IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 10, 2020, accepted July 19, 2020, date of publication July 27, 2020, date of current version August 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3012190

An Optimal Recovery Approach for Liberation Codes in Distributed Storage Systems

NINGJING LIANG[1], XINGJUN ZHANG[1], (Associate Member, IEEE), HAILONG YANG[2], (Member, IEEE), XIAOSHE DONG[1], AND CHANGJIANG ZHANG[1]

[1]School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710048, China
[2]School of Computer Science and Engineering, Beihang University, Beijing 100191, China

Corresponding author: Xingjun Zhang (xjzhang@xjtu.edu.cn)

This work was supported by the National Key Research and Development Program of China under Grant 2016YFB1000303.

ABSTRACT To reduce the storage cost, distributed storage systems are gradually using erasure codes to ensure data reliability. Liberation codes, which satisfy the maximum distance separable (MDS) property and provide optimal modification overhead, are a class of popular two fault tolerant erasure codes. However, erasure codes need to read from surviving nodes and transfer across the network large amounts of data when recovering from single node failures. Existing single node failure recovery approaches for Liberation codes are either time-consuming or suboptimal. In this article, firstly, we prove the minimum number of symbols required to recover one failed node for a Liberation coded system. Then we derive the conditions that optimal recovery solutions need to satisfy. Finally, we propose an algorithm, called Disk Read Optimal Recovery (DROR), which can determine an optimal recovery solution in linear time and recover the failed node reading the minimum amount of data. We have implemented DROR in a real-world storage system Ceph and evaluated DROR on a cluster of Amazon EC2 instances. We show that DROR reduces the reconstruction time by up to 23.6% compared to that of the recovery approach in Ceph.

INDEX TERMS Liberation codes, minimum amount of data, optimal recovery approach, single node failures.

I. INTRODUCTION

Inexpensive components are preferred for use in modern distributed storage systems due to the economic benefits; however, these components are less reliable, and data may become temporarily or permanently unavailable. Therefore, a crucial requirement for building distributed storage systems is their reliability in the face of component failures. High reliability can be obtained by equipping the system with redundancy techniques, which are mainly classified into the following two categories: replication and erasure codes.

Replication has been widely used in modern storage systems, for example, in GFS [1] and Dynamo [2]. Replication generates several copies of the original data and dispatches each copy to a different node. *Node* in this article refers to an independent failure domain, which can be a disk or a storage node. The major drawback of replication is consuming massive amounts of extra storage space.

The associate editor coordinating the review of this manuscript and approving it for publication was Congduan Li.

Increasingly, storage systems [3]–[6], [7] are gradually beginning to use erasure codes to maintain high reliability. Compared to replication under the same fault-tolerance condition, erasure codes usually have lower storage cost. For example, (6,4) Reed-Solomon (RS) code [8] and 3-way replication can both tolerate any two node failure, while the former has half the storage cost of the latter. When using a (n, k) code, the original object is divided into k equal-sized data chunks, and m $(= n - k)$ parity chunks are calculated by these k data chunks. There is one kind of widely used erasure codes — maximum distance separable (MDS) codes, which can tolerant any m node failure under the minimum storage cost.

MDS codes can greatly reduce the consumption of storage space; however, their recovery performance is far lower than that of replication. Single node failures represent 99.75% of recoveries [9] and attract more attention recent years [10], [11]. In a (n, k) MDS coded system, k chunks will be retrieved from $(n - 1)$ surviving nodes to reconstruct any missing chunk. While, in a 3-way replicated system, the failed chunk

can be recovered by copying any one surviving replica. This k-factor increases both in disk I/O[1] and network traffic[2] result in a long recovery time, which may seriously affect the system service performance. In addition, too long reconstruction time will lead to an increase in the probability of data loss [12]. Therefore, improving the recovery performance is of much concern, especially the performance of single node recovery.

Unlike general erasure codes using expensive Galois Field arithmetic, XOR-based codes perform only effective XOR operations, which is desired for storage systems. Two fault tolerant erasure codes, known as RAID-6 codes, have received more attention in recent decades, since the probability of multiple node failures is higher than ever [9], [13]. Typical XOR-based RAID-6 codes include EVENODD [14], RDP [15], Blaum-Roth [16], Liberation code [17], Liber8tion code [18], CRS code [19], X-code [20], B-Code [21], H-code [22], P-code [23], HV-code [24], etc. X-code, B-Code, H-code, P-code and HV-code are vertical codes, which are seldom used in practical systems due to their complex placement rules for parity chunks. Liberation codes provide nearly optimal encoding and decoding performance, and most importantly, they reach the lower bound in terms of modification overhead among all horizontal RAID-6 codes. Therefore, Liberation codes have great potential in providing high endurance of parity disks for storage systems built with solid-state drives (SSDs).

Though RAID-6 codes were originally designed for RAID disk arrays, they have recently been applied successfully to distributed storage systems [25], [26] [27], [28] [29], [30]. For example, Fan *et al.* [25] propose DiskReduce, which integrates Blaum-Roth code into HDFS and can reduce storage overhead from 200% to 25% (when $n = 10, k = 8$) at a little expense of the performance of large reads. The consensus for storage systems is that two-failure tolerance is the right level of tolerance, assuming that data stripes are not large. Our work supports this trend, we are concerned with one kind of MDS RAID-6 codes — Liberation codes and investigate their recovery performance in distributed storage systems.

Note that, for practical distributed storage systems, it is significant to reduce repair time without sacrificing storage cost and data reliability; however, it is challenging to achieve this goal. Some techniques for existing MDS codes, especially for RAID-6 codes, have emerged to reduce the amount of data required during recovery. For example, the minimum amount of data read for RDP, EVENODD, and X-code when recovering single node failures are derived [10], [31], [32]. However, none of them has conducted experiments on real-world storage systems. Authors of [10] evaluate their recovery approaches for RDP using a disk simulator. Authors of [31] do not carry out experiments evaluating their efficient recovery approach for EVENODD. Authors of [32] conduct testbed experiments for the optimal recovery approach of X-code on their own small-scale experimental file system called NCFS, which can encode and decode the data on multiple storage devices according to a specified erasure code. Unfortunately, NCFS can be hardly used in an actual production environment due to its extremely simple functions. So making evaluations on a simulator or NCFS can not reflect the performance of various recovery approaches in real-world storage systems. In this work, we focus on theoretically researching the issue of optimal recovery for Liberation codes. In addition, we implement the read-optimal recovery approach in a real-world storage system and draw the conclusion that more than 20% recovery time can be reduced when the size of symbol is relatively large.

[1]disk I/O means the amount of data read from disks, and is also konwn as disk read or the number of symbols read.

[2]network traffic means the amount of data transferred across network.

TABLE 1. Major notations used in this article.

Notation	Description
n	The number of nodes in one storage system
k	The number of data nodes in one storage system
m	The number of parity nodes in one storage system
w	The number of symbols in one chunk
D_i	The i-th data node
C_i	The i-th parity node
P	The first parity node for RAID-6 coded storage system
Q	The second parity node for RAID-6 coded storage system
X_i	The i-th bit matrix for Q-matrix
I^w	$w \times w$ identity matrix
$I^w_{\rightarrow i}$	The matrix which is derived through rotating columns of I^w to the right by i columns
$< x >_w$	x mod w
$O^w_{y,y+i-1}$	The $w \times w$ matrix whose element in row $< y >_w$ and column $< y + i - 1 >_w$ is one and every other element equals zero
$d_{i,j}$	The i-th symbol in j-th column (or node)
p	The parameter of Liberation codes when $k = w = p$
H_i	The i-th horizontal parity set
A_j	The j-th anti-diagonal parity set
N_f	The failed node
$x_0 x_1 \cdots x_{p-1}$	The recovery sequence
Defined in **Algorithm 1**	
X	The read-optimal recovery sequence that we need to determine
I_X	The set of j whose corresponding value x_j is not yet determined $((x_j) \in X)$
$L1$	The set of j whose corresponding value $x_j = 1$ $((x_j) \in X)$
$L0$	The set of j whose corresponding value $x_j = 0$ $((x_j) \in X)$

II. BACKGROUND

To facilitate the discussion, we summarize the notations used in this article in Table 1.

We consider a storage system using a (n, k, w) XOR-based erasure code. The system consists of n nodes, k of which store the raw data and the remaining m $(= n-k)$ of which hold parity information. The data nodes are labeled as $D_0, \cdots, D_{k-1}$, and parity nodes are marked as $C_0, \cdots, C_{m-1}$. Each node is divided into multiple equal-sized chunks, and the chunk itself is composed of w symbols. The set of $k+m$ chunks, including k data chunks and the resultant m parity chunks, is called a *stripe*. Fig. 1 illustrates an example of stripes. In reality, a symbol is on the order of kilobytes or megabytes, which

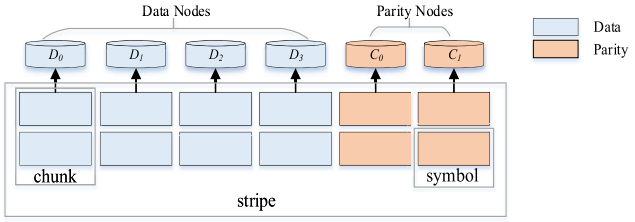


FIGURE 1. One stripe of erasure coded storage system when $k = 4, m = 2$, and $w = 2$.

depends on the specific storage system implementation. In the system, the stripes are encoded and decoded independently, so we only consider a single stripe.

A. MATRIX-VECTOR DEFINITION

XOR-based erasure codes can be expressed in terms of a bit matrix-vector product. A stripe of $(k+m)w$ bits are generated by multiplying the $(k+m)w \times kw$ bit matrix and a column vector composed of kw data bits. The bit matrix, called a binary distribution matrix (BDM), is made up of two parts. The first part contains a $kw \times kw$ identity matrix, which can be thought of as a $k \times k$ matrix, each of whose elements is a $w \times w$ bit matrix. The second part termed as the coding distribution matrix (CDM), comprises a $mw \times kw$ matrix. The CDM determines the generation of parity data. Therefore, we can use a CDM to specify one unique XOR-based code.

When the encoding methodology of XOR-based codes is applied to RAID-6 cods, $m = 2$, the two parity nodes C_0 and C_1 are called P and Q respectively, and the P node is computed as the parity of the data nodes. For simplicity of description, we propose the concepts of *P-matrix* and *Q-matrix* to represent the first w rows and the second w rows of CDM. For RAID-6 codes, the *P-matrix* is composed of k matrices, each of which is a $w \times w$ identity matrix; and the *Q-matrix* is composed of k bit matrices, which are labeled as $X_0, \cdots, X_{k-1}$. The *Q-matrices* are different for various kinds of RAID-6 codes.

