Load-Balanced Recovery Schemes for Single-disk Failure in Storage Systems with Any Erasure Code

Xianghong Luo and Jiwu Shu
Tsinghua National Laboratory for Information Science and Technology (TNList)
National Engineering Laboratory for Disaster Backup and Recovery
Department of Computer Science and Technology
Tsinghua University
Beijing, China
luo-xh09@mails.tsinghua.edu.cn, shujw@tsinghua.edu.cn
+Corresponding author: shujw@tsinghua.edu.cn

***Abstract*—As increasingly growing volume of data demanding high reliability are stored in disk arrays protected by erasure code, various codes with different error detection and correction capabilities are proposed. For higher reliability, codes that can correct multiple errors (such as RDP, EVENODD, and STAR) become popular. For each of the codes, there can be a number of recovery schemes for re-generating lost data. Among them the one recovering data for a single disk failure is the most critical to systems' performance and reliability as in most systems the recovery process is initiated as soon as the first failure is detected to reduce the window of vulnerability. Although there are efforts on improving recovery performance for single-disk failure, they either focus only on minimizing the total amount of data accessed for the recovery, which is not necessarily translated into minimal recovery time, or design only for specific codes and lack generality.**

In this paper, we propose two recovery algorithms that can not only work with any erasure code and produce minimal amount of accessed data, but also minimize the variation of volume of the data accessed on different disks. By minimizing the variation, the disk access can be fully parallelized and the recovery load is balanced, resulting in a faster recovery. We have implemented the recovery schemes in the Jerasure (ver. 1.2) library and evaluated them on a system with 16 SAS disks. Our measurements show that the recovery schemes generated by our algorithms reduce the recovery time for single disk failure situations by as high as 19.9% compared with the state-of-the-art recovery schemes.

***Keywords*-erasure code; single disk failure; load balance; recovery time;**

I. Introduction

In today's data center, hundreds of thousands of disks are employed to store rapidly growing amount of data and provide access to them supporting big-data applications and web-scale services. With disk count at this sale, disk failure is a norm [1] and constantly creates windows of vulnerability, in which I/O throughput is degraded and the system is subject to additional disk failures and risk of losing data. While failure detection and recovery mechanisms are implemented in various software layers, the erasure code for error detection and correction built in individual disk arrays provide the first-line of protection and are critical to data integrity, data availability, and system's performance. To this end, a number of erasure coding algorithms have been proposed and widely used, including RDP [2], EVENODD [3], and STAR [4] to protect a disk array from multiple-disk failure.

To ensure data availability, today's storage systems usually adopt on-line recovery [5], in which the system keeps serving requests from applications with a higher priority while the recovery is in process. Because data recovery usually starts as soon as the first failure is detected and the recovery time is usually smaller than disk-array's MTBF (Mean Time Between Failures), single disk failure occurs much more frequently than multiple disk failures. It has been reported that 99.75% of recoveries are for the single disk failure [6]. Accordingly, the performance of recovery for single disk failure is of high importance to reduce window of vulnerability. In this paper we use recovery time as the metric to measure the recovery performance. In this context, recovery time does not include the time period for writing the recovered data to the new disk(s). The rationale is that for on-line recovery it is the impact of the recovery on the quality of servicing user requests that matters. A recovery scheme can schedule recovery of lost data items in an order consistent to their request order, and try to keep the write-back of the recovered data in the background.

The data recovery for erasure-code protected systems has drawn much attention over years and a number of approaches have been proposed to address the issue from various perspectives [7]. Some erasure codes are designed with the consideration of reducing the amount of read data in the recovery, including non-MDS (Maximum Distance Separable) [8] codes such as Pyramid code [9] and Stepped Combination codes [10]. However, their recovery performance is determined when as long as the codes are designed. We still lack a general scheme that can efficiently recover lost data protected by any existing erasure code.

Khan et al. [11] propose a recovery algorithm that can generate recovery schemes requiring minimal amount of read data for the single disk failure with any erasure code. However, the algorithm does not consider the distribution of the data across the disks. In case where most data are read from a small portion of the disk array, the unbalanced I/O load can lead to a recovery performance worse than a scheme reading more data in its recovery. Xiang et al. [12],

0190-3918/13 $26.00 © 2013 IEEE
DOI 10.1109/ICPP.2013.69

[13] propose algorithms generating the recovery schemes of minimal read data and balanced load for the single disk failure situation. However, their algorithms are designed only for the RDP code [12] and EVENODD code [13] and lack generality.

