppose the initial recovery solution $G_{\text{r}} = \{p_0, p_1, p_2\}$ by randomly selecting one parity symbol from each recovery group, and the corresponding symbol-reading scheme $S = \{1110001111111111000\}$. In addition, initialize temperature T, the numbers of internal iterations N_{i} and external iterations N_{e}, and the termination temperature T_{end}. Thus, the initial number of symbols to be read is 12. Then we perform the following procedure multiple times to update G_{r} and S at temperature T. Assume that we pick a random symbol $p_3 \notin G_{\text{r}}$ from $\{p_0, p_1, \ldots, p_5\}$. We replace $p_2 \in g_2$ with $p_3 \in g_2$ to generate $G_{\text{r}_{\text{new}}} = \{p_0, p_1, p_3\}$, which corresponds to the symbol-reading scheme $S_{\text{new}} = \{1100001101101101000\}$, and the number of symbols to be read is 9 ($9 < 12$). So, we need to update G_{r} and S using $G_{\text{r}_{\text{new}}}$ and S_{new} respectively. After N_{i} iterations at temperature T, we reduce T by multiplying by a factor K ($K < 1$) and continue performing the update procedure multiple times at each T until $T < T_{\text{end}}$ or N_{e} is reached.

4.3 Complexity analysis

We analyze the complexities of Algorithms 1 and 2 based on the notations listed in Table 1.

1. Complexity of Algorithm 1: Algorithm 1 is a traversal process on the CDM of erasure codes, and the complexity of Algorithm 1 is $O(m \times w \times w)$.

2. Complexity of Algorithm 2: During the execution of Algorithm 2, two loops are executed, while the number of external iterations equals N_{e} and the number of internal iterations equals N_{i}. However, temperature T will gradually decrease with the execution of the search process, and the number of external iterations may not reach N_{e}. Therefore, the complexity of Algorithm 2 will not exceed $O(N_{\text{e}} \times N_{\text{i}})$.

3. Complexity of SA-RSR: We have mentioned that SA-RSR can be decomposed into parity symbol grouping and recovery solution search. Thus, the complexity of SA-RSR is $O(m \times w \times w) + O(N_{\text{e}} \times N_{\text{i}})$.

5 Simulations

We conducted a series of intensive simulations to evaluate the performance of SA-RSR. Both the enumeration algorithm (Khan et al., 2012) and the hill-climbing algorithm Zpacr (Zhu et al., 2014) can reduce the amount of data required for recovery and accelerate the data recovery process for any XOR-based erasure code. We therefore chose these two algorithms as baselines to describe the advantage of SA-RSR. Our goal is to show that SA-RSR can return one symbol-reading scheme in a significantly short time, and that the amount of data required by SA-RSR for data recovery is minimum. Simulations were performed on a physical storage server with an Intel XeonE7 processor, 16 GB memory, and several SATA HDDs. Each disk was 600 GB and it operated at 7200 r/min. The operating system was Centos 7. For each erasure code parameter setting, we performed 300 simulation runs to obtain the average value.

We implemented the enumeration algorithm, Zpacr, and SA-RSR using the C++ language and considered three kinds of XOR-based erasure code as the simulation objective, including Blaum_Roth code (Blaum and Roth, 1993), Liber8Tion code (Plank, 2009), and CRS code (Roth and Lempel, 1989). Blaum_Roth code and Liber8Tion code belong to a class of minimum density RAID-6 codes (Plank et al., 2011), provide the optimal write performance, and can tolerate up to two node failures. All selected erasure codes were implemented in the Jerasure-2.0 Library, which is an open-source library and is commonly employed in the erasure code community (Plank et al., 2009). Table 2 lists the restrictions of these three coding schemes.

For the data recovery operation, three steps were performed: (1) determining the symbols involved in the data recovery process, (2) reading surviving symbols from the selected nodes, and (3)

Table 2 Erasure codes and their parameters we tested

Coding scheme	Restriction(s)
Blaum_Roth	$k \leq w$, m=2, $w > 2$, and w+1 is prime
Liber8Tion	$k < w$, m=2, and w=8
CRS	$2^w \geq k+m$

reconstructing the failure data symbols. In reality, SA-RSR primarily acts on step (1), which calculates the corresponding symbol-reading scheme while selecting nodes to perform the data recovery. Our justification is that the performance of data recovery in a distributed storage system primarily depends on network transmission instead of computations. Previous studies (Khan et al., 2012; Zhu et al., 2014) have validated that data reading and transmission parts consume the majority of data recovery time. Thus, we need to ensure that SA-RSR does not become a computational bottleneck in the data recovery operation.

5.1 Search performance

We compared the enumeration algorithm, Zpacr, and SA-RSR in terms of runtime to prove that SA-RSR can rapidly return an optimal symbol-reading scheme. Table 3 displays the runtime of different algorithms over various parameter settings. In the simulations, we randomly selected one data node and let it fail. When the parameters (k, m, w) were relatively large, the enumeration algorithm was too time-consuming to calculate an optimal symbol-reading scheme. From Table 3, we can see that the enumeration algorithm took more than 1 min to obtain the results under most of the parameter settings. Conversely, both SA-RSR and Zpacr can maintain a short computation time for all parameter settings, where they took less than 0.5 s. SA-RSR execution time was slightly longer than that of Zpacr.

