2014 IEEE 33rd International Symposium on Reliable Distributed Systems

A Stack-Based Single Disk Failure Recovery Scheme for Erasure Coded Storage Systems

Yingxun Fu, Jiwu Shu*, and Xianghong Luo
Department of Computer Science and Technology
Tsinghua University
Beijing, China
*Corresponding author: shujw@tsinghua.edu.cn
fu-yx10@mails.tsinghua.edu.cn, luo-xh09@mails.tsinghua.edu.cn

***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 in this literature 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 our recovery mechanism based on greedy algorithm to seek for the near-optimal solution (BP-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 at most 38.9% higher recovery speed than Khan's Scheme, and owns up to 34.8% higher recovery speed than Luo's U-Scheme.

***Keywords*-single failure recovery; erasure code; stack; storage system;**

I. 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 symbols and once a disk failure happens, the storage system will retrieve a subset of available data symbols and the pre-stored parity symbols 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%) 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.

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 symbols and a considerable amount of parity symbols. However, things change when Xiang et al. [5] firstly 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 symbols, so that at most 25% symbols 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 symbols for any erasure array code. Inspired by the fact that the reconstruction time is determined by the disk loaded with the heaviest requests, Luo et al. [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 II-A). Though Luo's schemes well speed up the process of single disk failure recovery in the stripe-level, it is still unknown what 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 a new search-based algorithm called Balance-Prior Scheme (BP-Scheme). BP-Scheme first collects the feasible recovery solutions for each stripe by enumerating all feasible

1060-9857/14 $31.00 © 2014 IEEE
DOI 10.1109/SRDS.2014.29

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. Finally, a near-optimal solution with the minimal data accesses on the heaviest disk could be found in the stack-level.

Moreover, recovery based on stack-level will cause huge memory overheads, in order to eliminate the memory requirement for BP-Scheme realization, we then develop a Rotated Recovery Algorithm (RR-Algorithm), which iteratively reads a constant number of symbols during each parallel I/O process and invokes the reconstruction of the lost symbols of a stripe once the needed symbols in that stripe are all retrieved. Our contributions can be summarized as follows:

- We consider the single disk failure recovery in the stack-level and propose a search-based scheme (BP-Scheme) to seek for the near-optimal solution with the minimal retrieved symbols on the heaviest loaded disk. To reduce the memory overhead, we subsequently provide a Rotated Recovery Algorithm (RR-Algorithm).
- We make a series of quantity analysis on various kinds of erasure array codes. The results demonstrate that our BP-Scheme outperforms other competitors on the metric of recovery speed. On the other hand, the results indicate that BP-Scheme will provide the near-optimal recovery speed in the stack-level.
- We finally implement our proposed algorithms in the real storage system based on Jerasure-1.2 [12] library and evaluate their performance with different configurations. The results show that BP-Scheme provides 3.4% to 28.6% higher recovery speed than Khan's Scheme, and 3.4% to 20.3% higher than Luo's U-Scheme in double fault-tolerance erasure codes; while achieves 12.6% to 38.9% higher recovery speed than Khan's Scheme, and 6.4% to 34.8% higher than Luo's U-Scheme in triple fault-tolerance codes.

II. Background and Related Work

A. 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, $\cdots$, D_{k-1}) that keep original data, and $m = n - k$ logic disks (denoted as P_0, P_1, $\cdots$, 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 symbols, which are fixed-size units of data or parity information and

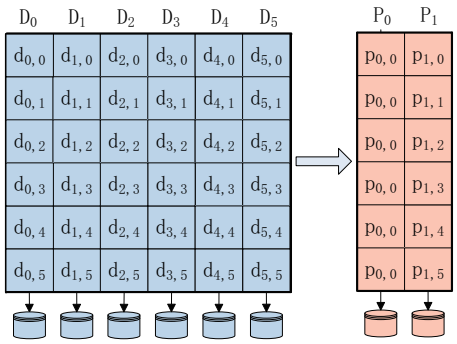


Figure 1: An example of one stripe in erasure coded storage systems, where $\boldsymbol{n = 8}$, $\boldsymbol{k = 6}$, $\boldsymbol{m = 2}$, and $\boldsymbol{w = 6}$.

the number is denoted as w. We label the w symbols on the ith data disk as $d_{i,0}$, $d_{i,1}$, $\cdots$, $d_{i,w-1}$, and on the ith parity disk as $p_{i,0}$, $p_{i,1}$, $\cdots$, $p_{i,w-1}$. For example, Figure 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 Caughy Reed-Solomon code with $n = 6$, $k = 4$, $m = 2$, and $w = 3$ in Figure 2. As the figure shows, the kr data symbols 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 information, i.e., computing the information of P_0 to P_{m-1}.