We address the data recovery problem for single disk failure by developing two algorithms generating high-performance recovery schemes for *any* codes. The first one ensures the recovery schemes maximizing load balance under the condition of minimal read data for the recovery, and we name it as conditional load-balance algorithm or *C-Algorithm* in short. The second one is to generate recovery schemes maximizing load balance without having to always read minimal amount of data, and we name it as unconditional load-balance algorithm, or *U-Algorithm* in short. With multiple disks serving the read requests for a recovery with parallel I/O, it is the service time of the disk that reads the largest amount of data that determine the recovery time. Even increasing the total read load compared with Khan's algorithm and C-Algorithm, U-Algorithm can minimize the read load of the most loaded disk and lead to a shorter recovery time. These recovery schemes are orthogonal to other existing recovery optimization strategies [7], and can be used together with them.

In this paper we make the following contributions:

1) We propose two optimized algorithms for generating recovery schemes with the objective of minimal recovery time. They take advantage of the design of state-of-the-art algorithms for minimal read data, and improve them by keeping the load across the disks balanced. Additionally the algorithms are designed for any erasure code.
2) We show that in theory recovery schemes generated by C-Algorithm and U-Algorithm can reduce the recovery time by up to 22.9% and 25.0%, and averages of 9.6% and 16.4%, respectively, over recovery schemes generated by a state-of-the-art algorithm (the Khan's algorithm).
3) We implement the two proposed algorithms in Jerasure-1.2 [14] and evaluate their performance with various system configurations. The experiment results show that recovery schemes generated by C-Algorithm and U-Algorithm reduce the recovery time by as much as 15.5% and 19.9% , respectively, over those from the Khan's algorithm, which is in line with the theoretical results.

The rest of the paper will be organized as follows. The next section will provide some background. The C-Algorithm is described in Section III and the U-Algorithm is described in Section IV. Then we will theoretically evaluate these two algorithms in Section V and provide the experiment evaluation results in Section VI. Finally, we make conclusions in the last section.

II. Background

A. Terms and Notations in Erasure Code

In an erasure coded storage system, we always use n to represent the number of disks for storing user data, while m represents the number of disks for parity codes. Data are striped over the disks and parity codes are computed for user data in individual stripes. On each stripe there are k elements in each disk, and element is the unit of data access in the encoding, modification, and recovery processes.

In most of the papers about erasure code, code constructions are only discussed logically in a single stripe. However, there are a large number of stripes that are stored in a real storage system [15], the stripe rotations under this implementation naturally avoid the "modification performance bottleneck" problems.

We always use generator matrix to describe erasure code [16]. As defined above, the matrix is always a $mk \times nk$ one, which means that the matrix times the vector of nk user data elements would generate the vector of mk parity elements. Under several basic matrix transformations, we may derive a lot of code properties. For example, the recoverability of a failure situation is determined by the corresponding survivor matrix being singular or not.

Under an erasure code, we define *calculation equation* that the XOR sum of all the elements within a calculation equation is equal to zero. For example, the user data elements used to generate a parity element and this parity element itself make up a calculation equation. As a result, each row of the generator matrix corresponds to a calculation equation while the iterative row transformations on generator matrix would create other calculation equations. In particular, we define the set of mk calculation equations that correspond to each row of the generator matrix as the set of original calculation equations for erasure code.

In a calculation equation, when an element fails, we can use the XOR sum of all the other elements to recover this failed element. All these non-failed elements are named as the *surviving elements*. During the recovery process, we have to read some surviving elements to help recover the failed elements. Furthermore, if two elements in a calculation equation are simultaneously failed, we can use the surviving elements in another calculation equation to represent one of failed elements; by replacing it, we can recover the other failed element. Under these iterations, we have an iteration algorithm [10] to generate all possible recovery schemes in any recoverable failure situation.

B. Recovery Speed in Single Disk Failure Situation

A lot of erasure codes have been proposed in the literature. However, no code to provide a fault tolerance of two or higher has been considered as the "standard". Among them, RDP code [2] and EVENODD code [3] are two of the most famous RAID-6 codes to provide a fault tolerance of two, while other array codes [17], such as STAR code [4] and generalized EVENODD code [18], provide even higher fault tolerance.

On-line recovery [5] requires storage systems to keep serving user requests while the recovery is in process. Because recovery is initiated as soon as the first failure is detected and the recovery time is always smaller than disk-array's MTBF (Mean Time Between Failures), the single disk failure accounts for 99.75% of all recoveries [6]. The performance of recovery for single disk failure situation is of high importance to reduce the window of vulnerability. During the recovery process, the XOR calculation for recovery in CPU is multiple orders of magnitude faster than the bandwidth of disk, so the crucial factor on recovery efficiency is the time spent on reading elements.

Most erasure codes are designed to read all surviving elements in the failure situations that the numbers of failed disks reach to its fault tolerance threshold. However, when a single disk fails, a degraded recovery scheme is to utilize the first parity disk and all the surviving user data elements to recover elements in the failed disk. We refer this degraded recovery scheme as the *naive recovery scheme* in this paper.