5.2 Data required for data recovery

We simulated the amount of data required for data recovery to verify that SA-RSR can obtain a symbol-reading scheme with the least amount of data. The computation time of the enumeration algorithm is excessive, and we believe that accelerating the data recovery operation using the enumeration algorithm in real large-scale distributed storage systems is impractical. Therefore, we did not compare the enumeration algorithm with SA-RSR in terms of

Table 3 Runtime of enumeration, Zpacr, and SA-RSR recoveries

Coding scheme	(k, m, w)	Runtime (s)		
		Enumeration	Zpacr	SA-RSR
Blaum_Roth	(2, 2, 4)	94.80	0.0003	0.0005
Blaum_Roth	(3, 2, 4)	96.16	0.0003	0.0007
Liber8Tion	(2, 2, 8)	–	0.0179	0.0243
Liber8Tion	(4, 2, 8)	–	0.0166	0.0351
CRS	(2, 2, 4)	94.36	0.0003	0.0006
CRS	(3, 2, 4)	96.14	0.0002	0.0008
CRS	(3, 3, 3)	599.61	0.0004	0.0005
CRS	(4, 3, 8)	–	0.1656	0.1854
CRS	(4, 4, 8)	–	0.2761	0.3067
CRS	(5, 3, 8)	–	0.2038	0.2141
CRS	(5, 4, 8)	–	0.4498	0.4930

"–" means that the results cannot be calculated in a short period of time

the amount of data required for data recovery. After making a random data node fail, we executed Zpacr and SA-RAR algorithms, and compared the amount of data required by the symbol-reading schemes that they generated.

Fig. 7 illustrates the percentage of the required amount of data for recovery of Zpacr and SA-RSR over the conventional recovery method for various double-failure-tolerant coding schemes, including Blaum_Roth code, Liber8Tion code, and CRS code (m=2). For Blaum_Roth code, SA-RSR reduced the required amount of data by up to 25.0%, while Zpacr can reduce it by up to only 20.0%. For Liber8Tion code, SA-RSR reduced the required amount of data by up to 28.12%, but Zpacr reduced it by up to 25.0%. For CRS code, SA-RSR can reduce the required amount of data by up to 30.0% for recovery, yet Zpacr can reduce it by up to only 26.70%. Fig. 8 shows the results for three-failure-tolerant codes. SA-RSR reduced the required amount of data by up to 25.08%, while Zpacr reduced it by up to 22.22%.

Figs. 7 and 8 demonstrate that SA-RSR can reduce the required amount of data for data recovery for various kinds of codes compared to Zpacr. The reason is that the SA approach can prevent a search from being confined in suboptimal search regions compared with other local search methods, such as the hill-climbing algorithm.

5.3 Algorithm stability

Another crucial metric we considered is algorithm stability. Russell and Norvig (2016) noted

that SA algorithm is a probability-based algorithm, for which one may obtain different answers if one repeats the algorithm several times. We verified that the convergence of SA-RSR is satisfactory for the current algorithm implementation. We provide a new concept of accuracy:

$$\text{Accuracy} = \frac{\text{average number of symbols to be read}}{\text{minimum number of symbols to be read}}. \quad (7)$$

Accuracy is used to denote the closeness between the optimum solution and the average solution obtained by SA-RSR. The result is a decimal number equal to or greater than 1. The closer the accuracy to 1, the better the convergence of SA-RSR.

We evaluated SA-RSR's accuracy through experiments. During each experiment, we selected a data node randomly and made it fail. Table 4 shows the accuracies for various erasure code parameter settings, where $\mathrm{SN}_{\min}$, $\mathrm{SN}_{\max}$, and $\mathrm{SN}_{\mathrm{avg}}$ represent the minimum, maximum, and average numbers of symbols, respectively. We observed that the accuracies for Blaum_Roth code and Liber8Tion code can reach 1; the accuracy for CRS code did not exceed 1.04. The reason is that there are strict restrictions for Blaum_Roth code and Liber8Tion code. The parameters of Blaum_Roth code and Liber8Tion code are not large, which ensures that the total number of recovery equations is small. Thus, SA-RSR can obtain stable symbol-reading schemes in a very short time. We conclude that SA-RSR has good stability.