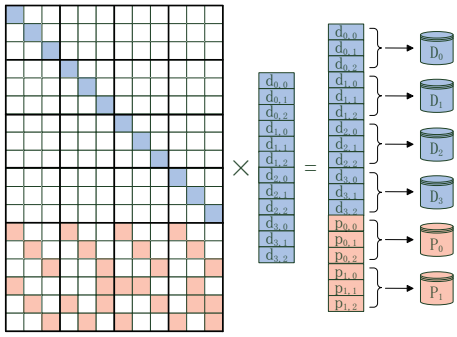


Figure 2: An example of generator bitmatrix of Caughy Reed-Solomon code with $\boldsymbol{n = 6}$, $\boldsymbol{k = 4}$, $\boldsymbol{m = 2}$, and $\boldsymbol{w = 3}$.

Erasure Coded Storage Systems: Though Figure 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. Furthermore, if a disk only contains parities, the storage system usually encounters a bottleneck on this disk under intensive write operations, because the up-

date 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.

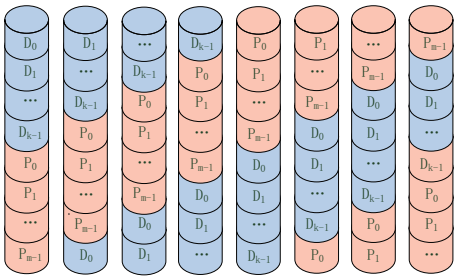


Figure 3: A widely used case of one stack in practical erasure coded storage systems.

Figure 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.

B. 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 symbols (in the k surviving disks) and the converted decoding bitmatrix will generate the original wk data symbols. In recent years, some researches have been proposed for improving the recovery speed. In this subsection, We will introduce these works.

Recovery Equations: A recovery equation is composed by a set of data symbols and parity symbols, whose XOR sum equal zero. If any symbol of the recovery equation is lost, we can reconstruct it by other survived symbols. For example, in Figure 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 symbols are failed, each failed symbol may belong to a group of recovery equations. We denote $E_{d_{i,j}}$ and $E_{p_{ij}}$ 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 symbols, this method reduces up to 25% I/O cost than the conventional method. We now explain the hybrid recovery principle based on the example of Figure 2.

Suppose that D_0 is failed, we can recovery all the failed symbols 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 symbols in total. However, if we use some parity symbols in P_1, the failed symbols 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 $d_{0,2}$.

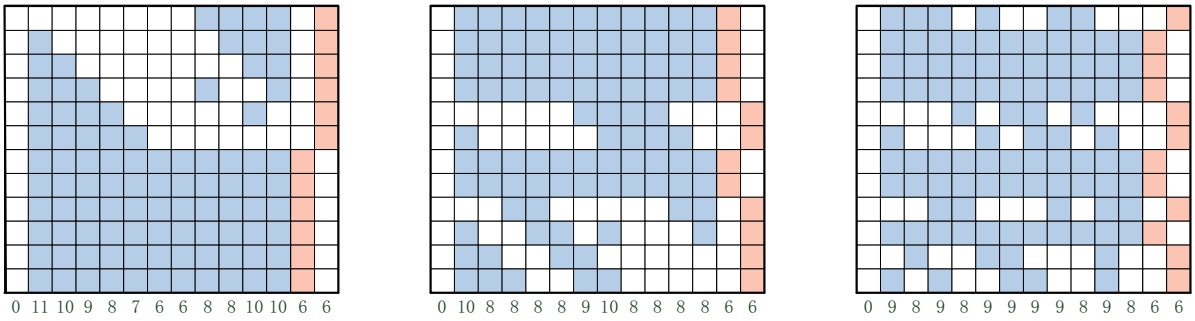
Obviously, this method only needs to read 10 symbols, because $d_{1,1}$, $d_{2,0}$, and $d_{3,0}$ are used in recovering two failed symbols. Therefore, based on hybrid recovery principle, we can reduce the I/O cost on single failure recoveries.