Either in EVENODD code or RDP code, we can recover a failed element by either its horizontal or diagonal calculation equation. Thus, there are 2^k different sets of surviving elements that could be read for recovering a single failed disk. However, different recovery schemes may have great gap on their performance. Even in the Lowest Density Codes (Liberation code [19], Liber8tion code [20], and Blaum-Roth code [21] are combined together to form a series of Lowest Density Codes), which propose the minimal number of "ones" in their generator matrices to provide higher performance, the recovery efficiency for single disk failure situation is also low if we choose to read an inappropriate set of surviving elements. For a specific single disk failure situation, which set of surviving elements should be read in the most efficient recovery scheme? And how can we rapidly generate this optimal recovery scheme? We have to solve these two problems to reduce the recovery time during recovery.

Xiang etc. proposed the algorithms for generating the optimal recovery schemes in the single disk failure situations of RDP code [12] and EVENODD code [13]. In these recovery schemes, both horizontal and diagonal calculation equations are used; the overlapping elements are read once but utilized twice. Thanks to these elements' re-utilization, Xiang's recovery schemes reduce 25% I/O cost compared with the naive recovery scheme. What's more, these recovery schemes not only access the theoretically minimal amount of data, but also evenly distribute the read data across all the disks, which provide load balance. However, these recovery schemes are designed specifically for RDP code or EVENODD code, lacking generality.

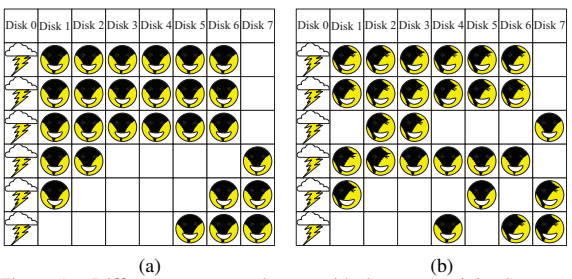
Khan etc. [11] provide an algorithm to generate recovery schemes for the single disk failure situations of *any* erasure code. However, they assume the most efficient recovery scheme is the one accessing the minimal amount of data. In this scheme, much data may be allocated on merely a portion of disks, leading these disks to be the bottleneck. What's worse, Khan's algorithm has not indicated which recovery scheme with the (same) minimal amount of read data should be chosen in case of a tie.

On the other hand, thanks to parallel I/O in RAID architecture, we can read an element from each disk synchronously by one parallel read access. So the recovery time is determined by the read load on the most loaded disk, thus the optimal recovery scheme must be the one that minimizes the maximum read load from a single disk. For example, motivated by parallel I/O property, the shifted mirror method [22] is tailor-designed for mirror method by adjusting its element arrangements, which provides high data availability during the reconstruction process of mirror method.

It should be noted that it is NP-Hard to find the optimal recovery schemes for a specific single disk failure situation [11]. We can address this issue only by search algorithms, so the algorithms' running time increases exponentially. Though some optimization may be used, such as the pruning in Khan's algorithm, we can only find recovery schemes for smaller number of disks within a relatively longer time. However, as the number of different single disk failure situations is equal to the number of disks, we can find the recovery schemes for each single disk failure situation ahead of time and directly use them whenever they are needed.

III. C-Algorithm: Optimizing Recovery Schemes with Minimal Read Data for Load Balance

In an instance of RDP code with 6 user data disks and 2 parity disks (from its definition [2], each disk stores 6 elements in a stripe), we assume the first disk fails, which leads to a single disk failure situation. Under Khan's algorithm, we can find several recovery schemes with minimal amount of read data, two of which are illustrated in Figure 1. In Figure 1, the two recovery schemes access the same (minimal) amount of data, but the right one is under load balance (this recovery scheme for RDP code could also be constructed by Xiang's algorithms [12]). An experiment on real disk array shows that the right recovery scheme provides 18.5% higher recovery speed than the left one.



(a) (b)

Figure 1. Different recovery schemes with the same (minimal) amount of read data for RDP code. (The lightning symbols represent the failed elements and the smile symbols represent the surviving elements to be read.)

In Khan's algorithm, a shortest path model is established: each node represents a set of surviving elements to recover a specific set of failed elements. Two nodes are connected with a directed edge if the end node could recover one more failed elements than the start node. The weight of this edge is marked as the incremented amount of read data from the start node to the end node. In this graph, they find the shortest path from the node that recovers no element to node that can recover all failed elements by Dijkstra's algorithm and prune the graph drastically, so that they can achieve the recovery schemes with minimal amount of read data. However, in case of a tie, the algorithm does not indicate which scheme with the minimal amount of read data should be chosen.