Liberation codes are one of the lowest density RAID-6 codes. The value of w is a prime number greater than 2 and $w \geq k$. The definition of Liberation codes is as follows:

$$X_0 = I^w, \tag{1}$$

$$X_i = I^w_{\rightarrow i} + O^w_{y,y+i-1}, \tag{2}$$

where I^w is the $w \times w$ identity matrix, $I^w_{\rightarrow i}$ is derived through rotating columns of I^w to the right by i columns, $O^w_{y,y+i-1}$ is a $w \times w$ matrix whose element in row $< y >_w$ and column $< y + i - 1 >_w$ is one and every other element equals zero ($< x >_w = x \bmod w$), and y is

$$y = \frac{i(w-1)}{2}. \tag{3}$$

An alternate and equivalent specification of y is

$$y = \begin{cases} \dfrac{w-i}{2}, & i \text{ is odd} \\ w - \dfrac{i}{2}, & i \text{ is even.} \end{cases} \tag{4}$$

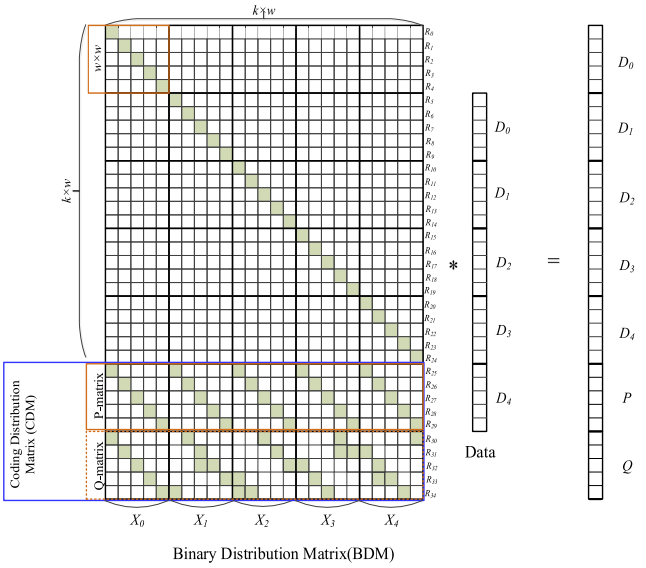


FIGURE 2. Encoding of the Liberation code when $k = w = 5$.

We present an example of Liberation codes in Fig. 2. In this figure, $k = 5$, $m = 2$,, $w = 5$, and R_i represents the row i of the BDM, which will be used in Section III-A2.

B. TWO-DIMENSIONAL ARRAY DESCRIPTION

For conveniently describing the recovery problem of single node failures, a $w \times (k+2)$ two-dimensional array experssion of Liberation codes is introduced. The first k columns in the array, corresponding to the k data nodes, store the data symbols. The last two columns hold the parity symbols, which comprise the contents of the P node and Q node, respectively. We call the first parity column *horizontal parity column*, and the second parity column *anti-diagonal parity column*. A symbol in the horizontal parity column is referred to as *horizontal parity symbol*, analogously, and a symbol in the anti-diagonal parity column is called *anti-diagonal parity symbol*.

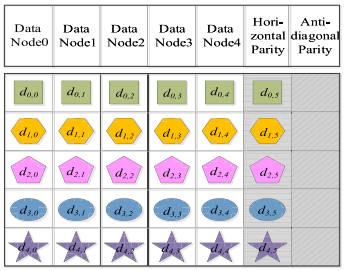
Assume $d_{i,j}$ $(0 \leq i \leq w-1, 0 \leq j \leq k+1)$ represents the i-th symbol in column j. The horizontal parity symbol $d_{i,k}$ and anti-diagonal parity symbol $d_{i,k+1}$ can be obtained as follows:

$$d_{i,k} = \bigoplus_{r=0}^{k-1} d_{i,r}, \tag{5}$$

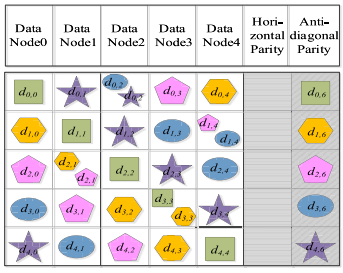
$$d_{i,k+1} = \begin{cases} \bigoplus\limits_{r=0}^{k-1} d_{<i+r>_w,r}, & i = 0 \\ \bigoplus\limits_{r=0}^{k-1} d_{<i+r>_w,r} \oplus d_{w-1-i,w-<2i>_w}, & \\ 0 < i \leq w-1. & \end{cases} \tag{6}$$

Note that, in Equation 6, $d_{w-1-i,w-<2i>_w} = 0$, if $w- < 2i>_w > (k-1)$.

Example 1: Fig. 3 shows an example of the two-dimensional array of one Liberation code. In this figure, each parity symbol is calculated by XOR-summing the data symbols with the same shape and color. Fig. 3a shows that, every horizontal parity symbol $d_{i,5}(0 \leq i \leq 4)$ is related to the data

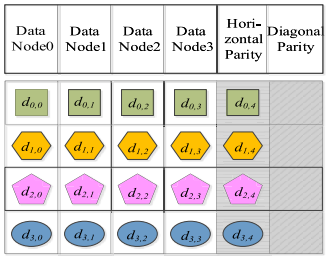


(a) Horizontal parity layout.

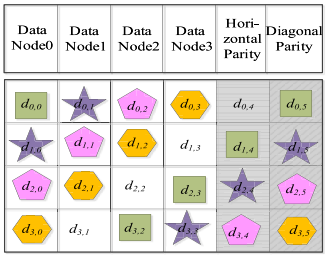


(b) Anti-diagonal parity layout.

FIGURE 3. Two-dimensional array form of the Liberation code when $k = w = 5$.



(a) Horizontal parity layout.



(b) Anti-diagonal parity layout.

FIGURE 4. Encoding of RDP code when $p = 5$.

symbols in the same row, e.g., $d_{0,5} = d_{0,0} \oplus d_{0,1} \oplus d_{0,2} \oplus d_{0,3} \oplus d_{0,4}$. We can see from Fig. 3b that, the first anti-diagonal parity symbol $d_{0,6}$ is associated with the data symbols along the same anti-diagonal; and each of the remaining anti-diagonal parity symbols $d_{1,6}, d_{2,6}, d_{3,6}$, and $d_{4,6}$ associates with an *extra symbol* $d_{3,3}, d_{2,1}, d_{1,4}$, and $d_{0,2}$ respectively, apart from the data symbols along the same anti-diagonal. For example, $d_{1,6} = d_{1,0} \oplus d_{2,1} \oplus d_{3,2} \oplus d_{4,3} \oplus d_{0,4} \oplus \boldsymbol{d_{3,3}}$.

In this article, we focus on the Liberation codes where k is equal to w. The Liberation codes mentioned in the rest of this article all meet the above conditions unless otherwise indicated. For the purpose of analysis, we introduce a new parameter p ($p = k = w$) to describe Liberation codes.

III. RELATED WORK AND MOTIVATION

A. SINGLE NODE FAILURE RECOVERY

In XOR-based erasure coded systems, if the failed node is a data node, the conventional method only uses the horizontal parity node to recover it. Each failed symbol can be recovered by XOR-summing the horizontal parity symbol and all the other surviving data symbols along the same row. This method is referred to as *Recovering From P Drive* (*RFPD*). It needs to read k symbols to recover each erased symbol for any XOR-based erasure code. If the failed node is a parity node, it simply needs to perform the corresponding encoding process to reconstruct the failed parity symbols.

There has been some work on reducing the amount of data needed to recover a failed node in a XOR-based erasure coded system, which can be mainly categorized as *Hybrid Recovery*, *Enumeration Recovery*, and *Hill-climbing Recovery*.

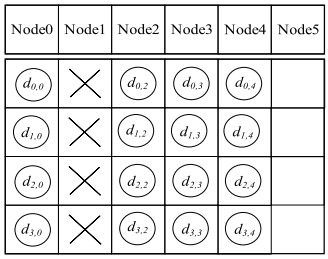
1) HYBRID RECOVERY

The authors of [10] first consider the problem of single node recovery for RDP. The lower bound of disk read is proven and an efficient recovery approach is proposed in [10]. Another related and independent research, which was published at nearly the same time, can be found in [31]; in this work, single disk recovery methods for EVENODD, X-code, and STAR code were studied. The authors show that the lower bound of X-code is $(3p^2 - 2p + 5)$. Xu *et al.* [32] prove the tight lower bound of X-code soon afterwards and consider the load balancing problem among different nodes. All the studies above follow the same idea of *hybrid recovery* for RDP [10], which makes use of the horizontal parity symbols and diagonal parity symbols interchangeably to reconstruct the failed symbols. RDP is defined with a $(p-1) \times (p+1)$ $(w = p-1, n = p+1)$ two-dimensional array, where p is a prime. Refer to [15] for more details. Fig. 4 shows an example of RDP code. Next, we explain the recovery idea using this RDP code.

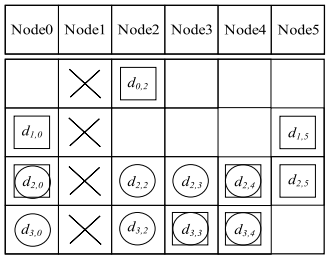
Example 2: Fig. 5 shows an example of *RFPD* and Hybrid Recovery for RDP. In this figure, $p = 5$ and *Node*1 happens failure. In Fig. 5a, we label the failed symbols with "×" and the surviving symbols read with " ◯ ". We can see that all the surviving data symbols from *Node*0, *Node*2, *Node*3 and the horizontal parity symbols from *Node*4 are required to be read. Hence, we need to read $(p-1)^2 = 16$ symbols in total. In Fig. 5b, the symbols read through the horizontal parity column and anti-diagonal parity column are labeled with " ◯ " and " □ " respectively. When recovering a failed symbol, we call the symbol that has been read *overlapping symbol*. Overlapping symbols are marked by using both " ◯ " and " □ ".

The symbol sets used to recover *Node*1 are as follows:

- $\{d_{1,0}, d_{3,3}, d_{2,4}, d_{1,5}\} \rightarrow d_{0,1}$
- $\{d_{2,0}, d_{0,2}, d_{3,4}, d_{2,5}\} \rightarrow d_{1,1}$
- $\{\boldsymbol{d_{2,0}}, d_{2,2}, d_{2,3}, \boldsymbol{d_{2,4}}\} \rightarrow d_{2,1}$
- $\{d_{3,0}, d_{3,2}, \boldsymbol{d_{3,3}}, \boldsymbol{d_{3,4}}\} \rightarrow d_{3,1}$.



(a) Conventional approach.



(b) Hybrid recovery approach.

FIGURE 5. Recovery approaches for RDP code when $p = 5$.

Therefore, the number of symbols read is $(p-1)^2 - n_{overlap} = 12$, where $n_{overlap}$ means the number of overlapping symbols.

