Reconsidering Single Disk Failure Recovery for Erasure Coded Storage Systems: Optimizing Load Balancing in Stack-Level

Yingxun Fu, Jiwu Shu, *Member, IEEE*, Zhirong Shen, and Guangyan Zhang

Abstract—The fast growing of data scale encourages the wide employment of data disks with large storage capacity. However, a mass of data disks' equipment will in turn increase the probability of data loss or damage, because of the appearance of various kinds of disk failures. To ensure the intactness of the hosted data, modern storage systems usually adopt erasure codes, which can recover the lost data by pre-storing a small amount of redundant information. As the most common case among all the recovery mechanisms, the single disk failure recovery has been receiving intensive attentions for the past few years. However, most of existing works still take the stripe-level recovery as their only consideration, and a considerable performance improvement on single failure disk reconstruction in the stack-level (i.e., a group of rotated stripes) is missed. To seize this potential improvement, in this paper we systematically study the problem of single failure recovery in the stack-level. We first propose two recovery mechanism based on greedy algorithm to seek for the near-optimal solution (BP-Scheme and STP-Scheme) for any erasure array code in stack level, and further design a rotated recovery algorithm (RR-Algorithm) to eliminate the size of required memory. Through a rigorous statistic analysis and intensive evaluation on a real system, the results show that BP-Scheme gains 3.4 to 38.9 percent (the average is 21.2 percent) higher recovery speed than Khan's Scheme and 3.4 to 34.8 percent (the average is 19.1 percent) higher recovery speed than Luo's U-Scheme, while STP-Scheme owns 3.4 to 46.9 percent (the average is 25.15 percent) and 3.4 to 41.1 percent (the average is 22.3 percent) higher recovery speed than Khan's Scheme and Luo's U-Scheme, respectively.

Index Terms—Single failure recovery, erasure code, stack, storage system

1 Introduction

With the amazing expansion of data scale, hundreds of thousands of disks are introduced in modern storage systems, such as GFS [1], Windows Azure [2], and OceanStore [3]. However, the employment of large number of disks not only leads to the high complexity of storage systems, but also increases the probability of data lost or damaged caused by various kinds of disk errors. To enhance data reliability, erasure codes, which can tolerate several disk failures by pre-storing a reasonable size of redundant information, is widely adopted in nowadays.

For erasure coded storage systems, a small amount of redundant information (called parity) is calculated based on the kept data elements and once a disk failure happens, the storage system will retrieve a subset of available data elements and the pre-stored parity elements from the surviving disks, so that the lost data will be successfully recovered. Among the diverse failure patterns in storage systems, previous study [4] has revealed that most of the recovery processes (about 99.75 percent) are triggered by single disk failures. This finding makes the single disk failure recovery as one of the hottest issues in the literature of erasure codes in the past few years.

• *The authors are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: mooncape1986@126.com, {shujw, gyzh}@tsinghua.edu.cn, zhirong.shen2601@gmail.com.*

Manuscript received 25 Feb. 2015; revised 2 June 2015; accepted 2 June 2015. Date of publication 8 June 2015; date of current version 13 Apr. 2016. Recommended for acceptance by R. Brightwell.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2442979

For a long time, it was thought that the single disk failure recovery is an expensive task, as it usually requires to read all the remained data elements and a considerable amount of parity elements. However, things change when Xiang et al. [5] first propose a hybrid recovery method for RDP code [6], in which the horizontal parity and diagonal parity are alternative chosen to achieve the maximum number of overlapped data elements, so that at most 25 percent elements to be retrieved will be reduced. A more general scheme for single disk failure recovery is then provided by Khan et al. [7], which seeks for the solutions with the minimal retrieved elements for any erasure array code. Inspired by the fact that the reconstruction time is determined by the disk loaded with the heaviest requests, Luo and Shu [8] accordingly proposes two search-based schemes to balance the load among disks and accelerate the single disk failure recovery for any erasure array codes.

However, most of previous works still restrict their attention on the stripe-level recovery for single failures, but miss a fact that the real storage systems usually rotationally map the logical disks to physical disks [7],[9],[10], in order to alleviate the parity disks from being the hot point when suffering from a huge number of writes (more details will be referred in Section 2.1). Though Luo's schemes well speed up the process of single disk failure recovery in the stripe-level, it is still unknown which method is the best choice once moving to the scenario where logical disks are mapped rotationally.

In this paper, we systematically study this problem and propose two search-based algorithms called balance priority scheme (BP-Scheme) and search time priority scheme (STP-Scheme). BP-Scheme first collects the feasible recovery

solutions for each stripe by enumerating all feasible solutions like in Khan's Scheme, and then starts with a primary set constituted with a group of feasible solutions picked from each stripe. BP-Scheme then incrementally replaces the current solution in each stripe with another one by using simulated annealing (SA) algorithm [11], once the refined set achieves the lighter load on the busiest disk, until a near-optimal solution with the minimal data accesses on the heaviest disk could be found in the stack-level. Different from BP-Scheme, STP-Scheme directly generate a feasible stack-based solution by recovery equations as the initial solution and utilize the simulated annealing algorithm to refine this solution, until a solution with satisfied data accesses on the heaviest disk has been returned.

On the other hand, recovery based on stack-level will cause huge memory overheads. In order to eliminate the memory requirement for BP-Scheme and STP-Scheme realization, we then develop a rotated recovery algorithm (RR-Algorithm), which iteratively reads a constant number of elements during each parallel I/O process and invokes the reconstruction of the lost elements of a stripe once the needed elements in that stripe are all retrieved. Furthermore, though BP-Scheme and STP-Scheme are both stack-based algorithms and have similar recovery speed, they provide quite different I/O cost and search time (compared to BP-Scheme, STP-Scheme achieves lower search time but suffers from higher I/O cost). The system designers could simultaneously consider recovery speed, recovery cost and search time to select one of them as recovery method, or implement both of them in order to choose the one with higher theoretical recovery speed. Our contributions can be summarized as follows:

- We consider the single disk failure recovery in the stack-level and propose two search-based schemes (BP-Scheme and STP-Scheme) to seek for the near-optimal solution with the minimal retrieved elements on the heaviest loaded disk. To reduce the memory overhead, we subsequently provide a RR-Algorithm.
- We make a series of quantity analysis on various kinds of erasure array codes. The results demonstrate that the theoretical recovery speed of our proposed BP-Scheme and STP-Scheme is not only much higher that of other competitors, but also very close to the theoretical optimal recovery speed in the stack-level.
- We finally implement our algorithms in a real storage system based on Jerasure-1.2 [12] library, and evaluate their performance with different configurations. The results show that our proposed schemes perform much faster than other existing recovery schemes. Specifically, BP-Scheme gains 3.4 to 38.9 percent (the average is 21.2 percent) higher recovery speed than Khan's Scheme and 3.4 to 34.8 percent (the average is 19.1 percent) higher recovery speed than Luo's U-Scheme, while STP-Scheme owns 3.4 to 46.9 percent (the average is 25.15 percent) and 3.4 to 41.1 percent (the average is 22.3 percent) higher recovery speed than Khan's Scheme and Luo's U-Scheme, respectively.

The rest of this paper continuous as follows: The next section provides the research background and related works. Section 3 discusses the problems in existing recovery schemes, and then gives our motivations. We present the detailed designs of our proposed algorithms in Section 4, and discusses their time complexity in Section 5. Section 6 gives a mathematical analysis on the recovery speed. An intensive evaluations are carried out and the comparison results are shown in Section 7. Finally, we conclude our work in the last section.

TABLE 1
The Frequently Used Symbols

Symbols	Description
n	The number of total logical disks
k	The number of logical data disks
m	The number of logical parity disks
w	The amount of elements in each column
l	The amount of stripes in a stack
D_i	Logical data disks
P_i	Logical parity disks
S_i	The ith stripe
$d_{i,j}$ & $p_{i,j}$	The ith element of the jth column
e_i	Recovery equation
$E_{d_{i,j}}$ & $E_{d_{p_{i,j}}}$	All recovery equation of $d_{i,j}$ or $p_{i,j}$
E_{all}	The full set of recovery equations

2 Background and Related Work

In this section, we introduce the background and the related works. Firstly, we summarize the frequently used symbols across this paper in Table 1.

2.1 Background: Erasure Coded Storage Systems

Terms and notations. We first define the frequently used terms based on [10]: An n-disks erasure coded storage system is partitioned into k logic disks (denoted as D_0, D_1, ..., D_{k-1}) that keep original data, and $m = n - k$ logic disks (denoted as P_0, P_1, ..., P_{m-1}) that store the parity information.

An erasure coded storage system is also partitioned into stripes, which are maximal sets of disk blocks that are independent on each other in terms of redundancy relations. Each block is partitioned into a fixed number of elements, which are fixed-size units of data or parity information and the number is denoted as w. We label the w elements on the ith data disk as $d_{i,0}$, $d_{i,1}$, ..., $d_{i,w-1}$, and on the ith parity disk as $p_{i,0}$, $p_{i,1}$, ..., $p_{i,w-1}$. For example, Fig. 1 shows a stripe of erasure code with $n = 8$, $k = 6$, $m = 2$, and $w = 6$.