A. C-Algorithm

There are many recovery schemes with the (same) minimal amount of read data for a single disk failure situation, as discussed in Section II, the load-balance one is the best choice. Maintaining the property of parallel I/O, we improve Khan's algorithm by choosing the recovery scheme whose most loaded disk is minimal loaded as our solution, in the condition of minimizing amount of read data.

In the actual implementation, we only prune the branches which make the weight of current path larger than the current optimality. However, we keep traversing the paths whose weights are equal to the current optimality. In addition, when we reach the leaf nodes that could recover all failed data, we add a comparison function between the current recovery scheme and the temporary optimal one if they have to access the same amount of data. The comparison function chooses the recovery scheme whose read load in the most loaded disk is smaller as the new temporary optimality. As a result, our algorithm generates the recovery schemes that maximize load balance in the condition of minimal read data for recovery, thus it is a *conditional load-balance algorithm* (C-Algorithm). The generative recovery scheme is regarded as *C-Scheme*.

In our optimization for load balance, we prune no path in Khan's algorithm, but traverse a few more paths; at the same time, we have an additional comparison procedure. So the correctness of our C-Algorithm is the same as that of Khan's algorithm. It is admittedly that our C-Algorithm runs a little slower than Khan's algorithm, but it is negligible, which will be discussed in Subsection V-B. However, our C-Algorithm evenly distributes the same minimal amount of read data across all disks in the recovery scheme, which minimizes read load of the most loaded disk. The balanced read load significantly improves recovery speed during the recovery of single disk failure situation.

IV. U-Algorithm: Maximizing Load Balance for Recovery Schemes

Standard RDP code, EVENODD code and Liberation code are regularly constructed. In their recovery schemes with minimal amount of read data, we can evenly distribute the read data across all disks, which naturally maximizes load balance. This property of standard RDP code and EVENODD code has been proved by Xiang etc. [12], [13], and the proof for that of standard Liberation code is not hard.

However, most erasure codes are not regularly constructed, such as the Liber8tion code [20] and Blaum-Roth code [21]. Even in the "shorten" EVENODD code and "shorten" RDP code (the "shorten" method [23] is proposed to adjust EVENODD code and RDP code to adapt to arbitrary number of disks), we may read much data from a small portion of disks if we always consider their recovery schemes in the condition of minimizing the amount of read data. In Figure 2, let us take Liber8tion code as an example.

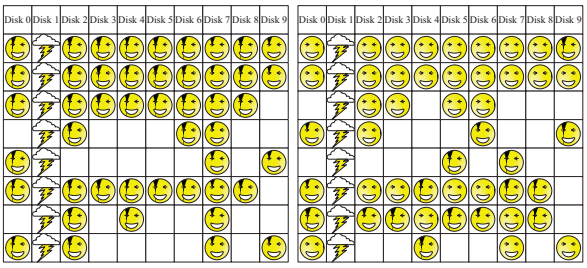


Figure 2. Different recovery schemes for Liber8tion code. (The lightning symbols represent the failed elements and the smile symbols represent the surviving elements to be read.)

There are 8 user data disks and 2 parity disks in Liber8tion code and each disk stores 8 elements in a stripe. In this single disk failure situation, we assume the second disk (disk 1 in Figure 2) fails. As shown in Figure 2(a), we provide the recovery scheme generated by C-Algorithm, which maintains load balance in the condition of accessing the minimal amount of data, but the last user data disk (disk 7 in Figure 2) is still the bottleneck.

However, if we jump out of the condition of accessing the minimal amount of data, we may provide more balanced read load. Another recovery scheme for this single disk failure situation is shown in Figure 2(b); the number of elements read from the most loaded disk is decreased from 8 in Figure 2(a) to 6 in Figure 2(b), although there is an increasing in the total number of elements (from 47 to 48). An experiment on real disk array shows that we reduce 16.0% of recovery time by changing the recovery scheme from Figure 2(a) to Figure 2(b).

As discussed in Section II, the optimal recovery scheme is the one that minimizes read load on the most loaded disk. In this section, our algorithm would maximize load balance without the condition of reading minimal amount of data, which is the *unconditional load-balance algorithm* (U-Algorithm). The generative recovery scheme is regarded as *U-Scheme*.

A. U-Algorithm

In the single disk failure situation of any erasure code, we have a **fail_ele** set, which is the unity of failed elements in this failure situation. The failed elements in **fail_ele** are labeled from 1 to k (we have k failed elements in a single disk failure situation). As defined in Section II, any specific erasure code has a set of its original calculation equations, **cal_equ**, where each equation represents the calculation relationship of a parity element. Taking these two sets as the input, our algorithm in Algorithm 1 generates the recovery schemes that maximize load balance for *any* erasure code in its single disk failure situations.