From the above analysis, we can see that the essence of Hybrid Recovery is maximizing the number of overlapping symbols. The advantage of this method is that it can determine the optimal amount of data read before performing the reconstruction operation and offer an optimal recovery algorithm with low computational complexity.

In this article, we solve the optimal recovery problem of Liberation codes based on the idea of Hybrid Recovery since Hybrid Recovery owns above distinct advantages and Liberation and RDP codes share the similar generative rules of parities. However, it is difficult to apply the optimal recovery conclusion of Hybrid Recovery to other erasure codes. Firstly, considering the derivation of the lower bound of disk read, if t erasure symbols are recovered from diagonal parity sets and the remaining $(p-1-t)$ failed symbols are recovered from row parity sets, $n_{overlap} = t(p-1-t)$ [10]. The problem is converted to solving the maximum of a quadratic equation.

When it comes to Liberation codes, it is almost impossible to express $n_{overlap}$ with a formula since the intersections between different parity sets are complicated. We use a relaxation technique, which introduces a simplified form and converts the problem into solving the optimal solution of the simplified form. Secondly, it is challenging to find an optimal recovery solution that meets the lower bound of disk read for Liberation codes due to its more complicated generative rule of the Q node compared to RDP. We deduce several sufficient and necessary conditions that optimal recovery solutions need to satisfy and design an algorithm to determine one optimal recovery solution with low linear time complexity.

2) ENUMERATION RECOVERY

Through above analysis, we see that it is difficult to extend Hybrid Recovery to other erasure codes that tolerate two or more node failures. Hence, the authors of [11] model the recovery problem of single node failure for any XOR-based code as an combinatorial optimization problem, and they search for an optimal solution using an enumeration approach.

Example 3: We use the example of Liberation code in Fig. 2 to explain how it works. Assume D_1 happens failure, and each symbol is represented by $R_i (0 \le i < (k+m)w)$ corresponding to one row in the BDM. We call the set of symbols whose corresponding rows in the BDM sum to a vector of zeroes a *decoding equation*, which indicates that we can decode any one symbol so long as the remaining symbols are survived. An example is $e_0 = \{R_0, R_5, R_{10}, R_{15}, R_{20}, R_{25}\}$. The set F includes all the failed symbols, e.g., $F = \{R_5, R_6, R_7, R_8, R_9\}$. For every symbol $R_i \in F$, it has a decoding equation set E_i. An equation $e_i \in E_i$ only if $e_i \cap F = R_i$, for example, e_0 is one equation for R_5, since $e_0 \cap F = R_5$ and $R_0 \oplus R_5 \oplus R_{10} \oplus R_{15} \oplus R_{20} \oplus R_{25} = 0$. Suppose we can enumerate all decoding equations of the BDM to determine $E_5 - E_9$. Then the problem of optimal recovery for D_1 is formulated as follows: selecting one equation from each E_i for R_i $(5 \le i \le 9)$, such that the number of symbols in the union of all selected equations is minimized. Actually, there are up to $2^{mw} - 1$ equations to be searched for. Therefore, it is time-consuming to select w equations.

3) HILL-CLIMBING RECOVERY

To reduce the computation burden, the authors of [33] solve the above problem using a heuristic algorithm. The idea is based on the assumption that any optimal recovery solution $e_0, e_1, \cdots, e_{w-1}$ (selected from $E_0, E_1, \cdots, E_{w-1}$) has exactly w parity symbols. Thus, the search space is reduced from $\binom{2^{mw-1}}{w}$ to $\binom{mw}{w}$. Then, it uses a hill-climbing technique to search for an optimal recovery solution.

Example 4: To illustrate, we use the example of Liberation code in Fig. 2 and assume D_1 happens failure. Let $P_i (1 \le i \le m)$ be the set of parity symbols in the ith parity node, X be the set of w symbols considered now, N_s be the number of symbols read, and Y be the collection symbols to replace the elements in X. Initially, $P_1 = \{R_{25}, R_{26}, R_{27}, R_{28}, R_{29}\}$, $P_2 = \{R_{30}, R_{31}, R_{32}, R_{33}, R_{34}\}$, $X = P_1$, $N_s = 25$ and $Y = P_2$. Then, it starts to update X. In every update process, it tries to use $y_j (0 < j < 5) \in Y$ to replace one element $x_i (0 < i < 5) \in X$ each time; if the replacement is valid (can recover all failed symbols) and reduce the number of symbols read, it will perform a replace operation. In the first update, it can be verified that, y_0, y_1, y_2, y_3, y_4 can only replace $x_1, x_2, \{x_3, x_2\}, x_4, x_0$ respectively to ensure X valid. The replacements of y_0, y_1, and y_3 can reduce N_s to a smaller value 21. Suppose we replace $x_1 = R_{25}$ by $y_0 = R_{30}$ and remove R_{30} from Y. Having $X = \{R_{30}, R_{26}, R_{27}, R_{28}, R_{29}\}$ and $Y = \{R_{31}, R_{32}, R_{33}, R_{34}\}$, we contiune the second update. We replace $x_2 = R_{27}$ by $y_1 = R_{31}$, reduce N_s to 19 and remove R_{31} from Y. N_s can not be reduced further in the third update process, so we stop the iteration

process of update. Finally, we obtain the optimal solution $\{R_{30}, R_{26}, R_{31}, R_{28}, R_{29}\}$.

The complexity of Hill-climbing Recovery is $O(mw^3)$, which gains a great reduction compared to Enumeration Recovery. However, the algorithm is based upon greedy thoughts that cause the defect of easily falling into the local optimum solution. In addition, it needs to perform mw^3 validity verification and mw^3 calculations of N_s at most, which is time-consuming when the value of w is large.

B. MOTIVATION

In general, existing Hybrid Recovery methods can only be applied to RDP code or X-code. The Enumeration Recovery algorithm has an extremely high time complexity, for example, it needs to select w from $2^{mw} - 1$ ($m = n - k$) decoding equations to achieve optimal recovery for a (n, k, w) erasure code. Thus, it would not be feasible to apply the enumeration algorithm to real-world systems. The Hill-climbing algorithm can search for a recovery solution in polynomial time; here, the time complexity can be $O(mw^3)$. However, the algorithm uses greedy thoughts that the solution searched may not be necessarily global optimal and the time complexity is still a little high when the value of w is large. In this article, we propose a recovery algorithm based on an analysis of optimal recovery solutions of Liberation codes. The algorithm can ensure the total number of symbols required for data recovery is minimal and has low linear time complexity. The main contributions of this article are as follows:

1) We derive the lower bound of disk read in distributed storage systems using Liberation codes with $k = w$ when recovering from a single node failure.
2) We propose a recovery algorithm called Disk Read Optimal Recovery (DROR), which reaches the lower bound of disk read and decreases almost 25% of the disk read in theory compared with that of the conventional approach.
3) To evaluate the proposed recovery algorithm, we perform a great deal of experiments on Ceph, a popular storage system used in production. Our experimental results are consistent with our theoretical discovery. DROR reduces the recovery time by up to 23.6% compared with that of the conventional approach.

IV. THE READ-OPTIMAL RECOVERY METHOD

We first give a lower bound of symbols read through Theorem 1. We then point out the conditions that the optimal recovery sequences need to meet by Theorem 2. Finally, we propose an algorithm called DROR, which first determines an optimal recovery sequence quickly according to Theorem 2 and then recover all failed symbols using the sequence found.

A. THE LOWER BOUND OF SYMBOLS READ

In order to prove Theorem 1, we provide the following definitions and lemmas.

Definition 1:

1) Define the i-th horizontal parity set as $H_i = \{d_{i,r} | 0 \le r \le p\}, 0 \le i \le p-1$;
2) Define the j-th anti-diagonal parity set as $A_j = \{d_{i,r} | <j+r>_p = i, 0 \le r, i \le p-1\} \cup \{d_{j,p+1}\}, j = 0$ and $A_j = \{d_{i,r} | <j+r>_p = i, 0 \le i, r \le p-1\} \cup \{d_{p-1-j,p-<2j>_p}, d_{j,p+1}\}, 1 \le j \le p-1$.

Definition 2: Define a recovery combination as the set including H_i and A_j ($0 \le i, j \le p-1$) that can recover all failed symbols for the failed node $N_f (0 \le f \le p+1)$ and has length p.

Example 5: We use the Liberation code in Fig. 3 to illustrate Definition 1 and Definition 2. In Fig. 3, each horizontal parity set and anti-diagonal parity set is composed of symbols with the same shape and color. Each data symbol is contained in only one horizontal parity set and anti-diagonal parity set, except for $d_{3,3}, d_{2,1}, d_{1,4}$, and $d_{0,2}$, which are included within two anti-diagonal parity sets. For example, the 1-th horizontal parity set $H_1 = \{d_{1,0}, d_{1,1}, d_{1,2}, d_{1,3}, d_{1,4}, d_{1,5}\}$, and the 1-th anti-diagonal parity set is $A_1 = \{d_{1,0}, d_{2,1}, d_{3,2}, d_{3,3}, d_{4,3}, d_{0,4}, d_{1,6}\}$. $\{H_0, H_1, A_1, A_2, A_3\}$ is a recovery combination since it can reconstruct $\{d_{0,0}, d_{1,0}, d_{2,0}, d_{3,0}, d_{4,0}\}$ when N_0 is failed.

Note that, given a failed symbol d, d can only be reconstructed through the parity sets to which it belongs as long as the remaining symbols are surviving. We give the recovery principle of any failed symbol through Lemma 1.

Lemma 1: Given a failed node $N_f, 0 \le f \le p+1$,

1) If $0 \le f \le p-1$, it means that the failed node N_f is a data node.
   a) If $f = 0, d_{i,0} \in H_i$ and $d_{i,0} \in A_i$. Symbol $d_{i,0}$ can be recovered through either H_i or A_i.
   b) If $f \ne 0$
      - When $i \ne <f(p+1)/2 - 1>_p$ or $\ne <f(p+1)/2>_p$, $d_{i,f} \in H_i$ and $d_{i,f} \in A_{<i-f>_p}$. Symbol $d_{i,f}$ can be recovered through either H_i or $A_{<i-f>_p}$.
      - When $i = <f(p+1)/2 - 1>_p$ and $j = i+1$, $d_{i,f} \in H_i, d_{i,f} \in A_{<i-f>_p}$, and $d_{i,f} \in A_{p-1-i}$; $d_{j,f} \in H_j$ and $d_{j,f} \in A_{<j-f>_p}$. $d_{i,f}$ can be recovered through either H_i or $A_{<i-f>_p}$, and in the meanwhile $d_{j,f}$ can be recovered through H_j or $A_{<j-f>_p}$. In addition, $d_{i,f}$ can be recovered through A_{p-1-i} only if $d_{j,f}$ has been recovered through H_j.
2) If $f = p$, it means that the failed node is the P node, so $d_{i,p} \in H_i$, $d_{i,p}$ can only be recovered through H_i.
3) If $f = p+1$, it means that the failed node is the Q node, so $d_{i,p+1} \in A_i$, $d_{i,p+1}$ can only be recovered through A_i.