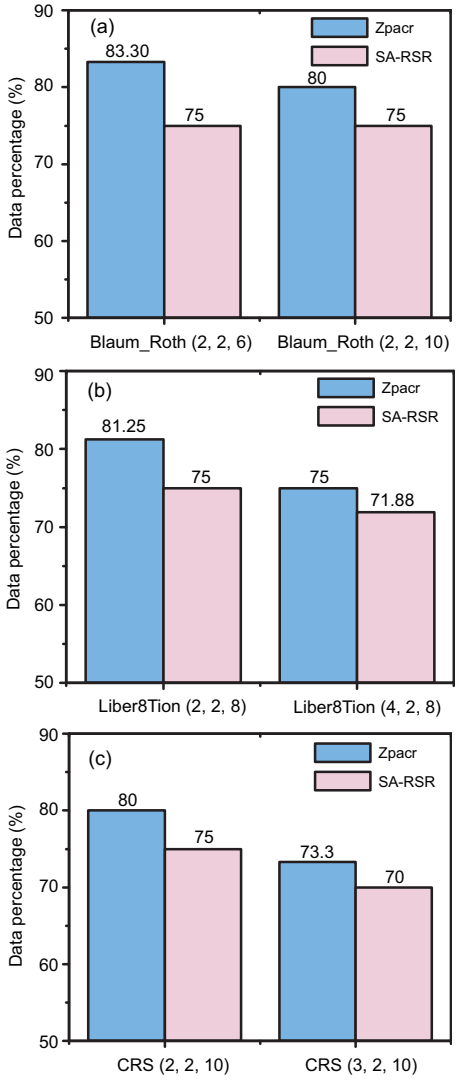

Fig. 7 Percentage of the data needed for Zpacr and SA-RSR over the conventional approach in terms of double-fault-tolerant codes: (a) Blaum_Roth code; (b) Liber8Tion code; (c) CRS code ($m = 2$)

6 Experiments

In Section 4, we described how SA-RSR reduces the amount of data required for data recovery. To

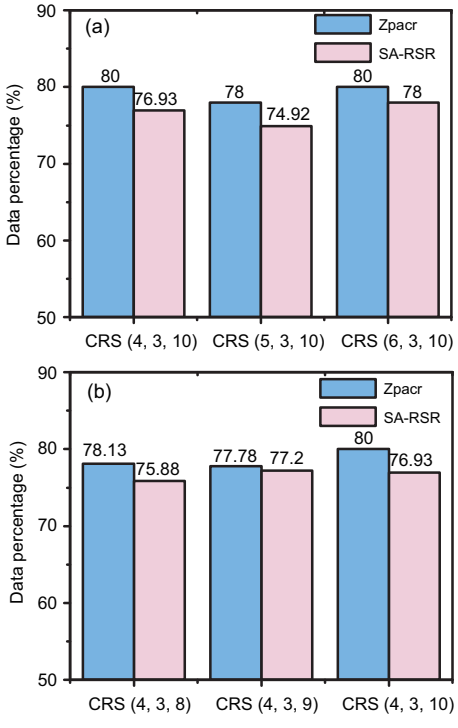

Fig. 8 Percentage of the data needed for Zpacr and SA-RSR over the conventional approach in terms of triple-fault-tolerant codes: (a) CRS code with varying k ($m = 3$, $w = 10$); (b) CRS code with varying w ($k = 4$, $m = 3$)

Table 4 Accuracy of SA-RSR with different coding schemes and different erasure code parameter settings

Coding scheme	(k, m, w)	SN_{min}	SN_{max}	SN_{avg}	Accuracy
Blaum_Roth	(2, 2, 6)	9	9	9	1
Blaum_Roth	(2, 2, 10)	15	15	15	1
Liber8Tion	(2, 2, 8)	12	12	12	1
Liber8Tion	(4, 2, 8)	23	23	23	1
CRS	(2, 2, 10)	15	16	15.01	1.0007
CRS	(3, 2, 10)	21	21	21	1
CRS	(4, 3, 7)	20	21	20.77	1.0385
CRS	(4, 3, 8)	24	25	24.28	1.0120
CRS	(5, 3, 4)	14	14	14	1
CRS	(4, 3, 10)	30	32	30.77	1.0260
CRS	(6, 3, 10)	46	48	46.907	1.0200

verify the effectiveness of SA-RSR in accelerating the data recovery process in real distributed storage systems, we implemented SA-RSR in the open-source distributed storage system, Ceph. Based on our implementation, we conducted substantial system experiments to test the recovery performance of SA-RSR in a distributed storage environment. In this section, we first give an overview of Ceph and its erasure coding scheme. We then present how to integrate SA-RSR into Ceph. Finally, we describe the SA-RSR experimental results.

6.1 Overview of Ceph

Ceph (Weil et al., 2006a) is an open-source distributed storage system with a decentralized system architecture and without a single point of failure. Ceph provides an infinitely scalable storage cluster that is based on a reliable, autonomic distributed object store (RADOS) (Weil et al., 2007) consisting of two types of daemons: the Ceph monitor (which is responsible for monitoring the entire storage cluster) and the Ceph object storage daemon (OSD) (which handles read/write operations on the storage disks). Generally, a single OSD is used to manage a single HDD or SSD. The Ceph clients and Ceph OSDs both use the CRUSH (Weil et al., 2006b) algorithm to efficiently determine an object's location information.