Algorithm 1: Unconditional Load-balance Recovery Generation Algorithm.

```
1:  rec_equ = Get_Rec_Equ(cal_equ, fail_ele);
2:  init_state.ele = ∅;
3:  init_state.cur = 1;
4:  insert init_state to rec_list[0];
5:  for r = 0; r <= k; r++ do
6:    for each state ∈ rec_list[r] do
7:      if state.cur > k then
8:        return state.ele;
9:      end if
10:     for each equ ∈ rec_equ[state.cur] do
11:       new_state.ele = state.ele ⋃ equ;
12:       new_state.cur = state.cur + 1;
13:       key = Max_Col(new_state.ele);
14:       insert new_state to rec_list[key];
15:     end for
16:   end for
17: end for
```

We define the *recovery equations* for each failed element that the elements in a recovery equation are all surviving elements and their XOR sum equals to this failed element. In Algorithm 1, using **cal_equ** and **fail_ele** as the input, we firstly generate all recovery equations for each failed element by **Get_Rec_Equ** function. As the output of **Get_Rec_Equ** function, we store all possible recovery equations of the i^{th} failed element in **rec_equ[i]**. In **Get_Rec_Equ** function, the recovery equations are generated not only directly from the original calculation equations, but also under the iteration algorithm [10] that deals with the recovery situations where one element is recovered based on other recovered failed elements.

A structure **state** in our algorithm includes two members: **ele** is a set of surviving elements to be read; and **cur** stands for the label of current failed element that we have to recover. A list of **states** are stored in **rec_list** to be traversed in our algorithm. We initialize **rec_list** in Lines 2-4 of Algorithm 1 by inserting a **state** with none elements in its **ele** member and assigning its **cur** member as 1. During the algorithm, we divide the **states** in **rec_list** into $k+1$ sublist, where **rec_list[r]** ($0 \leq r \leq k$) is a list of **states** that the numbers of elements read from the most loaded disk in their **ele** members are equal to r. As shown in Lines 5-6 of Algorithm 1, we traverse **states** in the ascending order of the number of elements in the most loaded disk in their **ele** members. This is the key point of our algorithm.

For each **state** we are about to deal with, if it has recovered all the failed elements (i.e. its **cur** member is larger than k in Line 7), we return its **ele** set as the solution, which is the U-Scheme. Otherwise, we extend the current **state** into new **states**, where we try to recover the failed element labeled with **cur**. We enumerate each recovery equation of this failed element and unite all the elements in this equation into the current **ele** set. By putting this set as the **ele** member of a new **state** (Line 11), we also move the **cur** member to the next failed element in the new **state** (Line 12). Moreover, the **Max_Col** function in Algorithm 1 is used to calculate the number of elements in the most loaded disk in **ele**, which is the indicator to the sublist that we are to insert the new **state** (Line 14).

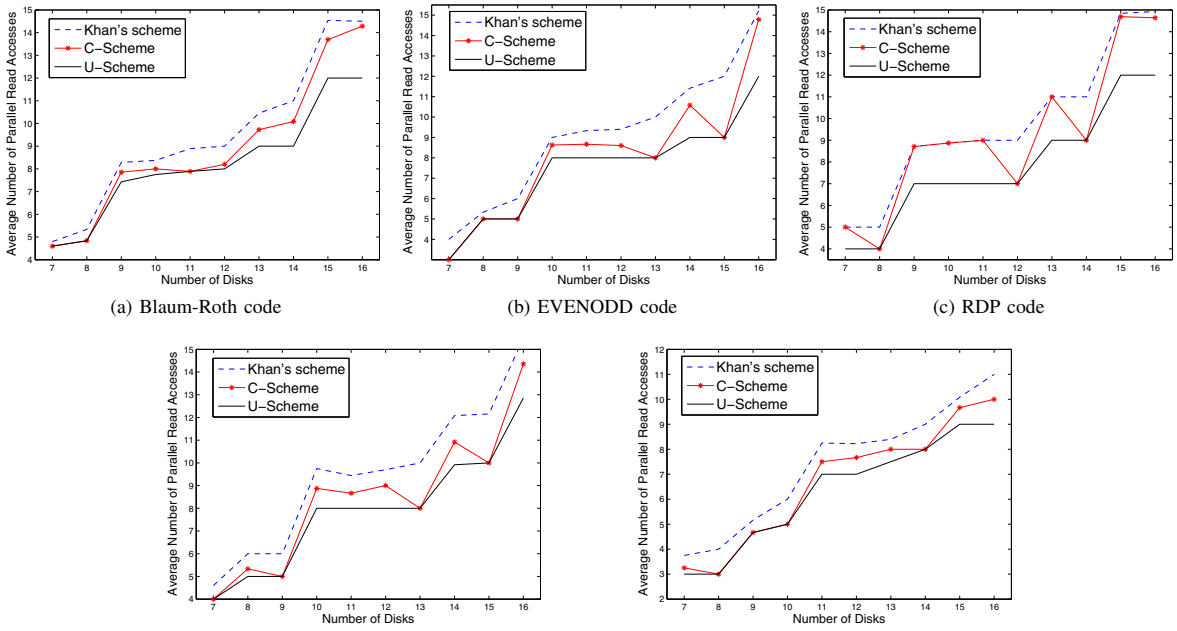
Thanks to parallel I/O, U-Scheme should be the one that minimizes the number of elements read from the most loaded disk. In Algorithm 1, we use this metric as the key value to divide the **states** into different sublists (Line 14), thus we traverse from the **states** with lower metrics to the **states** with higher metrics (Lines 5-6). As a result, we obtain the recovery scheme in which this metric is minimal. This is the key point of Algorithm 1, which generates our expected recovery schemes that maximize load balance.