Hybrid Recovery Methods: Besides [5], Wang et al. [14] proposed a similar method to minimize recovery I/O cost in EVENODD code [15]. Xu et al. [16] proposed another optimal method for X-Code [17] 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 symbols F; 2) calculate each $E_{d_{i,j}}$ or $E_{p_{i,j}}$ for each symbol 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. [18] proposed another general method to find out the approximate solution with polynomial complexity.

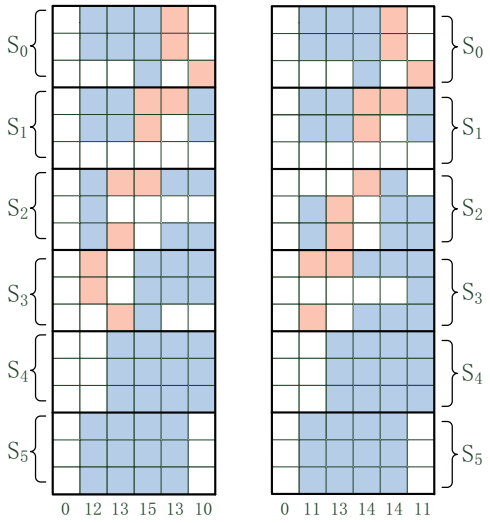
On the other hand, Luo et al. [8] pointed out that the recovery speed is due to the amount of read accesses on the heaviest loaded disk, and proposed two other general search-based algorithms to generate recovery schemes(C-Scheme and U-Scheme). Similar to Khan's Scheme, C-Scheme searches the solution 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 Figure 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 Figure 4, U-Scheme finds out the solution in (c), because the I/O cost on the heaviest loaded disk is the minimal. The experiment



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

Figure 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 symbols that require to be accessed, while orange squares indicate the parity symbols that need to be read).



(a) (b)

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

results in [8] shows that C-Scheme is faster than Khan's Scheme, while U-Scheme is even faster than C-Scheme.

III. 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 symbols 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 Figure 2 with the mappings from logic disks to physical disks shown in Figure 3, and compare the recovery efficiency in the stripe level and in the stack level, respectively.

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

Load Balancing among The Stack: When it 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, Figure 5(b) gives another stack-based recovery scheme, which needs to access the same amount of symbols (63 symbols) as in Figure 5(a), but owns a lighter load on the busiest disk (only 14 symbols' retrieval). This observation motivates us the necessity to design the recovery scheme with balanced I/O distribution.

Memory Optimization: Another important issue is the memory usage when designing stack-based recovery algorithms, because directly reading all the needed symbols 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 symbol is $16MB$ (this setting is commonly used in many community [7][19]) and the rotated mappings follow

Figure 3. Under this configuration, a stripe contains $4GB$ information (when $w = 16$) and the stripe-based method needs the memory with the size of $3.75GB$. When it comes to stack-level, if we directly access all required symbols like stripe-based method, the needed memory will expand to more than $48GB$, which is unacceptable for most of modern storage servers.

Fortunately, each stripe is independently encoded. As long as all the needed symbols of a stripe are read, we can recover the lost symbols of this stripe, thus don't have to wait until all symbols 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 symbols.

IV. The Stack-Based Recovery Algorithms

In this section, we will present the detailed design of our proposed algorithms, and analyze its complexity.

A. BP-Scheme

Motivated by the above section which indicates that there actually exist 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. Simulated annealing algorithm is a generic probabilistic metaheuristic algorithm for addressing the global optimization problem in a large search space when the search space is discrete [11], and it is usually more efficient than enumeration-based algorithms because the goal is merely to find an acceptable solution rather than the best solution.

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 symbols 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.

In Algorithm 1, we first calculate the recovery equations of the failed symbols in each stripe based on a similar method as in [20] (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

Algorithm 1 Balancing Prior Scheme (BP-Scheme)

```
1: initial parameter α
2: for each S_i (i ∈ [0, l − 1]) do
3:    E_all ← Calculate all recovery equations
4:    minimal_i = int_max
5:    for each s_{i,j} ← E_all do
6:       if |s_{i,j}| ⩽ minimal_i + α then
7:          RS_i ← 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} ∈ RS_i do
14:      if |s_{i,j}| > minimal_i + α 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 ← 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} ∈ RS_i do
26:       Solution_new = replace_state(Solution, s_{i,j}, i)
27:       de_new = cal_energy(Solution_new)
28:       Δt = de_new − de
29:       if Δt > 0 or exp(Δt/T) > rand()/rand_max then
30:          Solution = Solution_new
31:          de = de_new
32:          remain_times = M
33:          if Δt < 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
```

then define α as the threshold value and store all the schemes that need to read $[minimal_i, minimal_i + \alpha]$ symbols 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 $minimal_i + \alpha$ symbols (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).