The Ceph storage system supports the notion of "pools," which are logical partitions for storing objects. Each storage pool has an access control and redundancy policy to provide a separate namespace for users and applications. Each storage pool has numerous placement groups (PGs) which are basic units of data storage and migration in the Ceph storage system. The settings of PGs depend on the number of replicas or the settings of the erasure code. PGs are distributed on multiple OSDs and can be dynamically adjusted according to the size of clusters and the number of objects. One primary OSD (p-OSD) must be selected from the placement group. The pool that is based on the replication scheme is responsible for forwarding objects to all other OSDs in the placement group, while the pool that uses erasure codes must divide and encode the object and send the coded block to other OSDs in the placement group. Ceph supports an erasure coding scheme via a pluggable interface that enables the use of a variety of traditional erasure codes (e.g., RS code and CRS code) and locally repairable codes.

The encoding process of the erasure code in Ceph is performed as a real-time job; i.e., the data is encoded while being written into the system. When writing data into a Ceph storage cluster using an erasure coded pool, the selected p-OSD divides the input data stream into many stripes. Each stripe is then divided into k data chunks and is encoded to generate $m = n-k$ parity chunks using (n, k, w) erasure codes. The stripe size in Ceph refers to the size of k data chunks and can be specified in the configuration file. Furthermore, a chunk is composed of w symbols, the size of which is a multiple of 8192 bytes.

Fig. 9 shows an example of the encoding process of Ceph, where the original object is divided into two stripes, each of which is composed of two chunks labeled as chunk 0 and chunk 1. Each chunk is further partitioned into two symbols, e.g., X_{00} and X_{01} in stripe 0 or Y_{00} and Y_{01} in stripe 1.

6.2 SA-RSR in Ceph

To integrate SA-RSR in the distributed storage system Ceph, we reconstructed the execution logic of the data recovery process of erasure codes in Ceph. Fig. 10 shows our implementation architecture. During data recovery, Ceph first selects one OSD node, usually the p-OSD node, in the PG to perform data recovery operations. We refer to this OSD as the recovery node.

Ceph, by default, uses the conventional recovery strategy mentioned in Section 2.2. In detail, according to the coding scheme in the PG and the number and location of the failed nodes, Ceph selects k surviving nodes to participate in the data recovery operation, including data nodes and parity nodes. In our design, when performing data recovery, the recovery scheduling module in Ceph first executes SA-RSR to

obtain an optimal symbol-reading scheme. Then according to the symbol-reading scheme, the recovery node constructs a reading request for each node involved in the data recovery process. Once the reading requests are built, they are sent to the corresponding nodes.

A recovery read handler runs on each surviving node and handles the read requests sent by the recovery node. OSDs in Ceph are responsible for reading data blocks from the disk. Based on this system design, the recovery read handler performs data read operations according to the messages in the read requests, which contain the identification numbers of the symbols that the current node needs to read. All symbols needed for data recovery are sent back to the recovery node.

The failure data symbols can be obtained by the decoding operation of the decoding module. Ceph uses an erasure code plug-in infrastructure that enables dynamic use of external erasure code libraries, while the XOR-based erasure codes are implemented in the Jerasure-2.0 Library in Ceph. Therefore, we improve the decoding part of the Jerasure library to ensure that it can decode the lost data with fewer data.

Our integration of SA-RSR is based on the release of Ceph's Luminous v12.0.2 . We consider CRS code, Blaum_Roth code, and Liber8Tion code as representative XOR-based erasure codes, which have been implemented in the Jerasure library in Ceph.

6.3 Results

In this subsection, we evaluated the recovery performance of SA-RSR in Ceph. Ceph was deployed on a cluster consisting of five Linux-based servers, each of which was equipped with an Intel XeonE7 CPU and 16 GB RAM. Each storage device was a SATA HDD with 600 GB capacity and it operated at 7200 r/min. The servers were interconnected via

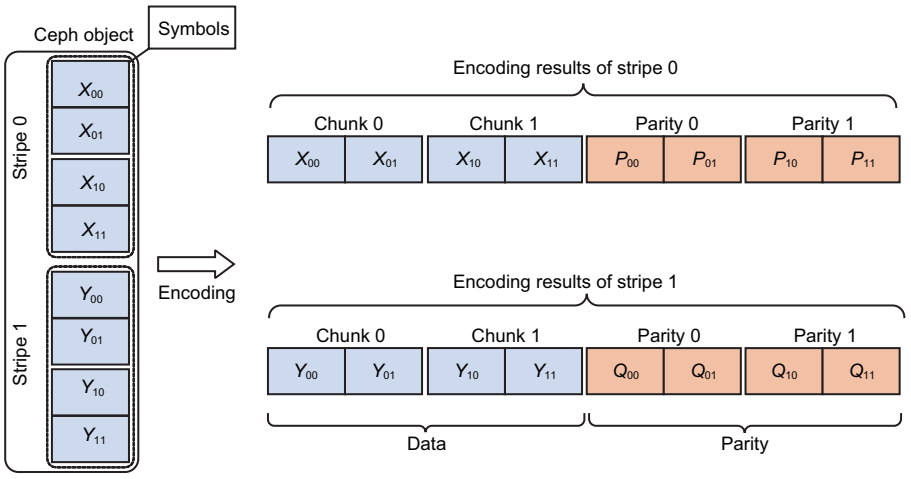

Fig. 9 A Ceph encoding example when k=2 and m=2 (The object is divided into two stripes, and the symbols within each of the stripe are encoded)