B. Optimization for Minimizing the Amount of Read Data

Our U-Algorithm has to face the same problem as Khan's algorithm: in case of a tie, which recovery scheme should be chosen as the solution. Obviously, by maintaining the property of minimizing read load on most loaded disk, the recovery scheme accessing the minimal amount of data is the best choice, which minimizes the variation of volume of the data accessed on different disks. So we have to revise Algorithm 1 to cope with the tie situation.

Specifically in Algorithm 1, when the first time we find a recovery scheme to recover all failed elements, assuming its number of elements on the most loaded disk is equal to t, we temporarily don't return it as the solution. Instead, we keep traversing all **states** in the sublist **rec_list[t]**. Among them, we choose the one that both recovers all failed elements and accesses the minimal amount of data as the solution.

It is not hard to revise Algorithm 1 to realize the optimization of minimizing the amount of read data. However, it should be noted that when we traverse the **states** in **rec_list[t]** in the revised algorithm, we still have to extend them into new **states**. The extension of a **state** may not affect the number of elements read from the most loaded disk in its **ele** member, which only moves the current failed element label in **cur** to the next one. The new extended **state** may have the ability to recover all failed elements and could be the solution with the minimal



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

(d) Liberation code (e) STAR code

Figure 3. Average numbers of parallel read accesses in different recovery schemes on different erasure codes.

amount of read data.

Under this optimization, the elements order in **fail_ele** dose not matter any more. In the revised algorithm, we would not miss any recovery scheme that both recovers all failed elements and minimizes read load on the most loaded disk. Among them, we choose the one accessing minimal amount of data as our solution.

Above all, we have proposed an algorithm to generate recovery schemes that maximize load balance to improve the recovery speed of single disk failure situations. The U-Scheme has two properties: primarily, it minimizes the number of elements read from the most loaded disk; what's more, we make an optimization for minimizing the amount of read data, which further mitigates the whole system's read load.

V. Features

In this section, we will discuss the features and performance achieved by the load-balance recovery schemes. The two load-balance recovery generation algorithms proposed in this paper (either C-Algorithm or U-Algorithm) are compared with Khan's algorithm, the state-of-the-art algorithm. The evaluations include: the recovery speed in single disk failure situation, the running time of the programs, the reconstruction efficiency and these recovery generation algorithms used on other failure situations. We only focus on the theoretical analysis in this section and the experiment results will be proposed in the next section.

In the theoretical analysis of erasure code, the performance metrics could be measured by rigorous counting and averaging with an assumption of equal failure probability for all disks [15]. We follow the same methodology in measuring the performance of balanced read load. In particular, the stack notion [15] allows us to do the measurements in the following evaluations by rigorous counting and averaging on a simple stripe. To see that, all different disk mappings from logical to physical are rotated from stripe to stripe, and we can logically regard all disks in a stripe as having the same possibility of getting into failure.

A. Recovery Speed

In a specific single disk failure situation, the amount of failed data to be recovered is fixed, so the recovery speed is determined by the efficiency of reading surviving elements. Thanks to parallel I/O, we can read data from different disks synchronously, thus the actual recovery time is measured by the time spend on the most loaded disk.

Since disk mappings from logical to physical are rotated from stripe to stripe, a physical disk maps each logical disk with the same possibility. So the average recovery time for recovering one logical stripe is the same as the one when any physical disk fails. Using the average number of parallel read accesses as the metric for recovery time, we compare different recovery schemes for the single disk failure situations of different erasure codes in Figure 3. For a specific number of disks, we enumerate each user data disk as the virtual failed disk and generate the recovery schemes for this single disk failure situation. As shown in Figure 3, by averaging the number of parallel read accesses, we compare different recovery schemes with the numbers of disks varied from 7 to 16. The five sub-figures in Figure 3 respectively correspond to the single disk failure situations of Blaum-Roth code, EVENODD

code, RDP code, Liberation code and STAR code.