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.

B. Rotated Recovery Algorithm (RR-Algorithm)

In BP-Scheme, we find out a near-optimal solution $Solution$ for recovering the lost symbols in stack-level. However, as referred in Section III, directly applying BP-Scheme (stack-based scheme) and simultaneously reading all needed symbols 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 Rotated Recovery Algorithm (RR-Algorithm), in order to save memory space while providing the similar recovery speed as simultaneously reading all needed symbols.

The key idea of RR-Algorithm is to set a parameter β for denoting the amount of symbols 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 symbols all the way, until the rest memory is not enough for each disk read β symbols. The recovery thread will recalculate the information of the lost symbols of each stripe when all needed symbols of this stripe are read, because the encoding of each stripe is independent. The detailed procedures of our algorithm is presented in Algorithm 2.

In Algorithm 2, we initiate three essential parameters: $read\_sche$, $reco\_sche$ records the needed symbols and the XOR relationship to recover each lost symbol respectively, which are generated by BP-Scheme; β is the number of symbols 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 symbols 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 0 at the beginning.

Algorithm 2 Rotated Recovery Algorithm (RR-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_Symbols(data, read_sche, β)
8:     read_stripe ← the last stripe that all needed symbols 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
```

During the recovery, two threads are simultaneously executed, where the read thread will take the constant number of β symbols from the disks into the memory (Step 6-9) and the other thread will repair the lost symbols based on the symbols in the memory (Step 11-17). The read thread will repeat until all the requested symbols are all read (when the memory is not enough, it will wait for recovery thread free memory). The recovery thread will recover lost symbols 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 symbols in the stack have been recovered.

C. The Searching Complexity

We now compare the complexity of BP-Scheme with Khan's Scheme in the rotated scenarios. Suppose a stack contains l different logic disk failed cases, both Khan's Scheme and the first stage of BP-Scheme (Step 1-18) need to search l disk failed cases by enumerating all feasible stripe-based solutions, thus their complexity are close. Furthermore, since simulated annealing algorithm is polynomial time, the other stage of BP-Scheme is polynomial time as well, which is negligible in front of the first stage. Therefore, the complexity of BP-Scheme and Khan's Scheme are close.

To verify our analysis, we evaluate the searching time of both BP-Scheme and other search-based schemes. We consider RDP code, EVENODD code, Liberation code, Generalize RDP code and STAR code with 16 disks, because the searching time of each recovery algorithm is modest.

Table I shows the running time of different recovery schemes. As it shows, the time overhead of BP-Scheme is extremely close to Khan's Scheme: BP-Scheme is just 0-9 seconds slower than Khan's Scheme. On the other hand, Luo's U-Scheme is slower than other schemes, because the complexity of calculating the I/O cost on the heaviest loaded disk is much higher than calculating the total I/O cost.

Table I: The time overhead of different recovery schemes in various codes with 16-disks

Code	Khan's	Luo's	BP-Scheme
Blaum-Roth	9s	11s	12s
RDP	42s	66s	44s
EVENODD	682s	653s	682s
Liberation	16s	31s	19s
STAR	257s	379s	260s
Generalized RDP	3060s	6230s	3069s

V. Analysis of Recovery Speed