Example 6: We illustrate Lemma 1 using the Liberation code in Fig. 3. There are four cases as follows:

1) If N_0 happens failure, its symbol $d_{0,0}$ can be recovered through H_0 or A_0, since $d_{0,0} \in H_0, d_{0,0} \in A_0$, and other symbols in H_0 or A_0 are all surviving. Similarly, other symbols in N_0 can be recovered through the corresponding parity sets.

2) The recovery situation of the failure node from N_1 to N_4 is same, and here we take N_1 for example. When $f = 1$, $i \leq 0 \leq 4 (i \neq 2, i \neq 3)$, symbol $d_{i,1}$ can be recovered through H_i or $A_{<i-1>_p}$. When $i = 2, j = 3$, $d_{2,1}$ can be recovered through H_2 or $A_{<2-1>_5} = A_1$, at this moment, $d_{3,1}$ can be recovered through H_3 or $A_{<3-1>_5} = A_2$, since the missing symbol $d_{2,1}$ has been reconstructed and other symbols are all surviving. In addition, $d_{2,1}$ can be recovered through $A_{5-1-2} = A_2$, unless the missing symbol $d_{3,1}$ has been reconstructed through H_3.

3) If N_5 is failure, the symbol $d_{i,5} \in H_i$ and can only be recovered by the corresponding H_i.

4) If N_6 is failure, the symbol $d_{i,6} \in A_i$ and can only be recovered by the corresponding A_i.

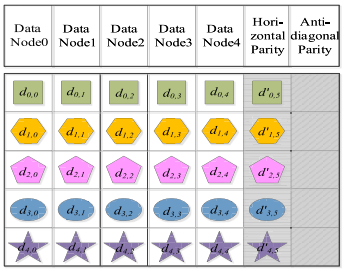
Lemma 2: The intersection between any two parity set is as follows:

1) For any $i, j, 0 \leq i, j \leq p-1, i \neq j, H_i \cap H_j = \emptyset$.
2) For any i, $1 \leq i \leq p-2$, if $j = i-1$, $A_i \cap A_j = d_{p-1-i,p-<2i>_p}$; if $j = i+1$, $A_i \cap A_j = d_{p-1-j,p-<2j>_p}$; otherwise, $A_i \cap A_j = \emptyset$.
3) For any $i, j, 1 \leq i \leq p-1, 0 \leq j \leq p-1$, if $j = p-i-1$, $A_i \cap H_j = \left\{d_{p-1-i,<p-1-2i>_p}, d_{p-1-i,p-<2i>_p}\right\}$, otherwise, $A_i \cap H_j = d_{j,<j-i>_p}$; for any j, $0 \leq j \leq p-1$, $A_0 \cap H_j = d_{j,j}$.

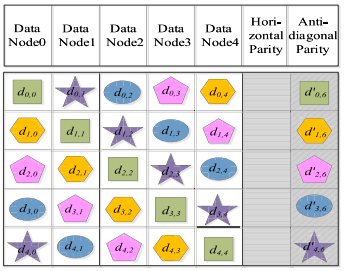
Example 7: We explain Lemma 2 with the Liberation code in Fig. 3. 1) of Lemma 3 means that there is no intersection between any two different horizontal parity sets, e.g., $H_0 \cap H_i \in \{H_1, H_2, H_3, H_4\} = \emptyset$. From 2) of Lemma 2, we can see that there is one overlapping element between A_i and A_{i-1}, and between A_i and A_{i+1}, if A_{i-1} or A_{i+1} exits. There is no overlapping element between A_i and other different anti-diagonal parity sets. For example, $A_1 \cap A_0 = d_{3,3}$, $A_1 \cap A_2 = d_{2,1}$, and $A_1 \cap A_j \in \{A_3, A_4\} = \emptyset$. 3) of Lemma 3 indicates the overlapping situations between A_i and H_j: when $i = 0$, the number of overlapping element between them is 1; when $i > 0, j \neq p-i-1$, the number also equals one; when $i > 0, j = p-i-1$, the overlapping number is two. For example, $A_0 \cap H_0 = d_{0,0}$, $A_1 \cap H_0 = d_{0,4}$, and $A_1 \cap H_3 = \left\{d_{3,2}, d_{3,3}\right\}$.

Theorem 1: For any failed node N_f, $(0 \leq f \leq p-1)$, a lower bound of symbols read for the recovery of a single node failure is $(3p^2+1)/4$.

Proof: We first introduce a *simplified form*, where $X_0 = I^p$, $X_r = I^p_{\rightarrow r}$, $0 \leq r \leq p-1$. An example of simplified form with $p = 5$ is shown in Fig. 6. In the simplified form, the horizontal parity symbols and anti-diagonal parity symbols are labeled as $d'_{j,p}$ and $d'_{j,p+1}$ $(0 \leq j \leq p-1)$, respectively. The rule of constructing the anti-diagonal parity symbols for the simplified form is not exactly the same as that of the Liberation codes. $d'_{j,p} = d_{j,p} (0 \leq j \leq p-1)$, $d'_{0,p+1} = d_{0,p+1}$, and $d'_{j,p+1} \neq d_{j,p+1} (0 < j \leq p-1)$. For any $0 \leq i, j, r \leq p-1$, $H'_i = \left\{d_{i,r}\right\} \cup \left\{d'_{i,p}\right\}$, and $A'_j = \left\{d_{i,r} \left| <j+r>_p = i \right.\right\} \cup \left\{d'_{j,p+1}\right\}$ represents the i-th horizontal parity set and j-th anti-diagonal parity set of the simplified form, respectively. It is easy to prove the following:



(a) Horizontal parity layout.



(b) Anti-diagonal parity layout.

FIGURE 6. The *simplified form* of Liberation code when $p = 5$.

(1) for any $i, j, 0 \leq i, j \leq p-1, i \neq j$, $\left|H'_i \cap H'_j\right| = 0$ and $\left|A'_i \cap A'_j\right| = 0$; (2) for any $i, j, 0 \leq i, j \leq p-1$, $\left|H'_i \cap A'_j\right| = \left|d_{i,<i-j>_p}\right| = 1$. Similarly, a recovery combination in the simplified form consists of H'_i and A'_j of length p.

Then, we provide a lower bound of disk read for Liberation codes considering the following three cases.

1) When $f = 0$, failed symbol $d_{i,0}$ $(0 \leq i \leq p-1)$ can be recovered through H_i or A_i. For any i, $0 \leq i \leq p-1$, A_i includes all the surviving data symbols belonging to A'_i, $A_i \supseteq A'_i$, and $H_i = H_i'$. For Liberation codes, the alternative recovery combinations correspond to those that own the same indices in the parity sets of the simplified form. Thus, the minimum number of symbols read in Liberation codes is no less than that in the simplified form, which is given as follows.

In the simplified form, if t failed symbols are recovered through anti-diagonal parity sets and the remaining $(p-t)$ symbols are recovered through horizontal parity sets, there are $t(p-t)$ overlapping symbols in total. The number of symbols read is as follows:

$$(p-t)p + tp - t(p-t) = t^2 - pt + p^2.$$

When $t = (p-1)/2$ or $t = (p+1)/2$, the minimum number of symbols read is $(3p^2+1)/4$.

Thus, when $f = 0$, a lower bound of disk read for Liberation codes is $(3p^2+1)/4$.

2) According to Lemma 3, when $f > 0$, for each value of f $(0 < f \leq p-1)$, there are two cases for failed symbols to be recovered and the smaller one is the lower bound that we want. In either case, the minimum number of symbols read is no less than that in the simplified form, which is analyzed as follows.

(1) Erased symbol $d_{i,f}$ $(0 < i \leq p-1)$ is recovered through H_i or $A_{<i-f>_p}$. It is easy to prove that the minimum number of symbols read in the simplified form is $(3p^2+1)/4$ using the method mentioned in 1), and the number of anti-diagonal parity sets required is $(p-1)/2$ or $(p+1)/2$.

(2) When $j =< f(p+1)/2-1>_p$, $d_{j,f}$ is recovered through A_{p-1-j} and $d_{<f-1-j>_p,f}$ is recovered through $H_{<f-1-j>_p}$. In the simplified form, if t anti-diagonal parity sets and $(p-t)$ horizontal parity sets are read, the total number of surviving symbols read is as follows:

$$(p-t)p + tp - t(p-t) - (p-1) + p = t^2 - pt + p^2 + 1.$$

When $t = (p-1)/2$ or $t = (p+1)/2$, the minimum number of symbols read is $(3p^2+5)/4$.

Integrate (1) and (2); when $f > 0$, a lower bound of disk read for Liberation codes is $(3p^2+1)/4$, and only when each failed symbol $d_{i,f}$ is recovered through H_i or $A_{<i-f>_p}$ may this lower bound be matched.

Thus, Theorem 1 concludes. ■

B. READ-OPTIMAL RECOVERY SEQUENCES

In the following, we introduce a notion *recovery sequence* to state how to reach the lower bound of symbols read derived in last subsection.

Definition 3: Define a recovery sequence as $x_0x_1\cdots x_{p-1}$, where each failed symbol $d_{i,f}$ is recovered through horizontal parity set H_i or through anti-diagonal parity set $A_{<i-f>_p}$. $x_i = 0$ means that failed symbol $d_{i,f}$ is recovered through horizontal parity set H_i and $x_i = 1$ means that failed symbol $d_{i,f}$ is recovered through anti-diagonal parity set $A_{<i-f>_p}$.

Note that there are total 2^p recovery sequences, which include the read-optimal recovery combinations according to Theorem 1. For example, when $N_f = N_1$ is failed and $p = 5$, $x_0x_1x_2x_3x_4 = \{00011\}$ means $d_{0,1}$, $d_{1,1}$, and $d_{2,1}$ are recovered using H_0, H_1 and H_2, respectively; $d_{3,1}$ and $d_{4,1}$ are recovered using A_2 and A_3, respectively.

Theorem 2: $\{x_i\}_{0\leq i\leq p-1}$ is a recovery sequence of Liberation codes that matches the lower bound of disk read for any failed node N_f $(0 \leq f \leq p-1)$ if the following two conditions hold:

1) $\sum_{i=0}^{p-1} x_i = (p-1)/2$ or $\sum_{i=0}^{p-1} x_i = (p+1)/2$
2) For any j, $0 \leq j \leq p-1$, $j \neq f$
   a) When $f = 0$, if $x_j = 1$, $x_{<j-1>_p}$ must be one or $x_{p-1-<j-f>_p}$ must be zero;
   b) When $f > 0$, $<j-f>_p \neq < f(p-1)/2 >_p$ (that is, $j \neq < f(p+1)/2 >_p$), if $x_j = 1$, $x_{<j-1>_p}$ must be one or $x_{p-1-<j-f>_p}$ must be zero.