In our program, failed elements are sorted from top to bottom in a stripe. We select the first searched suitable (the suitable here means the recovery scheme could recover all failed elements) recovery scheme with minimal amount of read data as Khan's recovery scheme. Khan's recovery scheme reduces recovery time from the naive recovery scheme by much less read load, however it dose not consider the read load distribution. In contrast, our C-Scheme is established by selecting the load-balance one among all recovery schemes with minimal amount of read data, thus Khan's recovery scheme is also one of the candidate schemes. As a result, it is admittedly that C-Scheme reduces much recovery time from Khan's scheme.

However, under the condition of accessing minimal amount of data, the improvement of recovery speed in C-Scheme is not as high as our U-Scheme provides. The objective of U-Scheme is to evenly distribute elements across all disks, which maximizes load balance by minimizing read load on the most loaded disk, thus our U-Scheme needs the lowest recovery time.

It should be noted that in RDP code, EVENODD code and Liberation code without "shorten" method, the numbers of parallel read accesses in C-Scheme and U-Scheme are the same [12], [13]. However, our key idea on evenly distributing data across all disks reduces recovery time in any single disk failure situation of any disk array size.

For theoretical comparisons of recovery time on different recovery schemes, C-Scheme improves recovery speed by up to 22.9% and an average of 9.6% compared with Khan's recovery scheme. Furthermore, our U-Scheme has another 7.0% less average read load than C-Scheme, which improves recovery speed by up to 25.0% and an average of 16.4% compared with Khan's recovery scheme. What's more, the reducing of recovery time in Figure 3 goes higher and higher with the increasing of the number of disks because of higher I/O parallelism. Comparing STAR code with RAID-6 codes in Figure 3, there are more calculation equations in the higher failure tolerance code, so we have more choices to recover a failed element, which potentially needs less recovery time.

B. Running Time

It is NP-Hard [11] to find the optimal recovery schemes in any single disk failure situation. As we can solve it only by search algorithms, any recovery generation algorithm above is actually a search program. Fortunately, the number of different single disk failure situations is equal to the number of disks, so we can generate recovery scheme for each situation ahead of time and directly use it whenever it is needed.

Our C-Algorithm certainly runs slower than Khan's algorithm. However, a lot of nodes are considered in the Dijkstra's algorithm; a node recovers only one more failed element than its precursor node, so we have to explore at least k steps before we reach a node to recover all failed elements. Under our optimization for load balance, we traverse some more paths than Khan's algorithm, however the number of these paths is multiple orders of magnitude less than the number of paths we have to traverse in Khan's algorithm. Our experiment results show the optimization for load balance costs no more than one percent extra running time compared with Khan's algorithm. As a result, the impact on the running time of our optimization for load balance is negligible.

On the other hand, the running time of C-Algorithm and Khan's algorithm is determined by the search order. If we always first explore the longer paths, we nearly have to traverse the whole graph. In contrast, in our U-Algorithm, each sublist contains a fixed number of **states**. The number of sublists that we have to traverse is determined by the final answer (the number of elements in the most loaded disk in the generative recovery scheme). So that, the running time of U-Algorithm is more stable.