In this section, we will analyze the recovery speed of different schemes theoretically. The goal of our analysis is to illustrate that BP-Scheme provides much high recovery speed than Khan's Scheme and Luo's schemes, while achieves the approximate recovery speed to the lower bound in the stack-level. We select the scenario that a stack contains n different stripes shown in Figure 3, because it is commonly used in erasure code community [7][9][10].

A. The Analysis Metrics

We analyze the recovery speed of different schemes by defining the *recovery factor*. Furthermore, we deduce the *lower bound* of recovery factor based on pigeonhole principle, for analyzing the gap between the recovery factor of BP-Scheme and the theoretical optimal recovery factor in the stack-level.

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

We denote max_read_i as the maximum symbols that need to be read on the heaviest disk of stripe S_i ($0 \leqslant i \leqslant n-1$), and denote max_read_{all} as the maximum symbols required on the heaviest loaded disk of a stack. Under these definitions, if we use stripe-based method (Like Khan's Scheme and Luo's schemes), the recovery factor can be represented in mathematics as follow.

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

However, if we use stack-based method, the recovery factor will be calculated by the following equation.

$$recovery\_factor = \frac{max\_read_{all}}{nw} \quad (2)$$

It is easy to deduce that recovering one failed symbol needs $recovery\_factor$ parallel read accesses on average. Therefore, the recovery speed should be linear with $\frac{1}{recovery\_factor}$, because both recalculating broken symbols and rewriting to the new disk can be processed in parallel with the reading process [18].

Lower Bound: We deduce a lower bound for evaluating the theoretical minimum recovery factor in stack-level by the following equation.

$$Lower\_Bound = \frac{\lceil \sum_{i=0}^{n-1} minimal_i/(n-1) \rceil}{nw} \quad (3)$$

In Equation (3), $\sum_{i=0}^{n-1} minimal_i$ is the minimal number of symbols 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 $\lceil \sum_{i=0}^{n-1} minimal_i/(n-1) \rceil$ symbols, 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 in any erasure array codes must no less than $\frac{\lceil \sum_{i=0}^{n-1} minimal_i/(n-1) \rceil}{nw}$. For example, in the case of Figure 5(a), the lower bound is $\frac{\lceil (10+9+10+10+12+12)/5 \rceil}{6\times 3} = 0.7$.

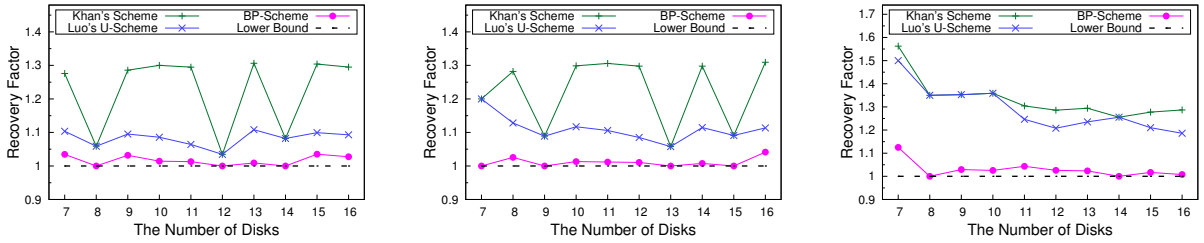
B. The Analysis Methodology

We select RDP code [6], Liberation code [9], and STAR code [24] to compare our BP-Scheme with Khan's Scheme and Luo's schemes, because these schemes can adapt for any erasure array codes and other schemes like in [5] can be found by either of these schemes. Furthermore, for Luo's schemes, we select the U-Scheme as comparison, due to that U-Scheme achieves higher recovery speed than C-Scheme. The parameter of the analyzed codes are shown in Table II. Moreover, to clarify the gap between these schemes, all our results normalize the value of lower bound as **one**.

For Khan's Scheme and Luo's U-Scheme, we calculate the recovery factor based on Equation (1). For BP-Scheme, we select parameter $T = 1$, $K = 0.999$, and $\alpha = \lceil minimal_i/100 \rceil$ to assure that the I/O cost is small, and compute the recovery factor based on Equation (2). This is because Khan's Scheme and Luo's U-Scheme are stripe-based, while BP-Scheme is stack-based.

C. The Analysis Results