Proof: Condition 1) means that $(p-1)/2$ or $(p+1)/2$ failed symbols are recovered through anti-diagonal parity sets, and the remaining failed symbols are recovered through horizontal parity sets. In the process of proving Theorem 1, we introduced a simplified form. Only when the number of anti-diagonal parity sets used is $(p-1)/2$ or $(p+1)/2$ can the lower bound of disk read for the simplified form be matched, so can the lower bound for Liberation codes be matched. Therefore, condition 1) holds.

In the following proof, we focus on condition 2). We set $z =< j-f>_p$, $0 < z \leq p-1$. $x_j = 1$ means $d_{j,f}$ is recovered through $A_{<j-f>_p} = A_z$, $x_{<j-1>_p} = 1$ means $d_{<j-1>_p,f}$ is recovered through $A_{<<j-1>_p-f>_p} = A_{z-1}$, and $x_{p-1-<j-f>_p} = 0$ means $d_{p-1-<j-f>_p,f}$ is recovered through H_{p-1-z}. In other words, when $f = 0$, if we can prove that for any z, $0 < z \leq p-1$, if A_z is used to recover a failed symbol, A_{z-1} or H_{p-1-z} must be used to recover another failed symbol, a) of condition 2) holds. Similarly, when $f > 0$, if we can prove that for any z, $0 < z \leq p-1$ and $z \neq < f(p-1)/2 >_p$, if A_z is used to recover a failed symbol, A_{z-1} or H_{p-1-z} must be used to recover another failed symbol, b) of condition 2) holds.

a) When condition 1) holds, the lower bound of disk read for the simplified form can be matched. A read-optimal recovery combination of the simplified form is labeled as

$$Rc' = \left\{H'_{x_1}, H'_{x_2}, \cdots, H'_{x_{p-t}} A'_{y_1}, A'_{y_2} \cdots, A'_{y_t}\right\}$$

and the corresponding recovery combination of Liberation codes is

$$Rc = \left\{H_{x_1}, H_{x_2}, \cdots, H_{x_{p-t}} A_{y_1}, A_{y_2}, \cdots, A_{y_t}\right\},$$

where $t = (p-1)/2$ or $t = (p+1)/2$, $0 \leq x_i, y_j \leq p-1$, $1 \leq i \leq p-t$, $1 \leq j \leq t$. The relationship between H_i (A_j) and H'_i (A'_j) is as follows:

$H_i = H_i'$

$$A_j = \begin{cases} A_j' - \left\{d'_{j,p+1}\right\} + \left\{d_{j,p+1}\right\}, j = 0 \\ A_j' - \left\{d'_{j,p+1}\right\} + \left\{d_{j,p+1}\right\} + \left\{d_{p-1-j,p-<2j>_p}\right\}, \\ j \neq 0. \end{cases}$$

When $f = 0$, for any j $(0 < j \leq p-1)$, A_j will introduce an extra surviving symbol $d_{p-1-j,p-<2j>_p}$ compared with A'_j. There are t anti-diagonal parity sets $A_{y_1}, A_{y_2}, \cdots, A_{y_t}$ in Rc, so A_{y_j} $(j \neq 0)$ introduces an extra surviving symbol $d_{p-1-y_j,p-<2y_j>_p}$ compared with the corresponding A'_{y_j}. If all extra surviving symbols $d_{p-1-y_j,p-<2y_j>_p}$ belong to Rc', the number of disk read of Rc will be equal to that of Rc' and will match the lower bound.

We assume Rc matches the lower bound of disk read, and $\exists A_z \in Rc$, $0 < z \leq p-1$, $A_{z-1} \notin Rc$ and $H_{p-1-z} \notin Rc$. The extra surviving symbol introduced by A_z is $d_{p-1-z,p-<2z>_p}$ compared with A_z'. From Lemma 2, we obtain that $A_z \cap A_{z-1} = d_{p-1-z,p-<2z>_p}$, $A_z \cap H_{p-1-z} = d_{p-1-z,p-<2z>_p}$, and the overlapping symbols between A_z and other parity sets do not include $d_{p-1-z,p-<2z>_p}$. According to the relationship between Liberation codes and the simplified form, we can infer that there are just two parity sets A'_{z-1} and H'_{p-1-z} including $d_{p-1-z,p-<2z>_p}$. In addition, $A'_{z-1} \notin Rc'$ and $H'_{p-1-z} \notin Rc'$, so $d_{p-1-z,p-<2z>_p} \notin Rc'$. Therefore, the number of disk read of Rc is at least one more than that of Rc', which contradicts that Rc matches the lower bound of disk read. Therefore, a) of condition 2) holds.

b) When $f > 0$ and $z =< j-f >_p =< f(p-1)/2 >_p$, the extra symbol introduced by A_j is $d_{p-1-j,f}$, which is a failed symbol. Thus, A_j does not introduce any extra surviving symbol compared with A_j'. So if $z =< f(p-1)/2 >_p$ $(f > 0)$, Theorem 2 holds when condition 1) is satisfied. If $z \neq < f(p-1)/2 >_p$ $(f > 0)$, it can be proved that for any z, $0 < z \leq p-1$, if A_z is used to recover a failed symbol, A_{z-1} or H_{p-1-z} must be used to recover another failed symbol in the same way as the proof of a).

Therefore, Theorem 2 holds. ■

From Theorem 2, we can see that we need to judge all the value of x_j, except for $j = f(f \geq 0)$ or $j =< f(p+1)/2 >_p$ $(f > 0)$, since its corresponding anti-diagonal parity set would not bring extra symbols compared with the simplified form.

Example 8: To illustrate, when $p = 7$ and $f = 1$, the recovery sequence $x_0x_1x_2x_3x_4x_5x_6 = 0110100$ is read-optimal, since $\sum_{i=0}^{6} x_i = 3$, when $j = f = 1$ or $j =< f(p+1)/2 >_p = 4$, $x_j = 1$, and when $x_2 = 1$, $x_{<j-1>_p} = x_1 = 1$.

C. READ-OPTIMAL RECOVERY ALGORITHM

We propose an algorithm called *Disk Read Optimal Recovery (DROR)*, which first calls *SearchOptimalSeq* to determine a read-optimal recovery sequence X^*, and then invokes *Recov-WithOptimalSeq* to caculate the failed symbols with X^*.

The main problem we face is how to determine a read-optimal recovery sequence. A straightforward approach is to enumerate all recovery sequences that match 1) of Theorem 2 and traverse them until find one sequence satisfying 2) of Theorem 2. However, there are $\binom{p}{t}$ possible sequences, where $t = (p+1)/2$ or $t = (p-1)/2$, so its time complexity is exponential.

In order to find read-optimal recovery sequences quickly, we put forward an efficient approach, called *SearchOptimalSeq*. We first define a set $Sub_{optimal}$, which includes a portion of the whole read-optimal sequences and each sequence $\{x_0x_1 \cdots x_{p-1}\}$ satisfies the following conditions (N_f is the failed node):

1) $\sum_{i=0}^{p-1} x_i = (p-1)/2, x_f = 1$
2) For any j, $0 \leq j \leq p-1, j \neq f$,
   a) When $f = 0$, if $x_j = 1$, $x_{p-1-<j-f>_p}$ must be zero;
   b) When $f > 0$, $x_{<f(p+1)/2>_p} = 1$, for any $j \neq < f(p+1)/2 >_p$, if $x_j = 1$, $x_{p-1-<j-f>_p}$ must be zero.

The detail of *SearchOptimalSeq* is as follows: the main idea of *SearchOptimalSeq* is to determine the value of every element $x_j \in X$ to make sure X belongs to $Sub_{optimal}$. Let I_X represent the set of j whose corresponding value x_j is not yet determined, $L1$ be the set of j whose corresponding value $x_j = 1$, and $L0$ be the set of j whose corresponding value $x_j = 0$. I_X is initialized to $\{0, 1, \cdots, p-1\}$, and $L1 = L0 = null$. We initialize X with p zeros (Step2). When $j = f(f \geq 0)$ or $j =< f(p+1)/2 >_p$ $(f > 0)$, we add j to $L1$ (Step3-6). To avoid obtaining the same result each run when given p and f, we shuffle the order of elements in I_X (Step7). We repeat Step9-Step14 until $|L_1| == (p-1)/2$. During each iteration, $j = I_X[0]$, and $y = p-1-(j-f)\%p$; if X_j can be set 1, we add j to $L1$ and y to $L0$, otherwise, we add j to $L0$. If $j == y$ or $y \in L1$, $x_j = 1$ is infeasible, since $x_y = 0$ is impossible. After completing all iterations, we set the corresponding $x_j = 1$ of X according to $L1$ (Step16-18).

Algorithm 1 SearchOptimalSeq

Require: p: parameter of Liberation Codes
f: index of failed node
Ensure: X: recovery sequence

```
1: Initialize I_X = {0, 1, ··· , p − 1}, L0 = L1 = null
2: Initialize X with p zeros
3: I_X = I_X − {f}, L1 = L1 + {f}
4: if f > 0 then
5:    j = (f ∗ (p + 1)/2)%p, I_X = I_X − {j}, L1 = L1 + {j}
6: end if
7: Shuffle the order of elements in I_X
8: while L1.length < (p − 1)/2 do
9:    j = I_X[0], y = p − 1 − (j − f)%p
10:   if y == j or y is in L1 then
11:      I_X = I_X − {j}, L0 = L0 + {j}
12:   else
13:      I_X = I_X − {j, y}, L1 = L1 + {j}, L0 = L0 + {y}
14:   end if
15: end while
16: for i = 0 to L1.length do
17:    X[L1[i]] = 1
18: end for
19: return X
```

Example 9: We illustrate Algorithm 1 via an example. Consider a storage system with Liberation code of $p = 11$, and the node $N_f = N_1$ is failure. Initially, $I_X = \{0, 1, \cdots, 10\}$, $L0 = L1 = null$. We delete $f = 1$ and $f*(p+1)/2 = 6$ from I_X, and add them to $L1$. Thus, $I_X = \{0, 2, 3, 4, 5, 7, 8, 9, 10\}$ and $L1 = \{1, 6\}$. Assume $I_X = \{0, 9, 8, 4, 2, 10, 5, 3, 7\}$ is shuffled. We then traverse each element in I_X until $|L1| = (p-1)/2 = 5$. Having $|L1| = 2 < 5$, we begin the 1-th iteration; after this iteration, $I_X = I_X - \{0\}$ and $L0 = L0 + \{0\}$ since $j = y = 0$. With $|L1| = 2 < 5$, and $I_X = \{9, 8, 4, 2, 10, 5, 3, 7\}$, we continue the 2-th iteration; $j = 9$, $y = 2$, $j \neq y$ and $y \notin L1$, so, $L1 = L1 + \{9\}$, $L0 = L0 + \{2\}$, and $I_X = I_X - \{9, 2\}$. After the 3-th and 4-th iterations, $L1 = L1 + \{8\} + \{4\}$, $L0 = L0 + \{3\} + \{7\}$, and $I_X = I_X - \{8, 3\} - \{4, 7\}$. We stop the iteration since $|L1 = \{1, 6, 9, 8, 4\}| = 5$, and then caculate $X = \{0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0\}$. Note that, Algorithm 1 can always find a read-optimal sequence since if $y = p-1- < j-f >_p$, then $p-1- < y-f >_p = j$.