Generator bitmatrix for erasure array codes. All erasure array codes can be represented by the generator matrix product. We show an example of Cauchy Reed-Solomon code with $n = 6$, $k = 4$, $m = 2$, and $w = 3$ in Fig. 2. As the figure shows, the kr data elements are organized as a kr-element bit vector, while the generator matrix is a $wn \times wk$ bit matrix. Based on the generator bitmatrix, we can easily compute all the parities, i.e., computing the parity information in P_0 to P_{m-1}.

Erasure coded storage systems. Though Fig. 1 shows that some disks (they are logic disks) only contains parities, when it comes to the real systems, each physical disk may contain both data and parities, because the mappings from the logic disks (i.e., each D_i and P_i) to physical disks are usually rotated. If a disk only contains parities, the storage

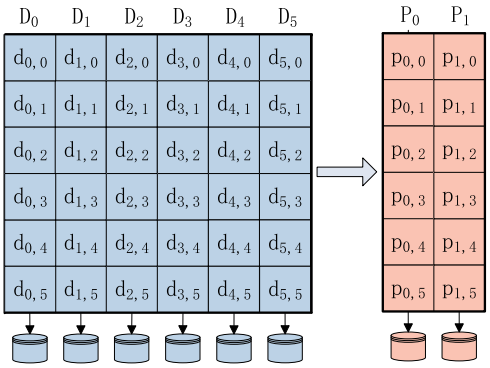


Fig. 1. An example of one stripe in erasure coded storage systems, where $n = 8$, $k = 6$, $m = 2$, and $w = 6$.

system usually encounters a bottleneck on this disk under intensive write operations, because the update of parities is more frequent. Some researches on RAID architectures and algorithms define these rotated stripes as a **stack** [13]. We follow this definition in this work.

Fig. 3 illustrates a widely used case of a stack with the mappings from logic disks to physical disks [7],[9],[10]. As it shows, in the first stripe, the first k disks hold data, while the last m disks store parities. When it comes to the nth stripe, the first disk and the last $m-1$ disks hold parities, while the other disks hold data. By rotated the mappings, each physical disk has the same probability to map with each logic disk, which is effective to alleviate the bottleneck on parity disks.

2.2 Related Work

The conventional method to recover from single failures is to select k surviving disks and create a kw-element decoding bitmatrix from the corresponding rows of generator matrix. The product of the wk elements (in the k surviving disks) and the converted decoding bitmatrix will generate the original wk data elements. In recent years, some researches have been proposed for improving the recovery speed. In this section, we introduce these works.

Recovery equations. A recovery equation is composed by a series of data elements and parity elements, whose XOR sum equal to zero. If one element of the recovery equation is lost, we can reconstruct it by other survived elements. For example, in Fig. 2, $d_{0,0}$, $d_{1,0}$, $d_{2,0}$, $d_{3,0}$, and $p_{0,0}$ form a specific recovery equation.

When some subset F of the elements are failed, each failed element may belong to a group of recovery equations.

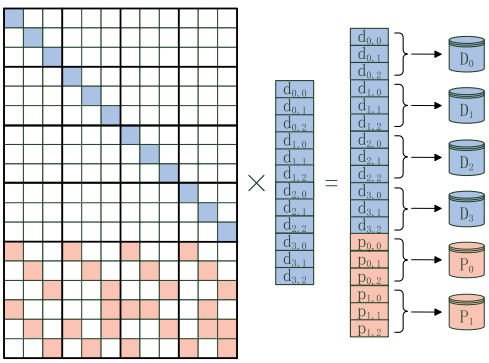


Fig. 2. An example of generator bitmatrix of Cauchy Reed-Solomon code with $n = 6$, $k = 4$, $m = 2$, and $w = 3$.

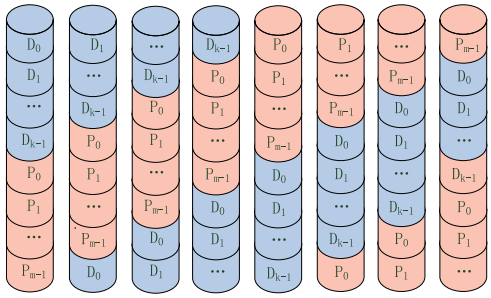


Fig. 3. A widely used case of one stack in practical erasure coded storage systems.

We denote $E_{d_{i,j}}$ and $E_{p_{i,j}}$ as the maximum set of recovery equations that can be used to reconstruct $d_{i,j}$ and $p_{i,j}$, respectively. Suppose that the maximum set of all equations is E_{all}. For each equation $e_i \in E_{all}$, if $e_i \cap F = d_{i,j}$, then $e_i \in E_{d_{i,j}}$. Similarly, if $e_i \cap F = p_{i,j}$, then $e_i \in E_{p_{i,j}}$. Recovery equations will facilitate our discussion and can be used in finding the hybrid recovery solutions.

Hybrid recovery principle. Xiang et al. [5] first proposed a hybrid recovery method for RDP code. Thanks to the overlapping elements, this method reduces up to 25 percent I/O cost than the conventional method. We now explain the hybrid recovery principle based on the example of Fig. 2.

Suppose that D_0 is failed, we can recovery all the failed elements under conventional recovery method as follow:

- $d_{1,0}$, $d_{2,0}$, $d_{3,0}$ and $p_{0,0}$ to recover $d_{0,0}$.
- $d_{1,1}$, $d_{2,1}$, $d_{3,1}$ and $p_{0,1}$ to recover $d_{0,1}$.
- $d_{1,2}$, $d_{2,2}$, $d_{3,2}$ and $p_{0,2}$ to recover $d_{0,2}$.

This method needs to read 12 elements in total. However, if we use some parity elements in P_1, the failed elements can be reconstructed as follow:

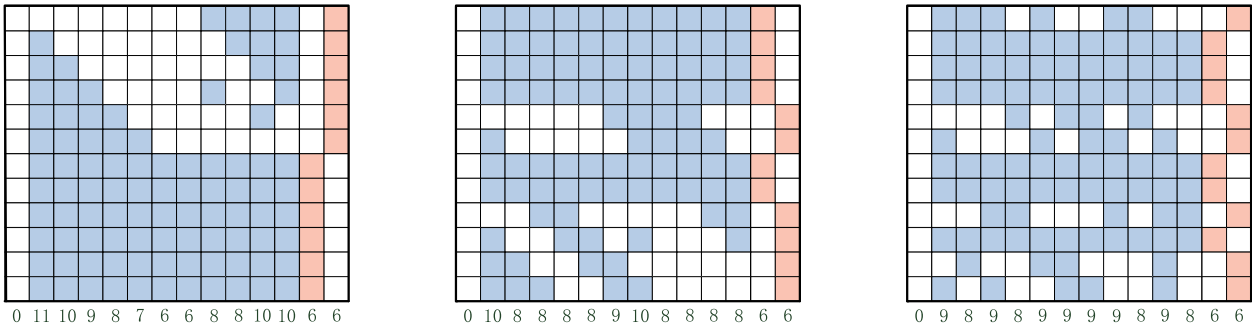
- $d_{1,0}$, $d_{2,0}$, $d_{3,0}$ and $p_{0,0}$ to recover $d_{0,0}$.
- $d_{1,1}$, $d_{2,1}$, $d_{3,1}$ and $p_{0,1}$ to recover $d_{0,1}$.
- $d_{1,1}$, $d_{2,0}$, $d_{3,0}$, $d_{3,2}$ and $p_{1,2}$ to recover $d_{0,2}$.

Obviously, this method only needs to read 10 elements, because $d_{1,1}$, $d_{2,0}$, and $d_{3,0}$ are used in recovering two failed elements. Therefore, based on hybrid recovery principle, we can reduce the I/O cost on single failure recoveries.

Hybrid recovery methods. Besides [5], Wang et al. [15] proposed a similar method to minimize recovery I/O cost in EVENODD code [16]. Xu et al. [17] proposed another optimal method for X-Code [18] and have some extend investigate in stack-based recovery. However, this work require the disks to rotate by following the fixed method they proposed and actually this rotated method is rarely used in real systems. Moreover, this method is only applicable for X-Code.

Different from above methods, Khan et al. [7] proposed a general search-based hybrid recovery method for any erasure array code. This method can be simply considered in 3 steps: 1) calculate the set of failed elements F; 2) calculate each $E_{d_{i,j}}$ or $E_{p_{i,j}}$ for each element of F; 3) enumerate all the feasible solutions based on $E_{d_{i,j}}$ and $E_{p_{i,j}}$, and search the optimal solution with the minimal I/O cost. Zhu et al. [19] proposed another general method to find out the approximate solution with polynomial complexity.

On the other hand, Luo and Shu [8] pointed out that the recovery speed is due to the amount of read accesses



(a) Khan's Scheme. (b) Luo's C-Scheme. (c) Luo's U-Scheme.

Fig. 4. An recovery example of 14-disks Blaum-Roth code under different recovery schemes when the first disk failed (The blue squares mean the data elements that require to be accessed, while orange squares indicate the parity elements that need to be read).

on the heaviest loaded disk, and proposed two other general search-based algorithms to generate recovery schemes(C-Scheme and U-Scheme). Similar to Khan's Scheme, C-Scheme searches the solution with the minimal I/O cost, but it has an extra condition that the read accesses on the heaviest loaded disk is the minimal. For example, as Fig. 4 shows, when D_0 failed, the solutions in (a) and (b) satisfy the condition of the minimal I/O cost, but the I/O cost on the heaviest loaded disk (D_1 and D_7) of (b) is also the minimal. Therefore, Khan's Scheme looks up the solution either in (a) or in (b), but C-Scheme finds the solution only in (b).