Figure 6 shows the recovery factor in different erasure codes. As it shows, BP-Scheme provides lower recovery factor than Khan's Scheme and Luo's U-Scheme, while very close to the lower bound. Specifically, for RDP code, BP-Scheme provides 3.4% to 22.8% less recovery factor than Khan's Scheme, 3.4% to 9.0% less than Luo's U-Scheme, and only 0 to 3.5% more than lower bound. Since the recovery speed should be linear with $\frac{1}{recovery\_factor}$, BP-Scheme will achieve 3.5% to 29.5% higher recovery speed than Khan's Scheme, 3.5% to 9.9% higher recovery speed than Luo's U-Scheme, and only 0 to 3.5% slower than the optimal recovery speed.



(a) RDP. (b) Liberation. (c) STAR.

Figure 6: Recovery factor in different erasure codes.

For STAR code, BP-Scheme achieves only 0 to 4.3% higher recovery factor than the lower bound when $n = 8$ to 16, but suffers 12.5% higher than the lower bound when $n = 7$. This is 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 n is small. Compared with Khan's Scheme and Luo's U-Scheme, BP-Scheme provides up to 36.0% less recovery factor, which indicates that BP-Scheme is much faster than these schemes.

Furthermore, 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 recovery speed in single disk failure recovery.

Table II: The erasure codes and their paraments we tested(For each k, we select the smallest w if they are matched.)

Code	m	k	w	Restrictions
Caughy-RS [25]	2	5-14	8	None
Blaum-Roth [22]	2	5-14	6-16	$w+1$ prime $> k$
RDP [6]	2	5-14	6-16	$w+1$ prime $> k$
EVENODD [15]	2	5-14	4-16	$w+1$ prime $\geqslant k$
Liberation [9]	2	5-14	7-17	w prime $\geqslant k$
Caughy-RS^2 [25]	3	4-13	8	None
Generalized RDP [23]	3	4-13	4-16	$w+1$ prime $> k$
STAR [24]	3	4-13	4-12	$w+1$ prime $\geqslant k$

VI. Experiment Evaluations

We have built 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 and BP-Scheme by C++ language, and implement all erasure codes of Table II in 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 Figure 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 [18], i.e., read process and recovery process are run concurrently.

A. Experimental Setup

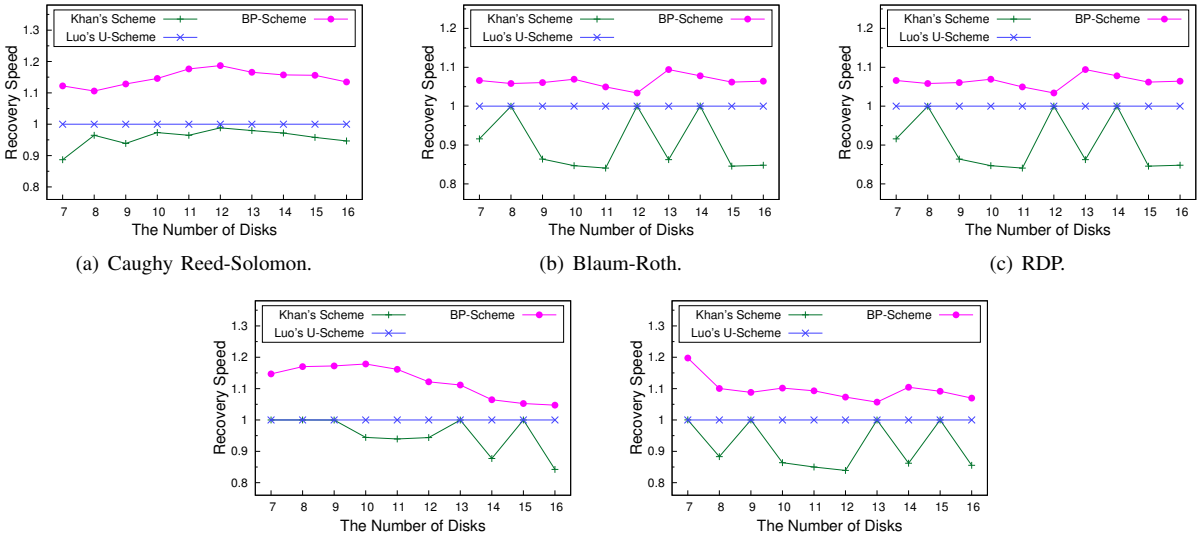
Our experiments are run on a machine and a disk array. The hardware environment of the machine is X5472 processor and 12GB memory. The disk array contains 16 Seagate/Savvio 10K.3 SAS disks, where the model number is ST 9300603SS. Each disk has 300GB capability and 10000rmp. The machine and disk array are connected by a fiber cable with 800MB bandwidth. The operation system of the machine is SUSE with Linux fs91 3.2.16. Our experiments set each symbol with a size of 16MB, which is a typical choice in storage systems [7][19]. All tests mirror the configurations in Table II, 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.

B. Evaluating of Recovery Speed

We evaluate the recovery speed of BP-Scheme by comparing with Khan' Scheme and Luo's U-Scheme. The goal of these experiments is to verify our theoretical analysis and to demonstrate that BP-Scheme achieves higher recovery speed than these 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. To clarify the gap between these schemes, all our evaluation results normalize the value of Luo's U-Scheme as **one**.

Recovery Speed in Different Codes with $m = 2$: Figure 7 shows the recovery speed for various $m = 2$ erasure codes with different schemes and illustrates that BP-Scheme provides much higher recovery speed than Khan's Scheme and Luo's U-Scheme. Specifically, BP-Scheme gains 17.7% to 26.5% higher speed than Khan's Scheme, and 10.6% to 18.7% higher speed than Luo's U-Scheme in Caughy Reed-Solomon code; achieves 16.5% to 23.7% higher speed than Khan's Scheme, and 7.4% to 20.3% higher speed than Luo's U-Scheme in Blaum-Roth code.

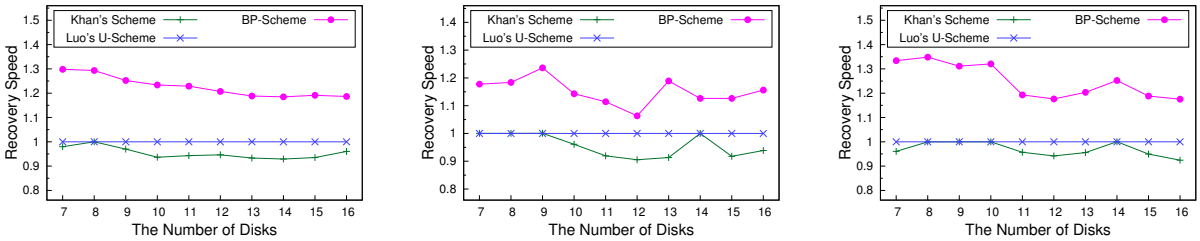
The improvement in various codes are different due to the fact that each codes have its own properties. For RDP code, Xiang et al in [5] has proved that it can achieve well balancing in each stripe under the standard forms (i.e., $n = 8, 12, 14$), while other forms also have well balanced stripe-