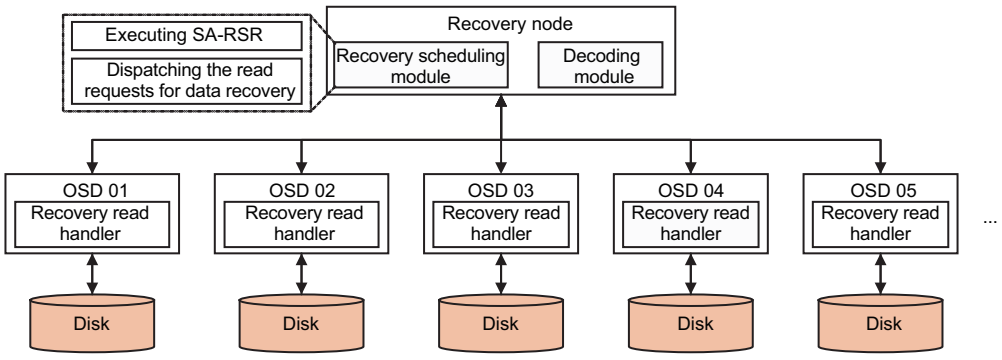

Fig. 10 Integration of SA-RSR in Ceph

a gigabit ethernet switch. In the Ceph system, a physical disk can serve as a logical OSD storage node. Here, each of the four servers was configured three disks to form three logical OSD nodes; the remaining server was responsible for managing the cluster and was used as a monitoring node. Thus, we had at most 12 OSD nodes with capacity of 7.03 TB depending on the chosen erasure coding schemes.

In Section 5, we explained the significant complexity of the enumeration algorithm and concluded that the enumeration algorithm is not always suitable for real distributed storage systems, so we implemented Zpacr and SA-RSR in Ceph without implementing the enumeration algorithm. We did not specifically optimize our implementation to demonstrate the absolute recovery performance. Instead, our goal is to evaluate the relative performance of SA-RSR compared with the conventional approach and Zpacr in fair conditions.

We focused on the metric of recovery speed, which is the amount of data recovered per second. We wrote a 200-MB object into a Ceph storage cluster using a specified coding scheme (e.g., CRS code, Blaum_Roth code, or Liber8Tion code). Note that SA-RSR may read non-contiguous symbols in a disk, which will result in random access to HDDs and performance degradation. However, Khan et al. (2012) pointed out that the enumeration algorithm with large stripes in cloud file systems can amortize the seek costs incurred when reading non-contiguous symbols. This also applies to our SA-RSR algorithm when implemented in a distributed storage system. Specifically, we set our symbol size to 1 MB to amortize the seek cost in our experiments. We made one node fail in the corresponding PG by setting the state of the OSD as "out" to trigger the data recovery operation of the Ceph system; the recovery operation includes performing SA-RSR, building a read request for data recovery operation, reading corresponding symbols from disks, and performing the decoding operation. The recovery operation was performed 20 times, and we obtained the total average.

Fig. 11 demonstrates the comparison of the recovery speed among the conventional approach, Zpacr, and SA-RSR for different double-failure-tolerant codes, including Blaum_Roth code, Liber8Tion code, and CRS code (m=2). For Blaum_Roth code, Liber8Tion code, and CRS code, the improvements of SA-RSR over the conventional approach for data recovery were up to 19.48%, 17.92%, and 20.36% respectively, while Zpacr had an acceleration rate of 16.88%, 13.87%, and 18.56% in recovery speed over the conventional approach, respectively. Fig. 12 shows the results of different coding schemes that tolerated three or more node failures. We observed that SA-RSR improved the recovery speed by 19.05%, 18.01%, and 15.63% with k=4, 5, and 6 respectively, while Zpacr improved the recovery speed by 14.88%, 13.04%, and 13.13%, respectively. SA-RSR achieved 17.03%, 18.18%, and 19.05% improvements in recovery speed over the conventional approach with w=8, 9, and 10, respectively, while Zpacr increased the recovery speed by 15.38%, 15.34%, and 14.88%, respectively.