Having a read-optimal recovery sequence is not enough, furthermore, we need to carefully design the order in which the failed symbols are recovered.

Algorithm 2 RecovWithOptimalSeq

Require: p: parameter of Liberation codes
f: index of failed node
X: recovery sequence
Ensure: *null*
1: Initialize $L_0 = null$, $L_1 = null$
2: **for** $i = 0$ to p **do**
3: **if** $X[i] == 0$ **then**
4: $L_0 = L_0 + \{i\}$
5: **else**
6: $L_1 = L_1 + \{i\}$
7: **end if**
8: **end for**
9: **for** $j = 0$ to $L_0.length$ **do**
10: $i = L_0[j]$, recover $d_{i,f}$ by XOR-summing all symbols in $H_i - \{d_{i,f}\}$
11: **end for**
12: **for** $j = 0$ to $L_1.length$ **do**
13: $i = L_1[j]$, recover $d_{i,f}$ by XOR-summing all symbols in $A_{((i-f)\%p)} - \{d_{i,f}\}$
14: **end for**

Example 10: We illustrate the idea by using the Liberation code of Fig. 3 as an example (assume $N_f = N_1$). Suppose the read-optimal recovery sequence X calculated by Algorithm 1 is 01010. Thus, we use H_0, A_0, H_2, A_2, H_4 to recover $d_{0,1}, d_{1,1}, d_{2,1}, d_{3,1}, d_{4,1}$, respectively. The problem is that A_2 chosen for the recovery of $d_{3,1}$ includes the symbol $d_{2,1}$, which is a missing symbol. So $d_{2,1}$ has to be recovered before $d_{3,1}$.

We propose an algorithm called *RecovWithOptimalSeq*, which first recovers the failed symbols with the order from small to large using the horizontal parity sets and then uses the anti-diagonal parity sets to recover the rest failed symbols with the same order. It is easy to prove that this recovery order can recover all failure symbols successfully.

The detail of *RecovWithOptimalSeq* is as follows: given a recovery sequence X, let L_0 and L_1 represent the set of j whose corresponding value $x_j = 0$ and $x_j = 1$, respectively. We first calculate L_0 and L_1 (Step2-6), and then compute the value of failure symbols whose indexes belong to L_0 (Step9-11), and then recover the remaining missing symbols whose indexes belong to L_1 (Step12-14).

Algorithm 3 Disk Read Optimal Recovery (DROR)

Require: p: parameter of Liberation Codes
f: index of failed node
1: $X^* = SearchOptimalSeq\ (p, f)$
2: $RecovWithOptimalSeq\ (p, f, X^*)$

Exapmle 11: We illustrate Algorithm 2 via an example using the same paramete configuration as Example 10. First, we obtain $L0 = \{0, 2, 4\}$, and $L1 = \{1, 3\}$ according to $X = 01010$. And then we recover the symbols $\in L0$ as follows: $d_{0,1} = d_{0,0} \oplus d_{0,2} \oplus d_{0,3} \oplus d_{0,4} \oplus d_{0,5}$, $d_{2,1} = d_{2,0} \oplus d_{2,2} \oplus d_{2,3} \oplus d_{2,4} \oplus d_{2,5}$, and $d_{4,1} = d_{4,0} \oplus d_{4,2} \oplus d_{4,3} \oplus d_{4,4} \oplus d_{4,5}$. Finally, we recover the symbols $\in L1$ in the following way, $d_{1,1} = d_{0,0} \oplus d_{2,2} \oplus d_{3,3} \oplus d_{4,4} \oplus d_{5,5}$, and $d_{3,1} = d_{2,0} \oplus d_{2,1} \oplus d_{4,2} \oplus d_{0,3} \oplus d_{1,4} \oplus d_{2,6}$. The number of symbols read is 19, which is consistent with the optimal value prposed in Theorem 1.

Algorithm Analysis: For *DROR*, the computational cost of *RecovWithOptimalSeq* mainly depends on about p^2 XOR operations, which is equal to that of the conventional approach. Therefore, the extra computational cost of *DROR* comes from *SearchOptimalSeq()*, whose complexity is $O(p)$ if the complexity of judging whether y is in $L1$ or not (Step10) is the least $O(1)$. Therefore, the complexity of *SearchOptimalSeq()* is far less than that of the Hill-climbing Recovery ($O(p^3)$) [33] and Enumeration Recovery ($O(2^p)$) [11].

V. PERFORMANCE EVALUATION

A. ERASURE CODES IN CEPH

We implement the proposed recovery algorithm on Ceph [29], an open source and widely used distributed storage system in industry and academia. Reliable, autonomic distributed object store (RADOS) [35], Ceph's cornerstone, consists of two types of daemons: monitors (MONs) and object storage daemons (OSDs). MONs are primarily in charge of maintaining the cluster map, and OSDs are responsible for storing objects on local filesystems. Generally, a single OSD is used to manage a single hard disk drive (HDD) or SSD.

Ceph stores data as objects within storage pools, each of which has independent access control and redundancy policy. The objects within each pool are collected into placement groups (PGs). A PG contains n OSDs, where n is the number of replicas or the length of the erasure code associated with the pool. The primary OSD (p-OSD) must be elected among the OSDs within the corresponding PG. When an object is stored into a pool, Ceph first calculates which PG the object must belong to and which OSDs need to be assigned to the PG using the controlled replication under scalable hashing (CRUSH) algorithm. Then, Ceph uploads the object to its p-OSD. Finally, p-OSD forwards the object to all other OSDs in a replicated pool or encodes the object, sending chunks to the corresponding OSDs within the same PG in an erasure coded pool.

Ceph supports all types of erasure coding techniques via a pluggable interface, which include RS codes, RAID-6 array codes, LRCs, Clay Codes [36] and so on. The process of encoding objects in Ceph is on-line. That is, when one object is placed into an erasure-coded pool, it is encoded by p-OSD. To reduce the amount of buffer memory required, first, one large object is partitioned into smaller units called *stripes*. Each stripe is then divided into k data chunks and obtains $m =$ n-k parity chunks using (n, k, w) erasure codes. The stripe size S (in bytes) in Ceph refers to the size of k data chunks and can be specified in the configuration file. Furthermore, One chunk is composed of w symbols, the size of which is the multiple of 8192 bytes.

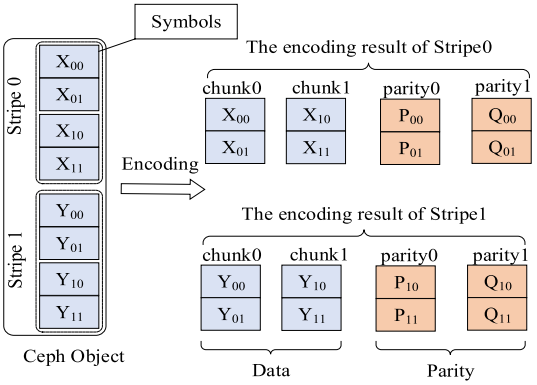


FIGURE 7. The encoding process in Ceph. The object is divided into two stripes, and the symbols within each of the stripes are encoded.

Note that, the p-OSD needs to zero pad each stripe as necessary to guarantee that the stripe size S can be divisible by $8192 \times k \times w$ when encoding, since the stripe size is equal to be $k \times w \times symbol\_size$. When setting a stripe size, zero padding is one significant factor that we need to consider.

Fig. 7 shows an example of the encoding process of one erasure code in Ceph. In this example, Ceph contains two data nodes and two parity nodes ($k = m = 2$). The original object is divided into two stripes, each of which is composed of two chunks labeled as chunk0 and chunk1 respectively. Chunk0 is further divided into two symbols X00, X01 in Stripe0 and Y00, Y01 in Stripe1.

B. DROR IMPLEMENTATION

We used the Jerasure Erasure Code plugin [34], which provides an implementation of Liberation Codes based on the Jeasure to implement the proposed recovery approach. In the current Ceph implementation, it only uses the P node to recover a single data node failure, just like the conventional recovery approach. More specifically, when repairing a data chunk, it uses the function *minimum_to_decode()* to obtain the IDs of k helper chunks, makes use of the function *decode()* to acquire the original data stripe, and only regenerates the failure chunk using the function *encode()*.

Under above mode, we can not recover a failure data chunk by downloading other chunks at the granularity of symbol, which goes against the realization of our recovery approach. We introduce a new concept, namely, *optimal-recovery-condition*, which is the condition that the erasure-coded pool adopts the Liberation coding technique which satisfies $k = w$ and only one failure data node needs to be recovered.

To support our recovery approach, we provide three extra functions and modify some existing structures as follows.

-is_repair_liberationcodes_optimal(): This boolean function allows the choice between an optimal repair algorithm for Liberation codes and the default decode algorithm. It returns true, in case the the *optimal-recovery-condition* is satisfied, and vice versa.

-minimum_to_repair_librationcodes(): It invokes the proposed algorithm DROR() to determine the optimal recovery sequence *ReSeq* and transforms *ReSeq* into *ChunkSolution*, which includes a list of helper chunk indices, in addition, each of them has corresponding symbol IDs.

-repair_liberationcodes(): Given the failure chunk ID and *ChunkSolution*, returned by *minimum_to_repair_ librationcodes()*, it reconstructs a failure chunk.

In order to read a symbol, a fraction of chunk, we feed some repair parameters to existing structures, including *read_request_t*, *ReadOp*, and *ECSubRead*. We also design a novel read function *read_for_liberationcodes()* with Filestore of Ceph to allow symbols read.

C. SETUP AND OVERVIEW

1) EXPERIMENTAL SETUP

We integrated the DROR approach in Ceph Luminous 12.0.2. To evaluate the proposed recovery algorithm, we conduct the experiments on Amazon EC2 instances of the t3a.large (8GB RAM, 2 CPU cores) configuration. We compare the performance of proposed recovery approach with that of the default recovery approach in Ceph, which is called *conventional approach* in the following. The Ceph storage cluster contains 17 nodes. One server runs one MON deamon, while each of the remaining 16 nodes runs one OSD deamon. Each instance is attached one HDD-type volume used for the OS, one SSD-type volume of size 200GB specialized for the OSD storage, and one SSD-type volume of size 12GB for the Ceph journal. Thus the total cluster capacity is 3.1 TB.