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

(d) EVENODD. (e) Liberation.

Figure 7: Recovery speed in different erasure codes with $m = 2$.



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

Figure 8: Recovery speed in different erasure codes with $m = 3$.

based solutions and that can be found by Luo's U-Scheme. On the other hand, the generating of Caughy 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% higher speed than Khan's Scheme, and achieves 3.4% to 20.3% higher speed than Luo's U-Scheme. Notice that our analysis results indicate that BP-Scheme will provide 3.5% to 29.5% higher speed than Khan's Scheme in RDP Code, but the actual improvement is 26.9%. 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 parallel stripe-based recovery method. Overall, the experiment results match the theoretical analysis results and indicate that the recovery speed of BP-Scheme is much higher than these schemes in $m = 2$ codes.

Recovery Speed in Different Codes with $m = 3$: We now consider the erasure codes with $m = 3$. Figure 8 shows the experiments results and illustrates that BP-Scheme provides higher recovery speed than Khan's Scheme and Luo's U-Scheme. Specifically, BP-Scheme provides 23.5% to 32.5% higher speed than Khan's Scheme and 18.5% to 29.8% higher speed than Luo's scheme in Caughy Reed-Solomin code, while achieves 24.7% to 38.9% higher speed than Khan's Scheme and 17.5% to 34.8% higher speed than Luo's Scheme in STAR code.

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%, because this code has well balanced stripe-based solution in each stripe when $n = 12$ and that can be found by Luo's U-Scheme. 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 stripes of each stack. On the other hand, the improvement in $m = 3$ codes is much higher than in

$m = 2$ codes, because $m = 3$ codes have two parity disk failed cases. When suffering from parity disk failures, the number of symbols 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 can balance them in other corresponding stripes, thus provide much higher speed.

In summary, BP-Scheme provides 12.6% to 38.9% higher recovery speed than Khan's Scheme, while gains 6.4% to 34.8% higher recovery speed than Luo's U-Scheme. The results indicate that our proposed scheme provides much better performance than these schemes.

C. Evaluating of Memory Overhead

We now evaluate the memory overhead with the goal of illustrating that the memory overhead of BP-Scheme (under RR-Algorithm) is acceptable.

Table III: The memory overhead of different recovery schemes in various codes with 16-disks (unit:MB)

Code	Khan's	Luo's	BP-Scheme
Blaum-Roth	3840	3840	9264
RDP	3840	3840	8208
EVENODD	3840	3840	8880
Liberation	4080	4080	10000
STAR	2688	2688	7824
Generalized RDP	3584	3584	8464

Table III illustrates the minimal memory overhead of each scheme in distinct codes with 16-disks. As it shows, BP-Scheme uses 1-2 times more memory than Khan's Scheme and Luo's U-Scheme. This is because stack-based recovery method needs to simultaneously consider multiple stripes rather than only one stripe, and thus needs more memory space. Fortunately, even when $n = 16$, the memory overhead of each code is less than $10G$, which is acceptable in state-of-the-art servers.

VII. Conclusions

In this paper, we have proposed a recovery generation scheme (BP-Scheme), 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 near-optimal stack-based recovery scheme based on simulated annealing algorithm. We also proposed a rotated recovery algorithm (RR-Algorithm) for BP-Scheme realization to reduce the memory overhead. Through a series of rigorous statistic analysis and intensive evaluations on a real system, the results demonstrate that BP-Scheme gains much higher recovery speed than other existing schemes.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (Grant No. 61232003, 61327902), the Special Program for Science and Technology of China (Grant No. 2013ZX03002004-003), the research fund of Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and the Tsinghua University Initiative Scientific Research Program.