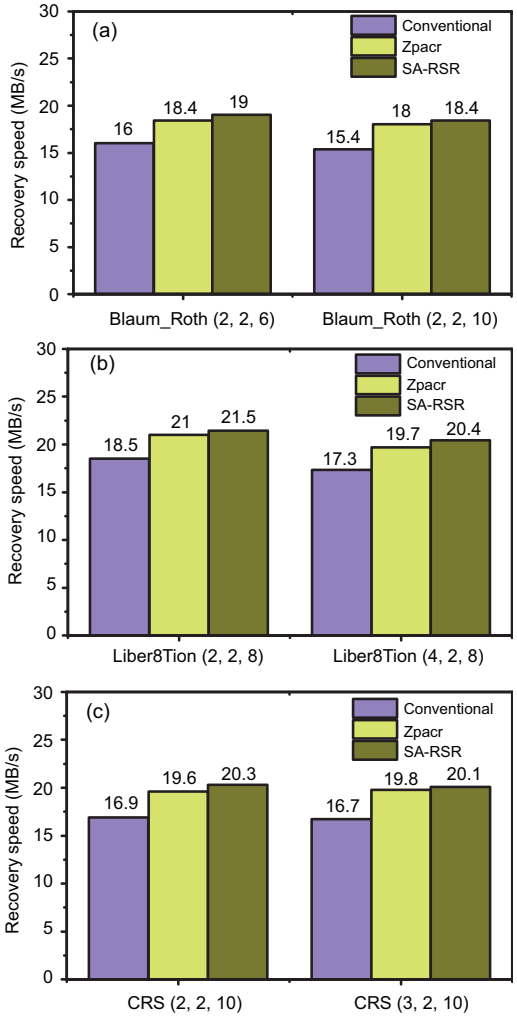

Fig. 11 Recovery speed comparison among the conventional approach, Zpacr, and SA-RSR for different codes that tolerate two node failures: (a) Blaum_Roth code; (b) Liber8Tion code; (c) CRS code ($m=2$)

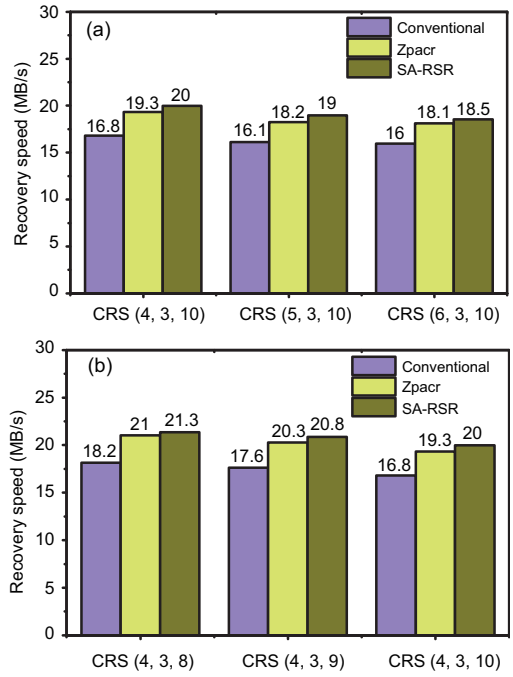

Fig. 12 Recovery speed comparison among the conventional approach, Zpacr, and SA-RSR for different codes that tolerate three or more node failures: (a) CRS code ($m = 3$, $w = 10$); (b) CRS code ($k = 4$, $m = 3$)

Figs. 11 and 12 indicate that SA-RSR can effectively accelerate the data recovery process of erasure coding schemes in the real distributed storage system. The reason is that SA-RSR effectively reduces the data amount in the data recovery process for various erasure coding schemes compared with the conventional approach, while Zpacr is prone to fall into a local optimum solution during execution. In Section 5, we declared that the recovery performance of the erasure coding schemes in distributed storage systems is determined mainly by network transmission rather than computing operation, and we verified that the computation overhead of SA-RSR for search is very small. Thus, the improvement of SA-RSR over Zpacr can be more significant.

7 Conclusions

Data recovery is a costly process in distributed storage systems that employ XOR-based erasure codes because it occupies a large amount of network bandwidth. To accelerate the recovery of failure data, we proposed a random search recovery algorithm that can provide the optimal recovery performance by minimizing the data needed for single-node failure recovery. This algorithm can significantly reduce the search time of an optimal recovery solution to a polynomial time. To demonstrate the effectiveness of our algorithm, we performed extensive simulations with a variety of XOR-based erasure codes. We also conducted experiments by implementing Zpacr and our algorithm in a real distributed storage system, Ceph. The results showed that our algorithm reduced the amount of data required for single-node failure recovery by up to 30.0% and improved the performance of data recovery by up to 20.36% in Ceph when compared to the conventional recovery method.

Contributors

Xingjun ZHANG designed the research. Ningjing LIANG and Yunfei LIU processed the data. Ningjing LIANG drafted the paper. Changjiang ZHANG helped organize the paper. Ningjing LIANG, Changjiang ZHANG, and Yang LI revised and finalized the paper.

Compliance with ethics guidelines

Xingjun ZHANG, Ningjing LIANG, Yunfei LIU, Changjiang ZHANG, and Yang LI declare that they have no conflict of interest.

References

Arnold J, 2014. OpenStack Swift Using, Administering, and Developing for Swift Object Storage. O'Reilly Media, Sebastopol, USA.