U-Scheme first searches the solutions with the minimal read accesses on the heaviest loaded disk, and then chooses the solution with the minimal total I/O cost. In Fig. 4, U-Scheme finds out the solution in (c), because the I/O cost on the heaviest loaded disk is the minimal. The experiment results in [8] shows that C-Scheme is faster than Khan's Scheme, while U-Scheme is even faster than C-Scheme.

3 Motivations

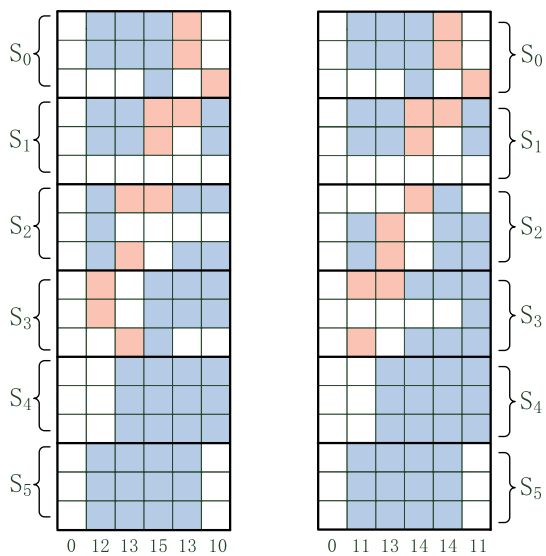
As referred above, most recovery methods are only applicable for a specific erasure code and consider the I/O cost in stripe level. Though Khan's Scheme and Luo's schemes are fit for any erasure array code, both of them pay all of their attention to the recovery only in the stripe level and things will be different when we re-consider this recovery problem in the stack level. In the following, we give our motivations.

Recovery in the stack level. Luo's schemes well balance the load in each stripe. However, if the failed elements in a stack can be reconstructed simultaneously, there is still room for the improvement of recovery speed. For example, let's consider the code in Fig. 2 with the mappings from logic disks to physical disks shown in Fig. 3, and compare the recovery efficiency in the stripe level and in the stack level, respectively.

Fig. 5a shows the recovery schemes derived from either Khan's Scheme or Luo's schemes for all six stripes (they find the same solutions for each stripe). Notice that for the second stripe (S_1), the heaviest loaded disk needs to afford two elements' accesses, while in each of other five stripes the heaviest loaded disk needs to fetch three elements. If we recover every stripe independently, it needs to read two elements in the second stripe and retrieve three elements in each of other five stripes in the heaviest loaded disk. Suppose it needs t seconds to read one element on average, the total time of above read process is $2t + 3t \times 5 = 17t$, because each disk will read elements in parallel. Instead, if we simultaneously recover these six stripes, the heaviest loaded disk just needs to read 15 elements, thus we only need $15t$ seconds to read the needed elements of all 6 rotated stripes in the stack level.

Load balancing among the stack. When comes to stack level, it is usually hard for existing recovery schemes to provide the well balanced I/O distribution, and this weakness will easily degrade the system's reliability and performance. For example, Fig. 5b gives another stack-based recovery scheme, which needs to access the same amount of elements (63 elements) as the scheme shown in Fig. 5a, but owns a lighter load on the busiest disk (only 14 elements' retrieval). This observation motivates us the necessity to design the recovery scheme with balanced I/O distribution.

Search time optimization. Although existing search-based recovery algorithms adapt for most of erasure coded storage systems, they cannot apply in some special storage systems cases such as the system erasure coded by Canghy Reed-Solomon code with large value of k, m and w [19]. Therefore, This observation motivate us not only provide one search-based algorithm for finding out the stack-based recovery



(a) The existing scheme. (b) A potential scheme

Fig. 5. A recovery example in stack-level when the first physical disk failed (S_i means the ith stripe, $0 \leqslant i \leqslant 5$).

schemes with optimal or near-optimal load balancing, but also give another polynomial algorithm to find out the approximate optimal recovery solutions.

Memory optimization. Another important issue is the memory usage when designing stack-based recovery algorithms, because directly reading all the needed elements of a whole stack into the memory seems inapplicable in real systems. For example, suppose a storage system consisting of 16 disks encoded by RDP code, in which the size of each element is $16MB$ (this setting is commonly used in many community [7],[20]) and the rotated mappings follow Fig. 3. Under this configuration, a stripe contains $4\,GB$ information (when $w = 16$) and the stripe-based method needs the memory with the size of $3.75\,GB$. When it comes to stack-level, if we directly access all required elements like stripe-based method, the needed memory will expand to nearly $45\,GB$, which is unacceptable for most of modern storage servers.

Fortunately, each stripe is independently encoded. As long as all the needed elements of a stripe are read, we can recover the lost elements of this stripe, thus don't have to wait until all elements of the stack are read. This observation inspires us that the design of stack-based recovery algorithm should utilize the independence of each stripe, in order to achieve a reasonable memory usage while provide the similar recovery speed as directly accessing all required elements.

4 Stack-Based Recovery Algorithms

In this section, we present the detailed design of our proposed algorithms.

4.1 Balance Priority Scheme

Motivated by the above section which indicates that there actually exists a recovery solution in the stack level to provide balanced I/O, we propose the following BP-Scheme to generate the stack-based recovery scheme based on simulated annealing algorithm, which is a generic probabilistic metaheuristic algorithm inspired by the process of metallurgy annealing to use temperature and energy, in order to find the approximately optimal solution in polynomial time. Compared to other greedy-based algorithms, simulated annealing algorithm is more configurable and has high probability to converge to the optimal solution [11],[26].

The key idea of our BP-Scheme is as follows (we assume a stack contains l stripes and let S_i $(0 \leqslant i \leqslant l-1)$ be the ith stripe of the stack): 1) for each S_i, we enumerate all feasible recovery solutions and find out the alternative solutions with proper I/O cost (like in Khan's Scheme), and store them in a structure RS_i; 2) For each RS_i, we select one solution $s_{i,j}$ as $solution_i$, to compose the initial stack-based solution $Solution$ of simulated annealing algorithm. It can be easily shown that $Solution$ can recover all the lost elements of a stack due to the fact that each S_i can be recovered by $solution_i$. 3) generate $Solution_{new}$ by sequently replacing one $solution_i$ (in $Solution$) with another $s_{i,j}$ of RS_i, and choose whether replace $Solution$ with $Solution_{new}$ based on simulated annealing algorithm. 4) repeat the third step, until the simulated annealing algorithm return a approximate optimal stack-based solution $Solution$. The main steps of the BP-Scheme is shown in Algorithm 1.

Algorithm 1. Balancing Priority Scheme (BP-Scheme)

1: Initial parameter α
2: **for** each S_i $(i \in [0, l-1])$ **do**
3: $E_{all} \leftarrow$ Calculate all recovery equations
4: $minimal_i = int\_max$
5: **for** each $s_{i,j} \leftarrow E_{all}$ **do**
6: **if** $|s_{i,j}| \leqslant minimal_i + \alpha$ **then**
7: $RS_i \leftarrow s_{i,j}$
8: **if** $|s_{i,j}| < minimal_i$ **then**
9: $minimal_i = |s_{i,j}|$
10: **end if**
11: **end if**
12: **end for**
13: **for** each $s_{i,j} \in RS_i$ **do**
14: **if** $|s_{i,j}| > minimal_i + \alpha$ **then**
15: Remove $s_{i,j}$ from RS_i
16: **end if**
17: **end for**
18: **end for**
19:
20: Initial parameter K,T, and M
21: $Solution \leftarrow RS_{all}$
22: $de = cal\_energy(Solution)$
23: $remain\_times = M$
24: **for** $i = 0$ to $l-1$ **do**
25: **for** each $s_{i,j} \in RS_i$ **do**
26: $Solution_{new} = replace\_state(Solution, s_{i,j}, i)$
27: $de_{new} = cal\_energy(Solution_{new})$
28: $\Delta t = de_{new} - de$
29: **if** $\Delta t > 0$ or $exp(\Delta t/T) > rand()/rand\_max$ **then**
30: $Solution = Solution_{new}$
31: $de = de_{new}$
32: $remain\_times = M$
33: **if** $\Delta t < 0$ **then**
34: $T = K * T$
35: **end if**
36: **else**
37: $remain\_times - -$
38: **end if**
39: **end for**
40: **end for**
41: Repeat Steps 24-40 until $remain\_times == 0$
42: Return $Solution$

In Algorithm 1, we first calculate the recovery equations of the failed elements in each stripe based on a similar method as in [14] (see Step 3), and then enumerate all feasible recovery schemes based on E_{all} for each stripe by applying the similar method in [7] (Step 5). After that, we then define α as the threshold value and store all the schemes that need to read $[minimal_i, minimal_i + \alpha]$ elements in RS_i (Step 6-11). Notice that the value of $minimal_i$ will be iteratively updated when the algorithm goes on. When the initiation completes, we re-consider each solution in RS_i, exclude the solution from RS_i if it needs to read more than $minimal_i + \alpha$ elements (Step 13-17), and keep all feasible recovery solution of each stripe that requires $[minimal_i, minimal_i + \alpha]$ in each RS_i.