References

[1] S. Ghemawat, H. Gobioff, and S. Leung. The google file system. In Proc. of SOSP'03, 2003.
[2] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. Mckelvie, Y. Xu, S, Srivastav, J. Wu, H. Simitci, et al. Windows azure system: a highly available cloud storage service with strong consistency. In Proc. of SOSP'11, 2011.
[3] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, et al. Oceanstore: An architecture for global-scale persistent storage. In Proc. of ASPLOS'00, 2000.
[4] E. Pinheiro, W. Weber, and L. Barroso. Failure trends in a large disk drive population. In Proc. of FAST'07, February 2007.
[5] L. Xiang, Y. Xu, J. Lui, and Q. Chang. Optimal recovery of single disk failures in RDP code storage systems. In Proc. of SIGMETRICS'10, New York, 2010.
[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. of FAST'04, San Francisco, March 2004.
[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. of FAST'12, February 2012.
[8] X. Luo, and J. Shu. Load-Balanced Recovery Schemes for Single-disk Failure in Storage Systems with Any Erasure Code. In Proc. of ICPP'13, October, 2013.
[9] J. Plank. The RAID-6 liberation codes. In Proc. of FAST'08, 2008.
[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. of FAST'09, February, 2009.
[11] Kirkpatrick. S, Gelatt. C. D, and Vecchi. M. P. Optimization by Simulated Annealing. Science, New Series, 220(4598):671C680, 1983.
[12] J. Plank, S. Simmerman, and C. Schuman. Jerasure: A library in C/C++ facilitating erasure coding for storage applications-Version 1.2. Technical Report CS-08-627, University of Tennessee, 2008.
[13] J. Hafner, V. Deenadhayalan, T. Kanungo, and K. Rao. Performance metrics for erasure codes in storage systems. Technical Report, RJ 10321, IBM Research, 2004.
[14] Z. Wang, A. Dimakis, and J.Bruck. Rebuilding for array codes in distributed storage systems. In GLOBALCOM Workshops, 2010.
[15] M. Blaum, J. Bruck, and J. Nebib. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. IEEE Transaction on Information Theory, 45(1):46-59, Junuary 1999.
[16] S. Xu, R. Li, P. Lee, Y. Zhu, L. Xiang, Y. Xu, and J. Lui. Single disk falure recovery for X-code-based parallel storage systems. IEEE Transaction on Computers, PP:99, January, 2013.
[17] L. Xu, and J. Bruck. X-Code: MDS array codes with optimal encoding. IEEE Transaction on Information Theory, 45(1):272-276, January 1999.
[18] 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. of MSST'12, 2012.
[19] 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. of FAST'04, 2004.
[20] K. Greenan, X. Li, and J. Wylie. Flat XOR-Based Erasure Codes in Storage Systems: Constructions, Efficient Recovery, and Tradeoffs. In Proc. of MSST'10, 2010.
[21] F. J. MacWilliams and N. J. A. Sloane. The Theory of Error-Correcting Codes. New York: North-Holland, 1977.
[22] M. Blaum, and R. Roth. On lowest density MDS codes. IEEE Transactions on Information Theory, 45(1):46-59, 1999.
[23] M. Blaum. A family of MDS array codes with minimal number of encoding operations. In IEEE ISIT'06, September, 2006.
[24] C. Huang and L. Xu. STAR: an efficient coding scheme for correcting triple storage node failures. In Proc. of FAST'05, 2005.
[25] J. Blomer, M. Kalfane, R. Krap, M. Karpinski, M. Luby, and D. Zuckerman. An XOR-based Erasure-Resilient coding scheme. Technical Report TR-95-048, International Computer Science Institute, 1995.