Blaum M, Roth RM, 1993. New array codes for multiple phased burst correction. *IEEE Trans Inform Theory*, 39(1):66-77. https://doi.org/10.1109/18.179343

Blaum M, Brady J, Bruck J, et al., 1995. EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Trans Comput*, 44(2):192-202. https://doi.org/10.1109/12.364531

Blaum M, Bruck J, Vardy A, 1996. MDS array codes with independent parity symbols. *IEEE Trans Inform Theory*, 42(2):529-542. https://doi.org/10.1109/18.485722

Borthakur D, 2007. The Hadoop Distributed File System: Architecture and Design. http://hadoop.apache.org/core/docs/current/hdfs_design.html

Calder B, Wang J, Ogus A, et al., 2011. Windows azure storage: a highly available cloud storage service with strong consistency. Proc 23rd ACM Symp on Operating Systems Principles, p.143-157. https://doi.org/10.1145/2043556.2043571

Corbett P, English B, Goel A, et al., 2004. Row-diagonal parity for double disk failure correction. Proc 3rd USENIX Conf on File and Storage Technologies, Article 1.

Facebook, 2018. HDFS-RAID. http://wiki.apache.org/hadoop/HDFS-RAID

Gad EE, Mateescu R, Blagojevic F, et al., 2013. Repair-optimal MDS array codes over GF(2). IEEE Int Symp

on Information Theory, p.887-891. https://doi.org/10.1109/ISIT.2013.6620354

Ghemawat S, Gobioff H, Leung ST, 2003. The Google file system. Proc 19th ACM Symp on Operating Systems Principles, p.29-43. https://doi.org/10.1145/945445.945450

Goel A, Corbett P, 2012. RAID triple parity. *ACM SIGOPS Oper Syst Rev*, 46(3):41-49. https://doi.org/10.1145/2421648.2421655

Hou HX, Lee PPC, 2020. Binary MDS array codes with optimal repair. *IEEE Trans Inform Theory*, 66(3):1405-1422. https://doi.org/10.1109/TIT.2019.2939111

Hou HX, Han YS, Lee PPC, et al., 2019a. A new design of binary MDS array codes with asymptotically weak-optimal repair. *IEEE Trans Inform Theory*, 65(11):7095-7113. https://doi.org/10.1109/TIT.2019.2923992

Hou HX, Han YS, Lee PPC, et al., 2019b. New regenerating codes over binary cyclic codes. IEEE Int Symp on Information Theory, p.216-220. https://doi.org/10.1109/ISIT.2019.8849354

Hou HX, Lee PPC, Shum KW, et al., 2019c. Rack-aware regenerating codes for data centers. *IEEE Trans Inform Theory*, 65(8):4730-4745. https://doi.org/10.1109/TIT.2019.2902835

Huang C, Xu LH, 2008. STAR: an efficient coding scheme for correcting triple storage node failures. *IEEE Trans Comput*, 57(7):889-901. https://doi.org/10.1109/TC.2007.70830

Huang C, Simitci H, Xu YK, et al., 2012. Erasure coding in windows azure storage. Proc USENIX Conf on Annual Technical Conf, Article 2.

Jiekak S, Kermarrec AM, Le Scouarnec N, et al., 2013. Regenerating codes: a system perspective. *ACM SIGOPS Oper Syst Rev*, 47(2):23-32. https://doi.org/10.1145/2506164.2506170

Jin C, Jiang H, Feng D, et al., 2009. P-Code: a new RAID-6 code with optimal properties. Proc 23rd Int Conf on Supercomputing, p.360-369. https://doi.org/10.1145/1542275.1542326

Khan O, Burns R, Plank J, et al., 2012. Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads. Proc 10th USENIX Conf on File and Storage Technologies, Article 20.

Liang NJ, Zhang XJ, Yang HL, et al., 2020. An optimal recovery approach for liberation codes in distributed storage systems. *IEEE Access*, 8:137631-137645. https://doi.org/10.1109/ACCESS.2020.3012190

Miyamae T, Nakao T, Shiozawa K, 2014. Erasure code with shingled local parity groups for efficient recovery from multiple disk failures. Proc 10th USENIX Conf on Hot Topics in System Dependability, Article 5.

Pamies-Juarez L, Blagojevic F, Mateescu R, et al., 2016. Opening the chrysalis: on the real repair performance of MSR codes. Proc 14th USENIX Conf on File and Storage Technologies, p.81-94.

Plank JS, 2008. The RAID-6 liberation codes. Proc 6th USENIX Conf on File and Storage Technologies, p.97-110.

Plank JS, 2009. The RAID-6 Liber8Tion code. *Int J High Perform Comput Appl*, 23(3):242-251. https://doi.org/10.1177/1094342009106191

Plank JS, Luo JQ, Schuman CD, et al., 2009. A performance evaluation and examination of open-source erasure coding libraries for storage. Proc 7th Conf on File and Storage Technologies, p.253-265.