In the next stage (Step 20-41), we employ the simulated annealing algorithm to obtain the balanced stack-based recovery solution. Specifically, we first initiate three

parameter (K,T,M) that will be used in simulated annealing algorithm (Step 20), where M is the end-condition. We then define the stack-based solution as $Solution$, which is composed of $\{Solution_0, Solution_1, \cdots, Solution_{l-1}\}$ selected from RS_{all}. That is to say, a feasible stripe-based solution $s_{i,j}$ is chosen from RS_i to act as the $Solution_i$. After constructing the primary $Solution$, we calculate its energy and set the end-condition (Step 21-23). Energy is an essential parameter for simulated annealing algorithm. Here, the higher energy means the better load balancing.

The following steps will iteratively optimize the $Solution$ by using the standard simulated annealing algorithm. First, another feasible solution ($Solution_{new}$) will be generated by replacing current $solution_i$ with another $s_{i,j}$ (Step 26), and both its energy and the energy gap will be subsequently calculated (Step 27-28). Based on Δt and T, we determine whether replace $Solution$ with $Solution_{new}$. If the replacement is adopted and the condition $\Delta t < 0$ establishes, we then change T to $K * T$ according to the simulated annealing algorithm (Step 33-34). This chosen process will be repeated until the $remain\_times = 0$ (Step 24-40), i.e., the algorithm can not find another solution after trying M times. Finally, the BP-Scheme returns the near optimal solution $Solution$, which is the balanced stack-based recovery scheme we are pursuing.

4.2 Search Time Priority Scheme

As referred in Section 3, when n, m, and w become large, existing stripe-based recovery method should spend a lot of time for finding the proper recovery scheme, which maybe unacceptable in some cases. Though BP-Scheme utilize the simulated annealing algorithm with polynomial time complexity to optimize the stack-based scheme, it also searches the whole stripe-based solution space and thus cannot run faster than existing recovery methods. Inspired by this observation, we propose the following STP-Scheme for finding the approximately optimal stack-based solution in polynomial time.

The key idea of STP-Scheme is as follows: 1) for each failed element, we calculate all recovery equations for recovering each of them and store these equations in $E_{failed\_elements}$; 2) we generate the initial stack-based solution $Solution$ by randomly select one equation for recovering each failed element; 3) we randomly replace some equations in $Solution$ by other proper equation of $E_{failed\_elements}$ to generate another stack-based solution $Solution_{new}$, and calculate the energy of both $Solution$ and $Solution_{new}$; 4) we use the simulated annealing algorithm to optimize the $Solution$, until the algorithm return the $Solution$ that we pursue. The main steps of the STP-Scheme is given in Algorithm 2.

In Algorithm 2, we first initiate three parameters (K,T, M) for standard simulated annealing algorithm, and initiate one parameter R for controlling the difference between $Solution_{new}$ and $Solution$ (Step 1). The setting and usage of parameters (K,T,M) are the same as those in the second stage of Algorithm 1, while R is a integer number that indicates the number of recovery equations that need to be changed. Afterward, we generate E_{all} and $E_{falied\_element}$ that compose of all recovery equations for each failed element by the similar method as Algorithm 1, and choose one recovery equation for each failed elements to constitute an initial stack-based solution $Solution$ (Step 2-4). Obviously, we can recover all failed elements based on $Solution$, because each failed elements can be recovered by its relative recovery equation.

Algorithm 2. Search Time Priority Scheme

```
1: Initial parameter K,T, R and M
2: E_all ← Calculate all recovery equations
3: E_failed_elements ← E_all
4: Solution ← E_failed_elements
5: de = cal_energy(Solution)
6: remain_times = M
7: while remain_times > 0 do
8:    E_need_replace ← Randomly select R equations in Solution
9:    E_new ← select_equations(E_need_replace, E_failed_elements)
10:   Solution_new = generate_solutions(Solution, E_new)
11:   de_new = cal_energy(Solution_new)
12:   Δt = de_new − de
13:   if Δt > 0 or exp(Δt/T) > rand()/rand_max then
14:      Solution = Solution_new
15:      de = de_new
16:      remain_times = M
17:      if Δt < 0 then
18:         T = K * T
19:      end if
20:   else
21:      remain_times − −
22:   end if
23: end while
24: Return Solution
```

The following steps are used to optimize the $Solution$. First, we randomly select R failed elements and denote their relative recovery equations in $Solution$ as $E_{need\_replace}$, and generate a replacement equation set E_{new} by choosing another recovery equation for each selected failed elements, and then replace each equation of $E_{need\_replace}$ by its relative equation in E_{new}, in order to generate another stack-based solution $Solution_{new}$ (Step 8-10). Then, we calculate the energy gap between $Solution_{new}$ and $Solution$, and utilize the simulated annealing algorithm to determine whether replace $Solution$ with $Solution_{new}$. The afterward steps (step 13-22) are standard simulated annealing algorithm and are the same as Step 29-37 in Algorithm 1, thus we don't explain it again. Finally, the STP-Scheme returns the near optimal solution $Solution$, which is the balanced stack-based recovery scheme that found in polynomial time.

4.3 Rotated Recovery Algorithm

Based on BP-Algorithm and STP-Algorithm, we can find out the near-optimal solution $Solution$ for recovering the lost elements in stack-level. However, as referred in Section 3, directly applying the stack-based schemes (BP-Scheme and STP-Scheme) and simultaneously reading all needed elements in each stack will take up a considerable size of memory, which is usually l times larger than the used memory in the stripe-based recovery algorithm. To address this weakness, we design and implement RR-Algorithm, in order to save memory space while providing the similar recovery speed as simultaneously reading all needed elements.

The key idea of RR-Algorithm is to set a parameter β for denoting the amount of elements that will be read in

each I/O process, and to run two threads (read thread and recovery thread) simultaneously. The read thread retrieve the needed elements all the way, until the rest memory is not enough for each disk read β elements. The recovery thread will recalculate the information of the lost elements of each stripe when all needed elements of this stripe are read, because the encoding of each stripe is independent. The detailed procedures of our algorithm is presented in Algorithm 3.

Algorithm 3. Rotated Recovery Algorithm

1: Initial parameter $read\_sche$, $reco\_sche$, and β
2: $read\_str = 0$
3: $reco\_str = 0$
4: Two threads run synchronously.
5: Read_Thread:
6: **while** $read\_str < stack.num$ **do**
7: $Read\_Elements(data, read\_sche, \beta)$
8: $read\_stripe \leftarrow$ the last $stripe$ that all needed elements of it have been fetched.
9: **end while**
10: Recovery_Thread:
11: **while** $recovery\_stripe < stack.num$ **do**
12: **if** $reco\_str < read\_str$ **then**
13: $Recovery(data, recovery\_stripe)$
14: $FreeMemory(data, recovery\_stripe)$
15: $recovery\_stripe++$
16: **end if**
17: **end while**

In Algorithm 3, we initiate three essential parameters: $read\_sche$, $reco\_sche$ records the needed elements and the XOR relationship to recover each lost element respectively, which are generated by BP-Scheme; β is the number of elements that will be read in each parallel I/O process.

Afterwards, two variants $read\_str$ and $reco\_str$ are defined. $read\_str$ represents the stripe, in which all the needed elements for the recovery has been currently retrieved, while $reco\_str$ records the number of stripes that has finished the recovery. These two variants are set to be zero at the beginning. During the recovery, two threads are simultaneously executed, where the read thread will take the constant number of β elements from the disks into the memory (Step 6-9) and the other thread will repair the lost elements based on the elements in the memory (Step 11-17). The read thread will repeat until all the requested elements are all read (when the memory is not enough, it will wait for recovery thread free memory). The recovery thread will recover lost elements in the stripe $reco\_str$ (Step 13), free the useless memory (Step 14), and increase the value of $reco\_str$ by 1. These two processes will go on until all the lost elements in the stack have been recovered.

5 Analysis of Search Complexity

We now analyze the time complexity of our proposed schemes by comparing with Khan's Scheme and Luo's U-Scheme. The goal of the analysis is to illustrate that the search complexities of BP-Scheme and STP-Scheme are nearly or better than existing stripe-based schemes.

We first suppose that a stack contains l different logic disk failed cases, so that the stripe-based methods (Khan's Scheme and Luo's U-Scheme) need to search l different disk failure patterns. For the ith stripe S_i $(0 \leqslant i \leqslant l-1)$, we assume each of the w failed data elements have $|E_{d_{i,j}}|$ $(0 \leqslant j \leqslant w-1)$ different recovery equations, then the time complexity of both Khan's Scheme and Luo's U-Scheme are $O(\sum_{i=0}^{l} \prod_{j=0}^{w} |E_{d_{i,j}}|)$.

As referred in Algorithm 1, the time complexity of the first stage of BP-Scheme is also $O(\sum_{i=0}^{l} \prod_{j=0}^{w} |E_{d_{i,j}}|)$, because both the first stage of BP-Scheme and Khan's Scheme need to enumerating all feasible stripe-based solutions, and calculating each feasible solution's I/O cost. Though Luo's U-Scheme has the same time complexity with Khan's Scheme and the first stage of BP-Scheme, it usually needs more running time due to the fact that calculating the I/O cost on each disk is much slower than calculating the total I/O cost.