2) OVERVIEW

The failure domain is set as a node or an OSD as well since there is one OSD for each node. The workload is chosen a fixed size of 64MB. We first write 1024 objects of size 64MB each to a single PG using a specified Liberation-coded scheme. Then, we make produce one node failure by setting the state of the OSD as out. Once an OSD is out, the Ceph system will perform data recovery operations. For Ceph, we have to build a new erasure-coded pool if we change one or more parameters of erasure codes, e.g., stripe-size S, and the number of data nodes k. Moreover, the assignment of OSDs to PGs are performed in a dynamic fashion. Therefore, we evaluate the average disk read, network traffic and recovery time by making different single data node failures and the recovery process of one data node failure is performed 3 times. We use nmon and NMONVisualizer tools to collect and analyze data respectively.

3) CODES EVALUATED

In fact, the number of nodes n for erasure codes need to be no more than 20 [3], [4], so $k \leq 18$ for Liberation codes $(k+2, k, w)$. Besides, the Liberation codes we consider is $k = w$. We have conducted experimental evaluation for Liberation codes $C_1(7,5,5)$, $C_2(9,7,7)$, $C_3(13,11,11)$, and $C_4(15,13,13)$, which all satisfy above conditions. The evaluation of codes $C_1 - C_4$ is carried out in Ceph for single node failures with fixed symbol size and code C_3 is evaluated with different symbol sizes.

D. RESULTS

1) PERFORMANCE WITH DIFFERENT SYMBOL SIZES

We measure the disk read, network traffic and recovery time of one data node failure for Liberation codes with different symbol sizes for code C_3.

The object size after zeropadded O_{padded} is equal to

$$\begin{aligned} O_{padded} &= stripe\_size \times stripe\_num \\ &= symbol\_size \times k \times w \times \left\lceil \frac{object\_size\_ori}{symbol\_size \times k \times w} \right\rceil \\ &= symbol\_size \times p^2 \times \left\lceil \frac{object\_size\_ori}{symbol\_size \times p^2} \right\rceil, \end{aligned}$$

where $p = k = w$ and the original object size is $object\_size\_ori$ bytes. The symbol size shoule be the multiple of 8KB in Ceph, and we want to generate as little as possible zero padding. So the maximum symbol size S_{max} can be calculated by setting the stripe size greater than or equal to the original object size with the least zero padding, which is given as follows:

$$\begin{aligned} S_{max} &= \left\lceil \frac{object\_size\_ori}{8192 \times k \times w} \right\rceil \times 8192 \\ &= \left\lceil \frac{object\_size\_ori}{8192p^2} \right\rceil \times 8192. \end{aligned} \tag{7}$$

We want to produce the same zero padding for all symbol sizes, therefore symbol sizes S1, S2, S3, and S4 are set 8KB, 32KB, 136KB, and 544KB respectively. Object sizes are all equal to 65824KB for all symbol sizes we evaluated, which will bring a negligible data padding of 0.04%.

a: DISK READ

Disk read here means the amount of data read from the disks of the helper nodes when recovering a failed node. Recovery is performed by the p-OSD, which is also a helper node. Theoretically the disk read R_{dror} for DROR is $(3p^2 + 1)T/4$ and the disk read R_{ori} for original method is p^2T, where $T = symbol\_size \times stripe\_num \times object\_num = O_{padded} \times object\_num/p^2$. Since O_{padded} are identical, disk read in theory is equivalent for symbol sizes S1-S4 for both methods.

The amount of disk read rests on the symbol-size for SSD-type volumes, in which reads are at a granularity of 4KB. The symbol sizes in Ceph are all the multiple of 8KB, which is aligned to the granularity of SSD reads. So it does not cause any additional disk read under different symbol-sizes for both recovery approaches as shown in Fig. 8. In addition, we observe a saving of 24.8% in disk read for DROR in comparison with conventional method under all symbol sizes $S_1 - S_4$.

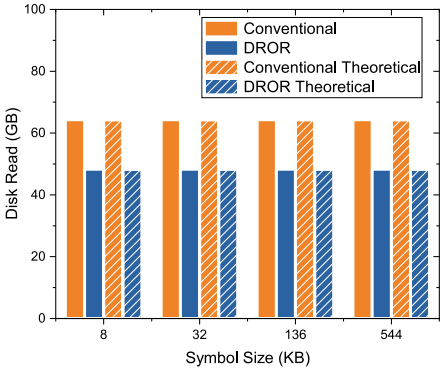


FIGURE 8. Disk read of DROR and Conventional approaches with different symbol sizes when $k = w = 11$.

b: NETWORK TRAFFIC

Network traffic here refers to the data transferred across the network during single node recovery. The network traffic during recovery includes both the amount of data transferred from helper nodes to the primer OSD and the amount of recovered chunk transferred from the primary OSD to the replacement OSD. For original method, net traffic N_{ori} is equal to disk read, $N_{ori} = R_{ori} = p^2T$. For DROR, the data transferred of helper nodes is equal to the data read from disks during recovery except for the primer OSD, which transfers p symbols and reads less than p symbols in each stripe. Besides, network traffic of the primer OSD is about $(p + 1)/4$ symbols more than disk read within one stripe. Specifically, for DROR, network traffic N_{dror} is

$$N_{dror} = \begin{cases} R_{dror} + \dfrac{p+1}{4}T, & <p>_4 = 3 \\ R_{dror} + \left\lfloor \dfrac{p+1}{4} \right\rfloor T \ \ or \ \ R_{dror} + \left\lceil \dfrac{p+1}{4} \right\rceil T, & \\ & <p>_4 = 1. \end{cases}$$

Network traffic here $N_{dror} = R_{dror} + (p + 1)T/4 = (3p^2 + p + 2)T/4$ for DROR, since $p = 11$. For both recovery approaches, theoretical network traffic is equivalent for symbol sizes $S_1 - S_4$ with same T. While in practice network traffic decreases slightly as the symbol size increases for both methods as shown in Fig. 9. We observe that network traffic in fact is averagely increased by 13.4% and 12.3% than that in theory for conventional approach and DROR respectively. As can be seen, we obtain an average reduction of 23.0% in network traffic for DROR compared with conventional approach, which is consistent with the theoretical value of 22.4%.

c: RECOVERY TIME

Recovery time is measured by acquiring the times of initial and end repair activities of the primer OSD when recovering all 1024 objects. Fig. 10 shows the average recovery time of per object for both recovery methods under different symbol sizes. We observe that the recovery time for both approaches decreases with the increase of symbol size. That is mainly because the number of disk I/O requests and the network traffic decrease as the symbol size increases. As can be seen, the recovery performance of DROR is superior to that of conventional approach for all symbol sizes. When the symbol size is maximum, for DROR, we observe an performance increasement of 22.1% in comparison with the conventional method.

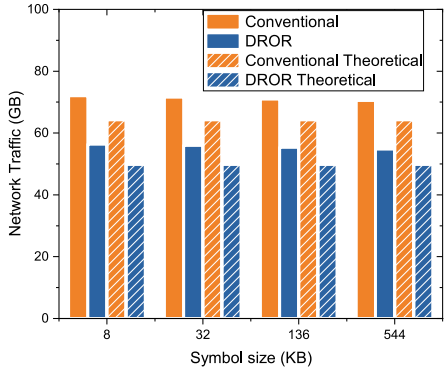


FIGURE 9. Network traffic of DROR and Conventional approaches with different symbol sizes when $k = w = 11$.

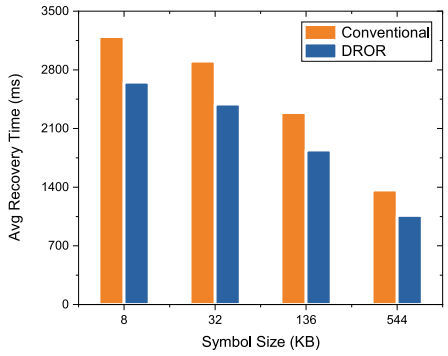


FIGURE 10. Recovery time of per object for DROR and Conventional approaches with different symbol sizes when $k = w = 11$.

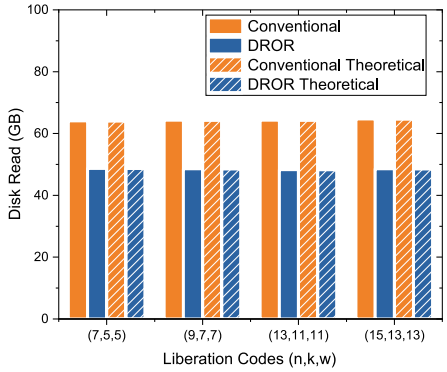


FIGURE 11. Disk read for DROR and Conventional approaches with different number of nodes.

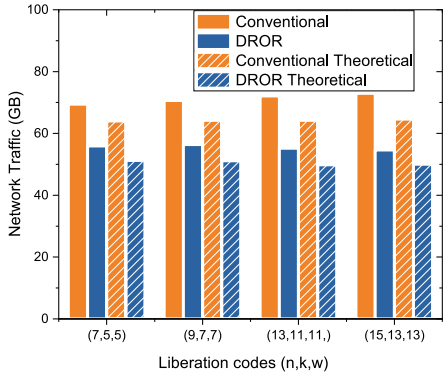


FIGURE 12. Network traffic for DROR and Conventional approaches with different number of nodes.

In the following experiments, the symbol sizes are fixed to be the largest ones for different number of data nodes, in which the optimal recovery performance is obtained for both recovery approaches.

2) PERFORMANCE WITH DIFFERENT NUMBER OF DATA NODES

We conduct experiments with different number of data nodes 5, 7, 11, and 13, which are corresponding to Liberation Codes C_1, C_2, C_3 and C_4 respectively. The symbol sizes are set to be maximum for all Liberation codes evaluated and can be calculated by Equation 7.

a: DISK READ

In theory, for conventional method, the amount of data read is positive related to the object size after zero padded, so disk read of C_4 is a little more than other three codes $C_1 - C_3$. However this is not so for DROR, in which C_1 reads the most among $C_1 - C_4$. Anyhow, disk read of codes $C_1 - C_4$ are nearly the same for both methods. In practice, for both recovery methods, the amount of data read is identical to that in theory, which can be seen from Fig. 11. For DROR, we observe reductions of 24.0%, 24.4%, 24.9%, and 24.9% in disk read for codes C_1, C_2, C_3 and C_4 respectively in comparison with the corresponding codes of the conventional approach.

b: NETWORK TRAFFIC

The theoretical network traffic of codes $C_1 - C_4$ is almost the same for both recovery methods, as shown in Fig. 12. As a matter of fact, for conventional method network traffic is increased along with an increasing number of data nodes, and network traffic is reduced slightly for DROR. We observe that the actual network traffic is increased by 10.8% on average compared with the theoretical value for conventional method, while increased by 9.4% for DROR. As can be seen, for DROR, we obtain reductions of 20.0%, 21.5%, 23.4% and 25.1% in network traffic for codes C_1, C_2, C_3 and C_4 compared with the corresponding codes of the conventional approach. The reductions are 20.0%, 20.4%, 22.4%, and 23.0% for codes $C_1 - C_4$ respectively in theory. The reductions we obtain are close to or better than the theoretical value. This is mainly because DROR has lower ratio of the actual network traffic to the theoretical value for codes $C_2 - C_4$ compared with conventional method.

c: RECOVERY TIME

In Fig. 13, we show the average recovery time of per object for both recovery methods under different codes $C_1 - C_4$. We observe that the recovery time for both approaches increases as the number of nodes goes up. This is mainly because the recovery processes of both recovery schemes may

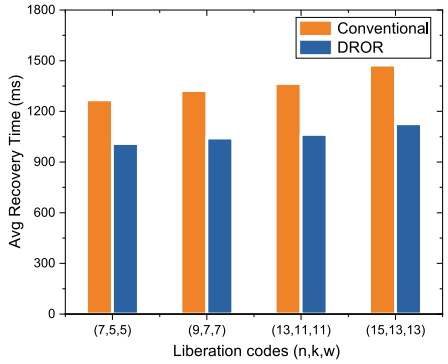


FIGURE 13. Recovery time of per object for DROR and Conventional approaches with different number of nodes.

cost more time in waiting and synchronizing as the number of nodes increases. As shown in Fig. 13, the recovery performance of DROR outperforms that of conventional approach for all codes $C_1 - C_4$. This is mainly due to reduction in disk I/O and network traffic during repair. Furthermore, DROR provides improvements of 20.4%, 21.2%, 22.1%, and 23.6% for codes C_1, C_2, C_3 and C_4 respectively, and the improvement increases along with the increase of the number of data nodes.