Plank JS, Buchsbaum AL, Zanden BTV, 2011. Minimum density RAID-6 codes. *ACM Trans Stor*, 6(4):16. https://doi.org/10.1145/1970338.1970340

RedHat, 2018. Ceph Erasure. http://docs.ceph.com/docs/master/architecture/erasurecodin

Reed IS, Solomon G, 1960. Polynomial codes over certain finite fields. *J Soc Ind Appl Math*, 8(2):300-304. https://doi.org/10.1137/0108018

Roth RM, Lempel A, 1989. On MDS codes via Cauchy matrices. *IEEE Trans Inform Theory*, 35(6):1314-1319. https://doi.org/10.1109/18.45291

Russell SJ, Norvig P, 2016. Artificial Intelligence: a Modern Approach. Prentice-Hall, Inc., USA.

Sathiamoorthy M, Asteris M, Papailiopoulos D, et al., 2013. XORing elephants: novel erasure codes for big data. *Proc VLDB Endow*, 6(5):325-336. https://doi.org/10.14778/2535573.2488339

Schroeder B, Gibson GA, 2007. Disk failures in the real world: what does an MTTF of 1 000 000 hours mean to you? Proc 5th USENIX Conf on File and Storage Technologies, p.1-16.

Shen ZR, Shu JW, 2014. HV Code: an all-around MDS code to improve efficiency and reliability of RAID-6 systems. Proc 44th Annual IEEE/IFIP Int Conf on Dependable Systems and Networks, p.550-561. https://doi.org/10.1109/DSN.2014.57

Tamo I, Wang ZY, Bruck J, 2011. MDS array codes with optimal rebuilding. IEEE Int Symp on Information Theory, p.1240-1244. https://doi.org/10.1109/ISIT.2011.6033733

Tamo I, Wang ZY, Bruck J, 2013. Zigzag codes: MDS array codes with optimal rebuilding. *IEEE Trans Inform Theory*, 59(3):1597-1616. https://doi.org/10.1109/TIT.2012.2227110

Vajha M, Ramkumar V, Puranik B, et al., 2018. Clay codes: moulding MDS codes to yield an MSR code. Proc 16th USENIX Conf on File and Storage Technologies, p.139-153.

Wang ZY, Dimakis AG, Bruck J, 2010. Rebuilding for array codes in distributed storage systems. IEEE Globecom Workshops, p.1905-1909. https://doi.org/10.1109/GLOCOMW.2010.5700274

Weil SA, Brandt SA, Miller EL, et al., 2006a. Ceph: a scalable, high-performance distributed file system. Proc 7th Symp on Operating Systems Design and Implementation, p.307-320.

Weil SA, Brandt SA, Miller EL, et al., 2006b. CRUSH: controlled, scalable, decentralized placement of replicated data. Proc ACM/IEEE Conf on Supercomputing, Article 122-es. https://doi.org/10.1145/1188455.1188582

Weil SA, Leung AW, Brandt SA, et al., 2007. RADOS: a scalable, reliable storage service for petabyte-scale storage clusters. Proc 2nd Int Workshop on Petascale Data Storage: held in conjunction with Supercomputing, p.35-44. https://doi.org/10.1145/1374596.1374606

Wu CT, Wan SG, He XB, et al., 2011. H-Code: a hybrid MDS array code to optimize partial stripe writes in RAID-6. Proc IEEE Int Parallel & Distributed Processing Symp, p.782-793. https://doi.org/10.1109/IPDPS.2011.78

Xiang LP, Xu YL, Lui JCS, et al., 2011. A hybrid approach to failed disk recovery using RAID-6 codes: algorithms and performance evaluation. *ACM Trans Stor*, 7(3):11. https://doi.org/10.1145/2027066.2027071

Xu LH, Bruck J, 1999. X-code: MDS array codes with optimal encoding. *IEEE Trans Inform Theory*, 45(1):272-276. https://doi.org/10.1109/18.746809

Xu SL, Li RH, Lee PPC, et al., 2014. Single disk failure recovery for X-code-based parallel storage systems. *IEEE Trans Comput*, 63(4):995-1007. https://doi.org/10.1109/TC.2013.8

Ye FW, Liu SQ, Shum KW, et al., 2020. On secure exact-repair regenerating codes with a single Pareto optimal point. *IEEE Trans Inform Theory*, 66(1):176-201. https://doi.org/10.1109/TIT.2019.2942315

Zhang YZ, Wu CT, Li J, et al., 2015. TIP-Code: a three independent parity code to tolerate triple disk failures with optimal update complextiy. Proc 45$^{\text{th}}$ Annual IEEE/IFIP Int Conf on Dependable Systems and Networks, p.136-147. https://doi.org/10.1109/DSN.2015.19

Zhu YF, Lee PPC, Xu YL, et al., 2014. On the speedup of recovery in large-scale erasure-coded storage systems. *IEEE Trans Parall Distrib Syst*, 25(7):1830-1840. https://doi.org/10.1109/TPDS.2013.244