On the other hand, since simulated annealing algorithm is polynomial time, the second stage of BP-Scheme and STP-Scheme are polynomial time as well, because they are just different applications of simulation annealing algorithm. According to Algorithm 1 and Algorithm 2, the time complexity of both STP-Scheme and the second stage of BP-Scheme should be polynomial with l, w, and m. Compared to the first stage of BP-Scheme, the complexity of the second stage of BP-Scheme is negligible. Therefore, the time complexity of BP-Scheme is $O(\sum_{i=0}^{l} \prod_{j=0}^{w} |E_{d_{i,j}}|)$, which is the same as Khan's Scheme and Luo's U-Scheme. STP-Scheme can be completed in polynomial time.

In addition, we evaluate the real search time of above schemes in Section 7.4, in order to verify our analysis and examine our proposed schemes' efficiency.

6 Analysis of Recovery Speed

In this section, we analyze the recovery speed of different schemes theoretically. The goal of our analysis is to illustrate that our proposed BP-Scheme and STP-Scheme not only provide much higher recovery speed than Khan's Scheme and Luo's schemes, but also achieve the approximately optimal recovery speed in the stack-level. We select the scenario that a stack contains n different stripes shown in Fig. 3 for comparison, because this scenario is commonly used in erasure code community [7],[9],[10].

6.1 The Analysis Metrics

We analyze the recovery speed of different schemes by defining *recovery factor* and its *lower bound*.

Recovery Factor. We define *recovery factor* as the total amount of parallel read accesses divided by the total number of elements of each disk in a stack (i.e., $n \times w$), where one parallel read access means each disk parallel reads one element. For example, we consider the example in Fig. 5a: if we read one stripe into memory in one round, it needs two parallel read accesses in the second stripe and three parallel read accesses in other stripes, thus the recovery factor is $(3+2+3+3+3+3)/18 = 0.944$; if we simultaneous read all n stripes into memory in one round, the recovery factor is $15/18 = 0.833$.

We use $max\_read_a^b$ to represent the number of elements that need to be read on the heaviest disk among S_a to S_b, and use u to denote the smaller number between $ir + r - 1$

and $n-1$. For a certain recovery scheme, if we simultaneously read r stripes into memory in one round, the recovery factor will be calculated by the following equation.

$$recovery\_factor = \frac{\sum_{i=0}^{\frac{n}{r}-1} max\_read_{ir}^{u}}{nw} \quad (1)$$

Specifically, we use max_read_i to represent the amount of elements that need to be read on the heaviest disk of stripe S_i $(0 \leqslant i \leqslant n-1)$, and use max_read_{all} to denote the number of required elements on the heaviest disk of a whole stack. Under these definitions, if we read one stripe into memory in one round (i.e., $r=1$), the recovery factor can be calculated by the following equation.

$$recovery\_factor = \frac{\sum_{i=0}^{n-1} max\_read_i}{nw}. \quad (2)$$

However, if we simultaneously read all n stripes into memory (i.e., $r=n$) in one round, the recovery factor should be computed as follows.

$$recovery\_factor = \frac{max\_read_{all}}{nw}. \quad (3)$$

It is easy to deduce that recovering one failed element needs $recovery\_factor$ parallel read accesses on average. Therefore, the recovery speed should be linear with $\frac{1}{recovery\_factor}$, because both recalculating broken elements and rewriting to the new disk can be processed in parallel with the reading process [19].

Lower Bound. We deduce the lower bound of recovery factor by the following equation.

$$Lower\_Bound = \frac{\lceil \sum_{i=0}^{n-1} minimal_i/(n-1) \rceil}{nw}. \quad (4)$$

In Equation (4), $\sum_{i=0}^{n-1} minimal_i$ is the minimal number of elements that requires in a stack, while $n-1$ presents the survived $n-1$ physical disks. According to pigeonhole principle, at least one physical disk needs to read $\lceil \sum_{i=0}^{n-1} minimal_i/(n-1) \rceil$ elements, thus both the $\sum_{i=0}^{n-1} max\_read_i$ and max_read_{all} must no less than $\lceil \sum_{i=0}^{n-1} minimal_i/(n-1) \rceil$. Therefore, the recovery factor over any erasure codes must no less than $\frac{\lceil \sum_{i=0}^{n-1} minimal_i/(n-1) \rceil}{nw}$. For example, in the case of Fig. 5a, the lower bound is $\frac{\lceil (10+9+10+10+12+12)/5 \rceil}{6\times 3} = 0.7$.

6.2 The Analysis Methodology

We select Khan's scheme [7] and Luo's U-Scheme [8] as comparison, because these schemes can adapt for any erasure array codes and other schemes like in [5] can be found by either of these schemes. The analyzed erasure codes and parameters are shown in Table 2.

For Khan's Scheme and Luo's U-Scheme, we follow the original paper's recovery process and calculate the recovery factor by Equation (2) (i.e., $r=1$). For BP-Scheme and STP-Scheme, we select parameter $T=1$, $K=0.999$, $\alpha = \lceil minimal_i/100 \rceil$, $R=2$, and compute the recovery factor based on Equation (3) (i.e., $r=n$), because they are implemented by RR-Algorithm which assures the memory space is enough across the stack-based recovery process. In order to clarify the gap between these schemes, all our results normalize the value of lower bound as **one**.

TABLE 2
The Erasure Codes and Paraments we Tested
(For each k, We Select the Smallest w if They are Matched)

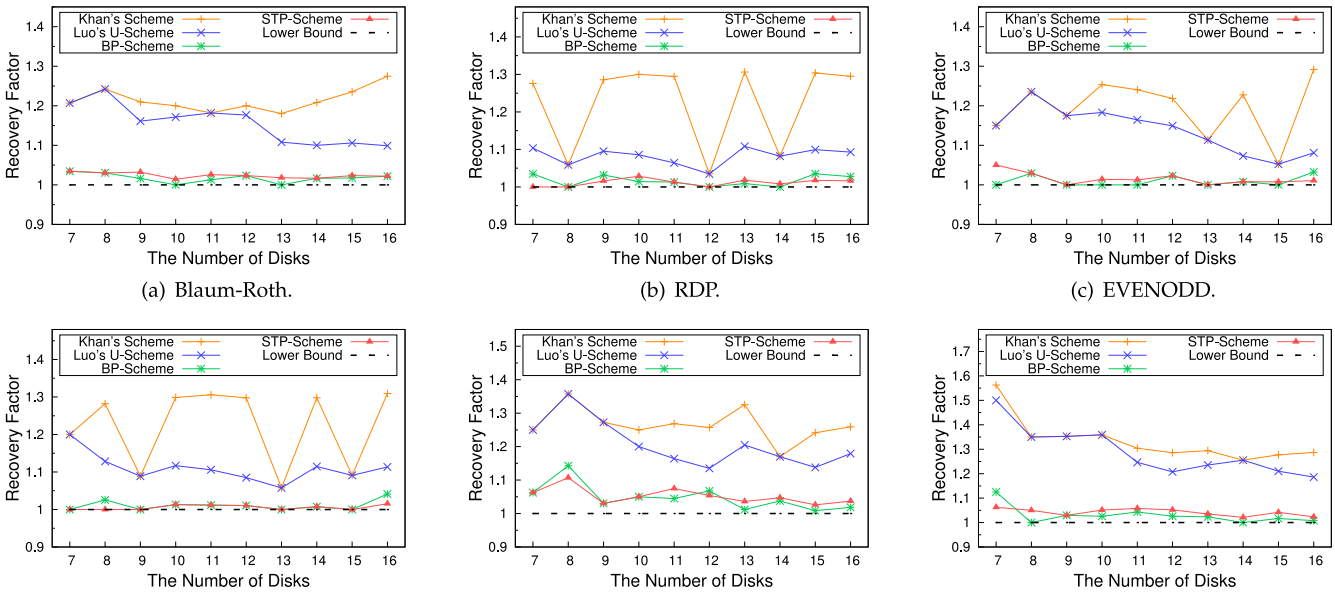
Code	k	w	Restrictions
Cauchy-RS (m = 2)	5-14	8	None
Blaum-Roth (m = 2)	5-14	6-16	$w+1$ prime $> k$
RDP (m = 2)	5-14	6-16	$w+1$ prime $> k$
EVENODD (m = 2)	5-14	4-16	$w+1$ prime $\geqslant k$
Liberation (m = 2)	5-14	7-17	w prime $\geqslant k$
Cauchy-RS (m = 3)	4-13	8	None
Generalized RDP (m = 3)	4-13	4-16	$w+1$ prime $> k$
STAR (m = 3)	4-13	4-12	$w+1$ prime $\geqslant k$

6.3 The Analysis Results

We now analyze the recovery speed of different recovery schemes by the methodology referred above, and show the normalized recovery factor over different erasure codes in Fig. 6. As the figure shows, BP-Scheme and STP-Scheme provides much lower recovery factor than Khan's Scheme and Luo's U-Scheme, and very close to the lower bound. Specifically, for RAID-6 codes (Bluam-Roth code RDP code, EVENODD code, Liberation Code), the recovery factor of BP-Scheme and STP-Scheme is 3.4 to 22.8 percent lower than Khan's Scheme, 3.4 to 17.1 percent lower than Luo's U-Scheme, and just 0 to 4.1 percent or 0 to 5.0 percent higher than the lower bound. Since the recovery speed should be linear with $\frac{1}{recovery\_factor}$, the recovery speed of both BP-Scheme and STP-Scheme will be 3.5 to 29.5 percent higher than Khan's Scheme, 3.5 to 20.6 percent higher than Luo's U-Scheme, and only 0 to 3.5 percent or 0 to 5.3 percent lower than the optimal recovery speed, respectively.