VI. CONCLUSION

We study the problem of minimizing the number of symbols read from surviving nodes when repairing an erased data node in Liberation coded storage systems. We first derive the lower bound of disk read to reconstruct a single node failure for Liberation codes using a relaxation technique. Then we elaborate the conditions that a read-optimal recovery sequence need to satisfy. Finally, we propose an optimal recovery approach DROR based on a subset of optimal recovery conditions, which can recover single node failures reading the minimum number of symbols. We implement DROR on a Ceph cluster deployed on Amazon EC2. Experimental results show that DROR can reduce recovery time by up to 23.6% compared to the conventional recovery approach.

REFERENCES

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proc. 19th ACM Symp. Operating Syst. Princ. (SOSP)*, 2003, pp. 29–43.
[2] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *Proc. ACM SOSP*, 2007, pp. 205–220.
[3] C. Huang, H. Simitci, Y. Xu, A. Ogus, B, Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Proc. USENIX ATC*, 2012, pp. 15–26.
[4] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing elephants: Novel erasure codes for big data," in *Proc. VLDB*, 2013, pp. 1–12.
[5] Facebook Developers. *Facebook's Erasure Coded Hadoop Distributed File System (HDFS-RAID)*. Accessed: Nov. 16, 2017. [Online]. Available: https://github.com/facebookarchive/hadoop-20
[6] H. Zhang, M. Dong, and H. Chen, "Efficient and available in-memory KV-store with hybrid erasure coding and replication," in *Proc. USENIX FAST*, 2016, pp. 167–180.
[7] M. Abebe, K. Daudjee, B. Glasbergen, and Y. Tian, "EC-store: Bridging the gap between storage and latency in distributed erasure coded systems," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 255–266.
[8] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, Jun. 1960.
[9] B. Schroeder and G. Gibson, "Disk failures in the real world: What does an MTTF of 1,000,000 mean to you?" in *Proc. USENIX FAST*, 2007, pp. 1–16.
[10] L. Xiang, Y. Xu, J. C. S. Lui, Q. Chang, Y. Pan, and R. Li, "A hybrid approach to failed disk recovery using RAID-6 codes: Algorithms and performance evaluation," *ACM Trans. Storage*, vol. 7, no. 3, pp. 1–34, Oct. 2011.
[11] O. Khan, R. C. Burns, J. S. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads," in *Proc. USENIX FAST*, 2012, pp. 1–14.
[12] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V. A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed file system," in *Proc. USENIX OSDI*, 2010, pp. 61–74.
[13] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *Proc. USENIX FAST*, 2007, pp. 17–29.
[14] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 192–202, Feb. 1995.
[15] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *Proc. USENIX FAST*, 2004, pp. 1–14.
[16] M. Blaum and R. M. Roth, "On lowest density MDS codes," *IEEE Trans. Inf. Theory*, vol. 45, no. 1, pp. 46–59, Jan. 1999.
[17] J. S. Plank, "The RAID-6 liberation codes," in *Proc. USENIX FAST*, 2008, pp. 1–14.
[18] J. S. Plank, "A new minimum density RAID-6 code with a word size of eight," in *Proc. 7th IEEE Int. Symp. Netw. Comput. Appl.*, Jul. 2008, pp. 85–92.
[19] J. Bloemer *et al.*, "An XOR-based erasure-resilient coding scheme," Int. Comput. Sci. Inst., Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. TR-95-048, Aug. 1995.
[20] L. Xu and J. Bruck, "X-code: MDS array codes with optimal encoding," *IEEE Trans. Inf. Theory*, vol. 45, no. 1, pp. 272–276, 1st Quart., 1999.
[21] L. Xu, V. Bohossian, J. Bruck, and D. G. Wagner, "Low-density MDS codes and factors of complete graphs," *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 1817–1826, Sep. 1999.
[22] C. Wu, S. Wan, X. He, Q. Cao, and C. Xie, "H-code: A hybrid MDS array code to optimize partial stripe writes in RAID-6," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2011, pp. 782–793.
[23] C. Jin, H. Jiang, D. Feng, and L. Tian, "P-code: A new RAID-6 code with optimal properties," in *Proc. 23rd Int. Conf. Conf. Supercomput. (ICS)*, 2009, pp. 360–369.
[24] Z. Shen and J. Shu, "HV code: An all-around MDS code to improve efficiency and reliability of RAID-6 systems," in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2014, pp. 550–561.
[25] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson, "DiskReduce: RAID for data-intensive scalable computing," in *Proc. 4th Annu. Workshop Petascale Data Storage (PDSW)*, 2009, pp. 6–10.
[26] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. S. Sarma, R. Murthy, and H. Liu, "Data warehousing and analytics infrastructure at facebook," in *Proc. Int. Conf. Manage. Data (SIGMOD)*, 2010, pp. 1013–1020.
[27] K. Hwang, H. Jin, and R. Ho, "RAID-x: A new distributed disk array for I/O-centric cluster computing," in *Proc. 9th Int. Symp. High-Perform. Distrib. Comput.*, 2000, pp. 279–286.
[28] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence, "FAB: Building distributed enterprise disk arrays from commodity components," *SIGARCH Comput. Archit. News*, vol. 32, no. 5, pp. 48–58, Oct. 2004.
[29] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. USENIX OSDI*, 2006, pp. 307–320.
[30] L. Pamies-Juarez, F. Blagojevic, and R. Mateescu, "Opening the chrysalis: On the real repair performance of MSR codes," in *Proc. FAST*, 2016, pp. 81–94.
[31] Z. Wang, A. G. Dimakis, and J. Bruck, "Rebuilding for array codes in distributed storage systems," in *Proc. IEEE Globecom Workshops*, Dec. 2010, pp. 1905–1909.

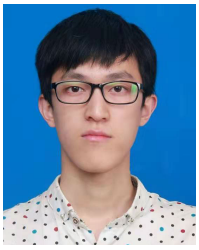
[32] S. Xu, R. Li, P. P. C. Lee, Y. Zhu, L. Xiang, Y. Xu, and J. C. S. Lui, "Single disk failure recovery for X-Code-Based parallel storage systems," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 995–1007, Apr. 2014.
[33] Y. Zhu, P. P. C. Lee, Y. Hu, L. Xiang, and Y. Xu, "On the speedup of single-disk failure recovery in XOR-coded storage systems: Theory and practice," in *Proc. IEEE 28th Symp. Mass Storage Syst. Technol. (MSST)*, Apr. 2012, pp. 1–12.
[34] (2017). *Jerasure Erasure Code Plugin*. Accessed: Sep. 24, 2017. [Online]. Available: http://docs.ceph.com/docs/hammer/rados/ operations/erasure-code-jerasure/
[35] S. A. Weil, A. W. Leung, S. A. Brandt, and C. Maltzahn, "RADOS: A scalable, reliable storage service for petabyte-scale storage clusters," in *Proc. 2nd Int. Workshop Petascale Data Storage Held Conjunct With Supercomput. (PDSW)*, 2007, pp. 35–44.
[36] M. Vajha, V. Ramkumar, B. Puranik, G. Kini, E. Lobo, B. Sasidharan, P. V. Kumar, A. Barg, M. Ye, S. Narayanamurthy, and S. Hussain, "Clay codes: Moulding MDS codes to yield an MSR code," in *Proc. USENIX FAST*, Oakland, CA, USA, 2018, pp. 139–153.

NINGJING LIANG received the M.S. degree from the School of Computer Science and Technology, Xidian University, Shaanxi, China, in 2015. She is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Xi'an Jiaotong University, Shaanxi. Her research interests include erasure codes, distributed storage systems, and cloud computing.

XINGJUN ZHANG (Associate Member, IEEE) received the Ph.D. degree in computer architecture from Xi'an Jiaotong University, China, in 2003. From January 2004 to December 2005, he was a Postdoctoral Fellow with the Computer School, Beihang University, China. From February 2006 to January 2009, he was a Research Fellow with the Department of Electronic Engineering, Aston University, U.K. He is currently a Full Professor and the Dean of the School of Computer Science and Technology, Xi'an Jiaotong University. His research interests include high-performance computing, big data storage systems, and machine learning acceleration.

HAILONG YANG (Member, IEEE) received the Ph.D. degree from the School of Computer Science and Engineering, Beihang University, in 2014. He is currently an Assistant Professor with the School of Computer Science and Engineering, Beihang University. He has also been involved in several scientific projects, such as performance analysis for big data systems and performance optimization for large scale applications. His research interests include cloud computing, big data, and HPC.

XIAOSHE DONG received the B.Eng. degree in computer hardware from Xi'an Jiaotong University, Xi'an, China, in 1985, and the M.S. and Ph.D. degrees in computer architecture from Keio University, Tokyo, Japan, in 1996 and 1999, respectively. He was a Lecturer, from 1987 to 1994, and an Associate Professor, from 1999 to 2003, with the Department of Computer Science and Engineering, Xi'an Jiaotong University, where he has been a Full Professor, since 2003. His research interests include high-performance computer architecture and grid computing.



CHANGJIANG ZHANG received the B.S. degree from the School of Information Science and Technology, Northwestern University, Shaanxi, China, in 2019. He is currently pursuing the M.Phil. degree with the School of Computer Science and Technology, Xi'an Jiaotong University, Shaanxi. His research interests include cloud storage systems and network coding.