For $m=3$ codes (Generalized RDP code and STAR code), BP-Scheme and STP-Scheme achieves 0.9 to 14.3 percent and 2.6 to 10.7 percent higher recovery factor than the lower bound respectively, because the recoveries from parity disk failures cause intensive I/O in the adjacent physical disks of the failed physical disk, which is hard to balance when m equals to 3. Compared with Khan's Scheme, BP-Scheme and STP-Scheme gains 11.3 to 28.0 percent and 10.5 to 32.0 percent lower recovery factor; compared with Luo's U-Scheme, BP-Scheme and STP-Scheme achieves 6.0 to 25.9 percent and 7.1 to 29.2 percent lower recovery factor, respectively. All these results indicate that both BP-Scheme and STP-Scheme will provide much higher recovery speed than other recovery schemes.

We now compare the proposed STP-Scheme with BP-Scheme: STP-Scheme provides -3.3 to 5.0 percent higher recovery factor than BP-Scheme when $m=2$, while gains -5.6 to 5.0 percent higher recovery factor when $m=3$, because 1) both BP-Scheme and STP-Scheme utilize the polynomial algorithm and don't search the whole solution space, thus cannot certainly reach the optimal solution; 2) STP-Scheme speeds a little more time to find out an approximately optimal solution than BP-Scheme when n is small, but spends much less time than BP-Scheme when n is large; 3) compared to STP-Scheme, BP-Scheme has one limited



(a) Blaum-Roth. (b) RDP. (c) EVENODD.

(d) Liberation. (e) Generalized RDP. (f) STAR.

Fig. 6. The normalized recovery factor over different erasure codes.

condition that α equals to $\lceil minimal_i/100 \rceil$ which assures the low I/O cost but limits its balancing in some level. Combine with these factors, the recovery factor of BP-Scheme and STP-Scheme should be close. The analysis results also confirm this concern.

In addition, since the lower bound is calculate by pigeonhole principle, it is probably that the optimal solution also can not achieve the lower bound. Therefore, combined with above analysis, BP-Scheme will achieve very good speed for single disk failure recovery.

7 Experiment Evaluations

We build a number of experiments to evaluate the efficiency of single disk failure recovery with different recovery schemes. We implement Khan's Scheme, Luo's U-Scheme, BP-Scheme and STP-Scheme by C++ language, and implement all erasure codes of Table 2in real storage nodes based on Jerasure-1.2 library [12], which is an open source library and commonly used in erasure code community [10]. The mappings from logic disks to physical disks in our storage system follow Fig. 3, which is a commonly used choice in erasure code community [7], [9], [10]. To eliminate the influence of parallelism, we implement Khan's Scheme and Luo's U-scheme by a parallel recovery method like in [19], i.e., the read process and the recovery process are running concurrently.

7.1 Experimental Setup

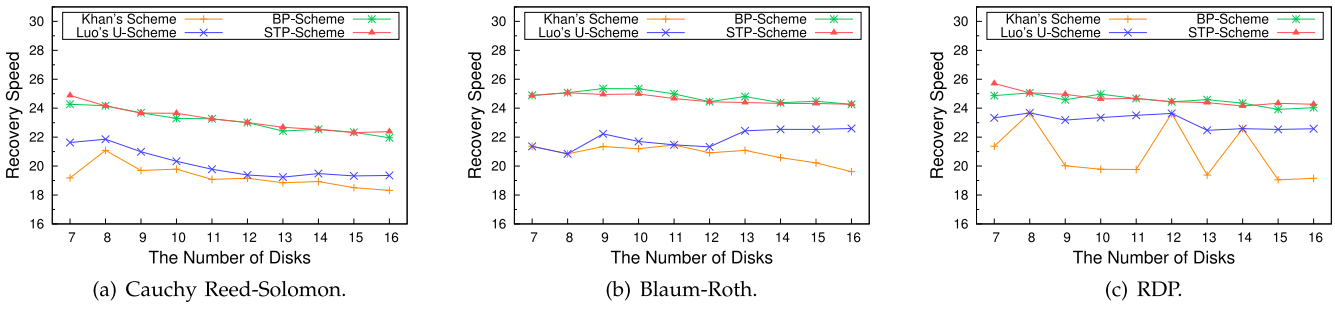
Our experiments are run on a machine and a disk array. The hardware environment of the machine is an Intel Xeon X5472 processor and 12 GB memory. The disk array contains 16 Seagate/Savvio 10K.3 SAS disks, where the model number is ST 9300603SS. Each disk has 300 GB capability and 10,000 rpm. The machine and disk array are connected by a fiber cable with 800 MB bandwidth. The operation system of the machine is SUSE with Linux fs91 3.2.16. Our experiments set each element with a size of 16 MB, which is a typical choice in storage systems [7],[20]. All tests mirror the configurations in Table 2, evaluating a series of erasure codes with $k = 7$ to 16 and $m = 2, 3$. For each k, we choose the smallest w of the table if they are permitted. In addition, we select the parameter $\beta = w$ for RR-Algorithm.

7.2 Evaluation of Recovery Speed

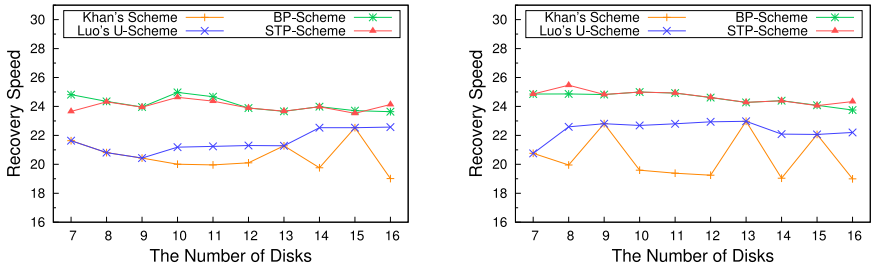
We evaluate the recovery speed of our proposed schemes by comparing with Khan's Scheme and Luo's U-Scheme. The goal of these experiments is to verify our theoretical analysis and to demonstrate that BP-Scheme and STP-Scheme gains higher recovery speed than these existing schemes. For each erasure code, we consider 20 stacks and each stack contains n stripes, thus we have to recover up to 87.04GB data from the failed physical disk.

The recovery speed over different erasure codes with $\boldsymbol{m = 2}$. Fig. 7 shows the recovery speed for various $m = 2$ erasure codes with different schemes and illustrates that BP-Scheme and STP-Scheme provide much higher recovery speed than Khan's Scheme and Luo's U-Scheme. In statistics, for Cauchy Reed-Solomon code, BP-Scheme gains 14.6 to 26.5 percent higher recovery speed than Khan's Scheme and 10.6 to 18.7 percent higher speed than Luo's U-Scheme, and STP-Scheme achieves 14.6 percent to 26.5 percent higher speed than Khan's Scheme and 10.6 to 18.7 percent higher speed than Luo's U-Scheme; for Blaum-Roth code, BP-Scheme gains 16.4 to 23.7 percent higher recovery speed than Khan's Scheme and 7.4 to 20.3 percent higher speed than Luo's U-Scheme, while STP-Scheme gains 14.9 to 23.8 percent higher recovery speed than Khan's Scheme and owns 7.4 to 20.3 percent higher speed than Luo's U-Scheme. In addition, STP-Scheme achieves polynomial time complexity provides a little higher recovery speed than BP-Scheme, it suffers from a little higher recovery cost, because BP-Scheme uses parameter α to ensure the low I/O cost while STP-Scheme has no restriction on I/O cost.

The improvement in different codes are quite different due to the fact that each code has its own properties. For



(a) Cauchy Reed-Solomon. (b) Blaum-Roth. (c) RDP.



(d) EVENODD. (e) Liberation.

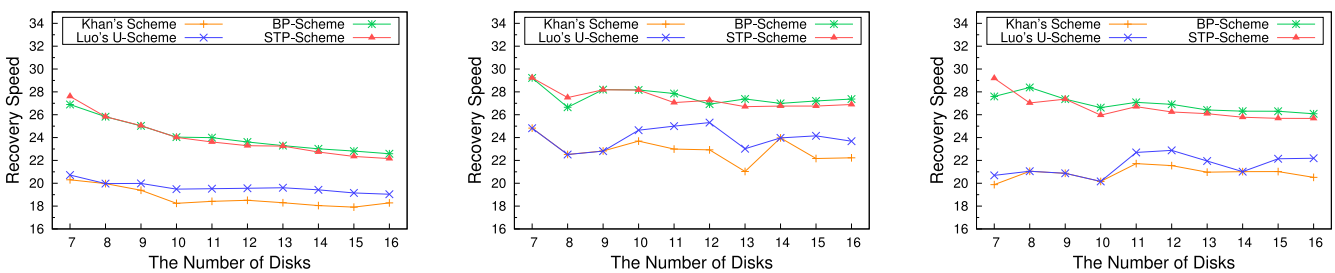
Fig. 7. The recovery speed over different erasure codes with $m = 2$ (Unit: MB/s).

RDP code, Xiang et al. in [5] has proved that it can achieve well balancing in each stripe under the standard forms (i.e., $n = 8, 12, 14$), while other forms also have well balanced stripe-based solutions that can be found by Luo's U-Scheme. On the other hand, the generating of Cauchy Reed-Solomon code is not well facilitated for load balancing in each stripe, thus the stripe-based method can not provide well recovery speed.

In summary, BP-Scheme provides 3.4 to 28.6 percent higher recovery speed than Khan's Scheme and achieves 3.4 to 20.3 percent higher speed than Luo's U-Scheme, while STP-Scheme gains 3.4 to 29.7 percent higher speed than Khan's Scheme and owns 3.4 to 20.3 percent higher speed than Luo's U-Scheme. Notice that our analysis results indicate that BP-Scheme will provide 3.5 to 29.5 percent higher speed than Khan's Scheme in $m = 2$ codes, but the actual improvement is 3.4 to 28.6 percent. This is because our analysis assumes that the XOR computation operations and read operations are completely running in parallel. But in real systems, in order to save memory usage, the parallelism of RR-Algorithm may be a little worse than that of parallelly stripe-based recovery method. Overall, the experiment results match the theoretical analysis closely and indicate that the recovery speed of BP-Scheme and STP-Scheme is much higher than other schemes over $m = 2$ codes.

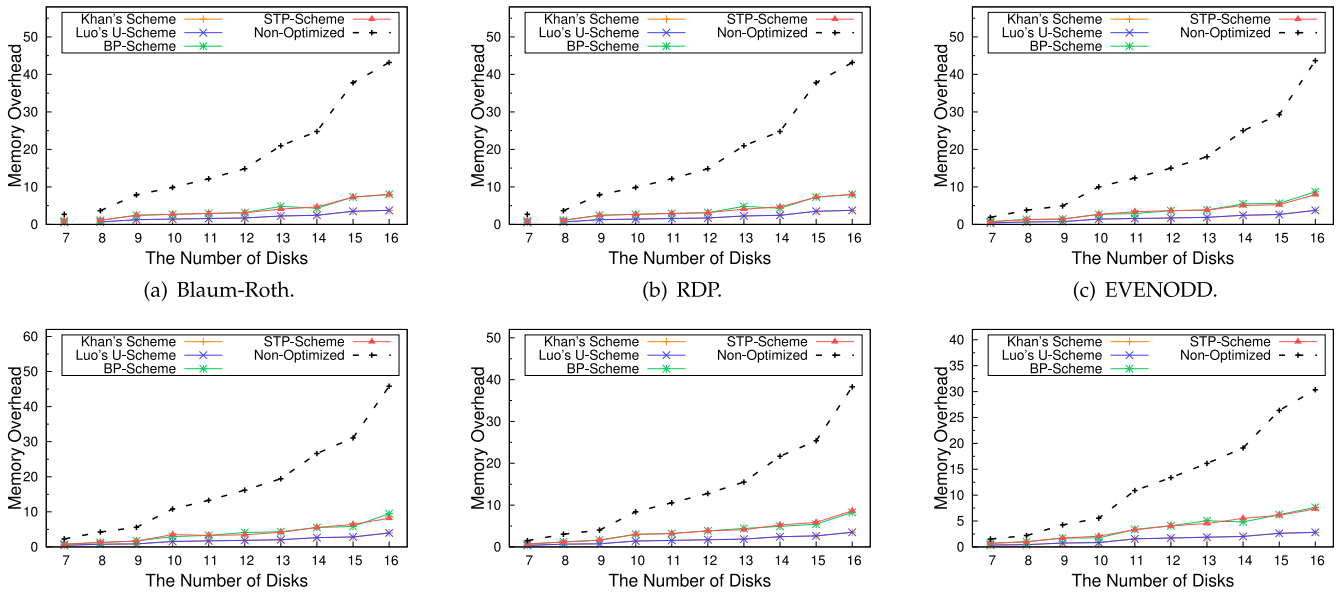
The Recovery Speed over Different Erasure Codes with $m = 3$. We now consider the erasure codes with $m = 3$, and show the experiments result in Fig. 8. As the figure shows, for Cauchy Reed-Solomon code, BP-Scheme gains 23.5 to 32.5 percent higher speed than Khan's Scheme and 18.5 to 29.8 percent higher speed than Luo's scheme, and STP-Scheme provides 21.2 to 36.0 percent higher speed than Khan's Scheme and 16.4 to 33.2 percent higher speed than Luo's U-Scheme; for STAR code, BP-Scheme achieves 24.7 to 38.9 percent higher speed than Khan's Scheme and 17.5 to 34.8 percent higher speed than Luo's Scheme, while STP-Scheme owns 21.8 to 46.9 percent and 14.7 to 41.1 percent higher speed than Khan's Scheme and Luo's U-Scheme respectively, which illustrate that BP-Scheme runs much faster than Khan's Scheme and Luo's U-Scheme and STP-Scheme runs a little faster than BP-Scheme with a trade-off on I/O cost.

The amount of disks is another important factor in $m = 3$ codes, because each code with distinct amount of disks will provide different balancing. For example, in 12-disks Generalized RDP code, the improvement between BP-Scheme and Luo's Scheme is only 6.4 percent, because this code provides an well balanced stripe-based solution that can be found by Luo's U-Scheme when $n = 12$. In other cases of n, it does not have well balanced stripe-based solution and thus we can improve the recovery speed by balancing the I/O among the



(a) Cauchy Reed-Solomon. (b) Generalized RDP. (c) STAR.

Fig. 8. The recovery speed over different erasure codes with $m = 3$ (Unit: MB/s).



(a) Blaum-Roth. (b) RDP. (c) EVENODD.

(d) Liberation. (e) Generalized RDP. (f) STAR.

Fig. 9. The memory overhead of different recovery schemes (Unit: GB).

stripes of each stack. On the other hand, the improvement in $m = 3$ codes is much higher than that in $m = 2$ codes, because $m = 3$ codes have two parity disk failed cases. When suffering from parity disk failures, the number of elements that requires on heaviest loaded disk in Khan's Scheme and Luo's U-Scheme is w, which is not well balanced. However, BP-Scheme and STP-Scheme can balance them in other corresponding stripes, thus provide much higher speed.

In summary, for $m = 3$ codes, BP-Scheme provides 12.6 to 38.9 percent higher recovery speed than Khan's Scheme and 6.4 to 34.8 percent higher recovery speed than Luo's U-Scheme, while STP-Scheme gains 11.6 to 46.9 percent and 7.7 to 41.1 percent higher speed than Khan's Scheme and Luo's U-Scheme, respectively. The results match the analysis closely, and illustrate that our proposed schemes provide much better performance than other schemes.

7.3 Evaluation of Memory Overhead

We now evaluate the memory overhead of BP-Scheme and STP-Scheme in different erasure codes. The goal of these experiments is to illustrate that the memory overhead of our proposed schemes is acceptable. We evaluate the memory overhead of Khan's Scheme, Luo's U-Scheme, BP-Scheme with RR-Algorithm, STP-Scheme with RR-Algorithm in real system, and calculate the memory overhead of non-optimized STP-Scheme by simulations.

Fig. 9 illustrates the real memory overhead of these schemes. As the figure shows, the memory overheads of BP-Scheme and STP-Scheme after optimized are close, because these schemes are both stack-based. On the other hand, BP-Scheme and STP-Scheme (after optimized) need to utilize 1-2 times more memory space than Khan's Scheme and Luo's U-Scheme, because the stack-based recovery methods need to simultaneously consider multiple stripes rather than only one stripe, thus need more memory space. Fortunately, even when $n = 16$, the maximum memory overhead of both BP-Scheme and STP-Scheme (with RR-Algorithm) in our experiments is no more than $10\,G$, which is acceptable in state-of-the-art servers.

7.4 Evaluation of Search Time

We evaluate our proposed schemes over Blaum-Roth code, RDP code, EVENODD code, Liberation code, Generalized RDP code and STAR code with 16 disks and Cauchy-RS code with 20 disks, and record the real search time of these schemes. The evaluation machine has four cores and 12 GB RAM, where each core has 3.0 GHz dominant frequency and 12 MB L2-Cache.

Table 3 gives the main configurations of the tested erasure codes and the search time of different recovery schemes. As the table shows, the time overhead of BP-Scheme is just 0-9 seconds higher than Khan's Scheme, which indicates that the time complexity of BP-Scheme and Khan's Scheme are close. STP-Scheme only executes 4-15 seconds over all tested configurations, because it has polynomial complexity. Luo's U-Scheme runs slower than other schemes, because calculating the I/O cost on the heaviest loaded disk is more complex than calculating the total I/O cost. Furthermore, though all tested schemes can find out the optimal or satisfied solution over most configurations when $n = 16$, when comes to Cauchy-RS code with 20-disks, Khan's Scheme, Luo's U-Scheme and our proposed BP-Scheme cannot find out a satisfied solution in 24 hours. In summary, the search time of BP-Scheme is very close to Khan's scheme and is acceptable for a majority of erasure codes with reasonable configurations, while STP-Scheme's search time is just several seconds over all tested cases.

8 Conclusions

In this paper, we have proposed two recovery generation schemes: BP-Scheme and STP-Scheme, in order to improve the speed on single disk failure recovery in the stack-level. BP-Scheme is a search-based algorithm, which first enumerates all the feasible solutions of each stripe to seek for a set of alternative stripe-based solutions, and then finds out a

TABLE 3
The Time Overhead of Different Recovery Schemes over Different Erasure Codes

Erasure Code	k	w	Khan's Scheme	Luo's U-Scheme	BP-Scheme	STP-Scheme
Blaum-Roth (m = 2)	14	16	9s	11s	12s	4s
RDP (m = 2)	14	16	42s	66s	44s	4s
EVENODD (m = 2)	14	16	682s	653s	682s	5s
Liberation (m = 2)	14	17	16s	31s	19s	3s
STAR (m = 3)	13	16	257s	379s	260s	5s
Generalized RDP (m = 3)	13	12	3060s	6230s	3069s	5s
Caughy-RS (m = 4)	16	10	>24h	>24h	>24h	15s
Cauchy-RS (m = 5)	15	10	>24h	>24h	>24h	15s

near-optimal stack-based recovery scheme based on simulated annealing algorithm. STP-Scheme utilizes simulated annealing algorithm directly, in order to find out the near-optimal stack-based recovery scheme in polynomial time. Furthermore, we also proposed a RR-Algorithm for the realization of BP-Scheme and STP-Scheme to reduce the memory overhead.

We conduct a series of rigorous statistic analysis and intensive evaluations on a real system, in order to evaluate the recovery speed of our proposed schemes on both theory and practice. The analysis and experiment results show that, BP-Scheme and STP-Scheme not only gain much higher recovery speed than other existing schemes, but also provide an acceptable memory overhead. Specifically, BP-Scheme provides 3.4 to 38.9 percent (the average is 21.2 percent) higher recovery speed than Khan's Scheme and 3.4 to 34.8 percent (the average is 19.1 percent) higher speed than Luo's U-Scheme, while STP-Scheme gains 3.4 to 46.9 percent (the average is 25.15 percent) higher recovery speed than Khan's Scheme and 3.4 to 41.1 percent (the average is 22.3 percent) higher speed than Luo's U-Scheme over the tested erasure codes.

Acknowledgments

The authors would like to greatly appreciate the anonymous reviewers for their insightful comments. This work was supported by the National Natural Science Foundation of China (Grant No. 61232003, 61327902), the National High Technology Research and Development Program of China (Grant No. 2013AA013201), and the State Key Laboratory of High-end Server and Storage Technology (Grant No. 2014HSSA02). A 10-page conference version of this paper appeared in Proceedings of the 33th IEEE Symposium on Reliable Distributed Systems (SRDS), October 2014 [27]. In this journal version, we include another stack-based recovery algorithm with polynomial complexity on CORE. The corresponding author is J. Shu.

References

[1] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *Proc. 19th ACM Symp. Operating Syst. Principles*, 2003, pp. 29–43.

[2] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. Mckelvie, Y. Xu, S, Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. Fahim-ul-Haq, M. Ikram-ul-Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas, "Windows azure system: A highly available cloud storage service with strong consistency," in *Proc. 23rd ACM Symp. Operating Syst. Principles*, 2011, pp. 143–157.

[3] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, W. Weimer, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," in *Proc. ACM 9th Int. Conf. Archit. Support Programm. Lang. Oper. Syst.*, 2000, pp. 190–201.

[4] E. Pinheiro, W. Weber, and L. Barroso, "Failure trends in a large disk drive population," in *Proc. 5th USENIX Conf. File Storage Technol.*, 2007, p. 5.

[5] L. Xiang, Y. Xu, J. Lui, and Q. Chang, "Optimal recovery of single disk failures in RDP code storage systems," in *Proc. 12th ACM SIGMETRICS/PERFORMANCE Joint Int. Conf. Meas. Model. Comput. Syst.*, 2010, pp. 119–130.

[6] 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. 3rd USENIX Conf. File Storage Technol.*, San Francisco, CA, USA, Mar. 2004, p. 1.

[7] O. Khan, R. Burns, J. Plank, and W. Pierce, "Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads," in *Proc. 10th USENIX Conf. File Storage Technol.*, 2012, p. 20.

[8] X. Luo, and J. Shu, "Load-balanced recovery schemes for single-disk failure in storage systems with any erasure code," in *Proc. 42nd IEEE Int. Conf. Parallel Process.*, Oct. 2013, pp. 552–561.

[9] J. Plank, "The RAID-6 liberation codes," in *Proc. USENIX Conf. File Storage Technol.*, Feb. 2008, p. 7.

[10] J. Plank, J. Luo, C. Schuman, L. Xu, and Z. W. O'Hearn, "A performance evaluation and examination of open-source erasure coding libraries for storage," in *Proc. 7th USENIX Conf. File Storage Technol.*, Feb. 2009, pp. 253–265.

[11] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[12] J. Plank, S. Simmerman, and C. Schuman, "Jerasure: A library in C/C++ facilitating erasure coding for storage applications-Version 1.2," Dept. Comput. Sci., Univ. Tennessee, Knoxville, TN, USA, Tech. Rep. CS-08-627, 2008.

[13] J. Hafner, V. Deenadhayalan, T. Kanungo, and K. Rao, "Performance metrics for erasure codes in storage systems," IBM Res., Yorktown Heights, NY, USA, Tech. Rep. RJ 10321, 2004.

[14] K. Greenan, X. Li, and J. Wylie, "Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs," in *Proc. 26th Symp. IEEE Mass Storage Syst. Technol.*, 2010, pp. 1–14.

[15] Z. Wang, A. Dimakis, and J. Bruck, "Rebuilding for array codes in distributed storage systems," in *Proc. IEEE GLOBECOM Workshops*, 2010, pp. 1905–1909.

[16] M. Blaum, J. Bruck, and J. Nebib, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Trans. Inf. Theory*, vol. 45, no. 1, pp. 46–59, Jun. 1999.

[17] S. Xu, R. Li, P. Lee, Y. Zhu, L. Xiang, Y. Xu, and J. Lui, "Single disk failure recovery for X-code-based parallel storage systems," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 995–1007, Jan. 2013.

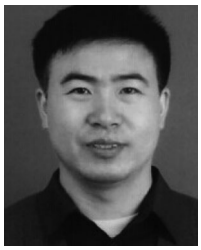
[18] L. Xu, and J. Bruck, "X-Code: MDS array codes with optimal encoding," *IEEE Trans. Inf. Theory*, vol. 45, no. 1, pp. 272–276, Jan. 1999.

[19] Y.Zhu, P. 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. 28th IEEE Mass Storage Syst. Technol.*, 2012, pp. 1–12.

[20] J. Schindler, S. W. Schlosser, M. Shao, A. Ailamaki, and G. R. Ganger, "Atropos: A disk array volume manager for orchestrated use of disks," in *Proc. USENIX Conf. File Storage Technol.*, Mar. 2004, pp. 159–172.

[21] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. New York, NY, USA: North-Holland, 1977.
[22] M. Blaum, and R. Roth, "On lowest density MDS codes," *IEEE Trans. Inf. Theory*, vol. 45, no. 1, pp. 46–59, Jan. 1999.
[23] M. Blaum, "A family of MDS array codes with minimal number of encoding operations," in *Proc. IEEE Int. Symp. Inf. Theory*, Sep. 2006, pp. 2784–2788.
[24] C. Huang and L. Xu, "STAR: An efficient coding scheme for correcting triple storage node failures," in *Proc. USENIX Conf. File Storage Technol.*, 2005, p. 15.
[25] J. Blomer, M. Kalfane, R. Krap, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-based Erasure-resilient coding scheme," Int. Comput. Sci. Inst., Berkeley, CA, USA, Tech. Rep. TR-95-048, Aug. 1995.
[26] Simulated Annealing [Online]. Available: http://en.wikipedia.org/wiki/Simulated_annealing, May 2015.
[27] Y. Fu, J. Shu, and X. Luo, "A stack-based single disk failure recovery scheme for erasure coded storage systems," in *Proc. IEEE 33rd Int. Symp. Rel. Distrib. Syst.*, Oct. 2014, pp. 136–145.

Yingxun Fu received the bachelor's degree from North China Electric Power University in 2007, the master's degree from the Beijing University of Posts and Telecommunications in 2010, and the doctor's degree from Tsinghua University in 2015. His current research interests include storage reliability and distributed systems.



Jiwu Shu received the PhD degree in computer science from Nanjing University in 1998, and finished the postdoctoral position research at Tsinghua University in 2000. Since then, he has been teaching at Tsinghua University. His current research interests include storage security and reliability, non-volatile memory based storage systems, and parallel and distributed computing. He is a member of the IEEE.

Zhirong Shen receives the bachelor's degree from the University of Electronic Science and Technology, China. He is currently working toward the PhD degree with the Department of Computer Science and Technology, Tsinghua University. His current research interests include storage reliability and storage security.

Guangyan Zhang received the bachelor's and master's degrees in computer science from Jilin University, in 2000 and 2003, the doctor's degree in computer science and technology from Tsinghua University in 2008. He is currently an associate professor in the Department of Computer Science and Technology, Tsinghua University. His current research interests include big data computing, network storage, and distributed systems.