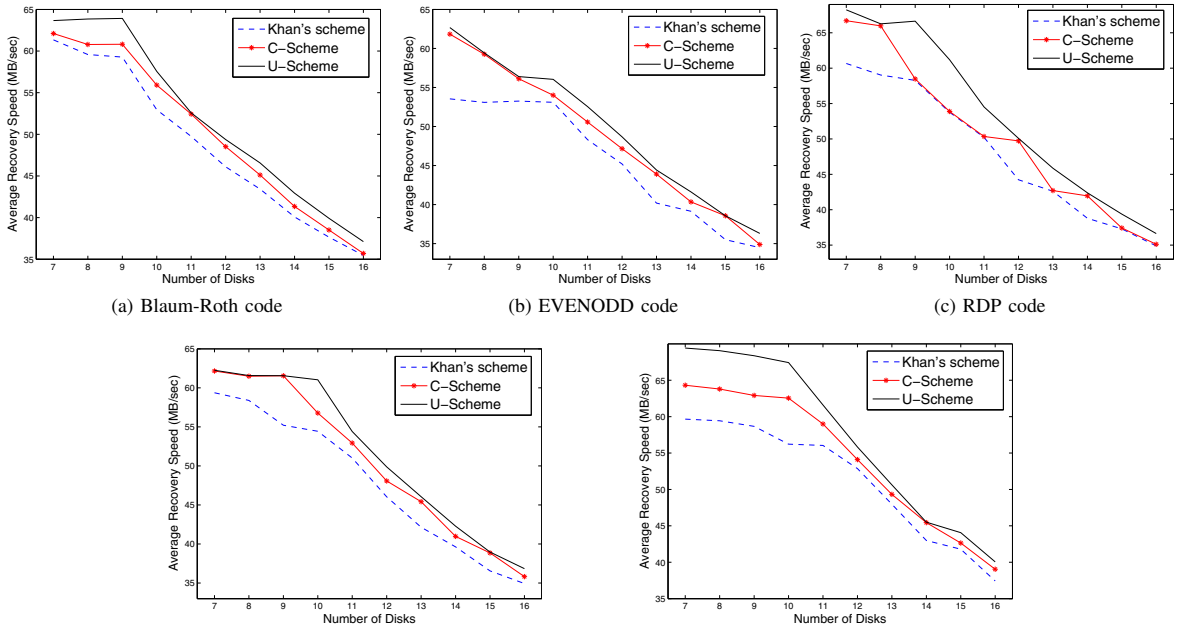
C. Reconstruction Efficiency

In most real storage systems, disk write speed is higher than its read speed (for example, in our experiment environment in the next section). Under our C-Scheme or U-Scheme, we can rapidly read data and recover the failed elements, which may potentially improve the reconstruction efficiency. Some other existing reconstruction optimization strategies [7] may be implemented on RAID level, our C-Scheme and U-scheme are orthogonal with them to provide much higher reconstruction efficiency.

D. Other Failure Situations

Actually, the single disk failure situation is a representative of the degraded failure situations for erasure code. In the erasure codes to provide a fault tolerance of higher than two (such as STAR code [4] and generalized EVENODD code [18]), a burst of multiple disk failures (for example, the simultaneously two-disks failure) is another degraded failure situation. Besides the whole disk failures [6], [24], there may be some partial disk failures on other disks, such as latent sector errors [25], [26], [27] and undetected disk errors [28]. The combinations of whole disk failures and partial disk failures also bring another kind of degraded failure situations.

To reduce recovery time in above failure situations, our objective is also to minimize read load for the most loaded disk. Actually, in Algorithm 1, the failed elements set (**fail_ele**) does not need to require the failed elements to be come from a single disk. In any failure situation, adding all the failed elements to **fail_ele**, the **Get_Rec_Equ** function always generates the recovery equations for each failed element [10]. As a result, we can always use Algorithm 1 to generate the recovery schemes that minimize read load on the most loaded disk. However, before running the algorithm, we have to add a function to check whether the failure situation is recoverable. This judgement is actually owned by Algorithm 1; if we have traversed all **states** in all the $k+1$ sublists and found no one could recover all the failed elements, we conclude that the failure situation is not recoverable.



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

(d) Liberation code (e) STAR code

Figure 4. Average recovery speed of different recovery schemes on different erasure codes.

In cloud storage systems [29], we have a lot of disks, different disks may provide different read speed and bandwidth. We have to provide the optimal recovery schemes in this heterogeneous environment. Specifically in heterogeneous environment, each disk has a weight value to identify the cost of reading an element from this disk. We use this weight value times the number of elements read from this disk, thus we get the total read cost for this disk. To revise Algorithm 1, we use the read cost on the most loaded disk as the metric to divide **states** into different sublists, which determines the traversal order. As a result, we generate the recovery scheme that minimizes the read cost on the most loaded disk, which provides load balance during recovery. Algorithm 1 in Section IV is actually a special case of the heterogeneous environment recovery, where the cost to read an element from each disk is equal to 1.

VI. Experiment Evaluations

A. Experiment Environment

All our experiments are performed on a machine with Intel Xeon processor (4 cores) of 2.50 GHz and 24G physical memory, and the Operating System is Fedora 10 with a Linux kernel of V2.6.27.5-117.fc10.x86_64. All the 16 SAS disks in our experiments are sitting in an individual array. The disk type is Seagate/Savvio 10K.3, and the disk model number is ST9300603SS. Every disk is 300 GB and 10000 rpm with a cache of 16 MB, with each providing a peak read speed of 56.1 MB/Sec and a peak write speed of 131 MB/Sec. All our programs are implemented based on an open source library, Jerasure-1.2 [14], which is widely used by the erasure code community [30].

In the following experiments, we set each element with a size of 16 MB, which is a typical choice in storage systems [11], [31]. The numbers of disks are varied from 7 to 16 in our experiments. The recovery speed is expressed as the amount of data recovered in unit time, so the actual size of data under recovery does not matter. We considered 20 stacks in each of the following experiments. According to its definition and property [15], each stack contains all possible disk mappings from logical to physical for a specific erasure code, so we have to recover as much as 74.375 GB data for a failed disk in our experiments. As a result, the following experiment results are true reflections of recovery speed. The property of stack also guarantees that a real disk failure involves all logical single disk failure situations, each with the same occurrence possibility.

Note that after the recovery process, we also compare the original data in the virtual failed disk with the recovered data, which verifies the correctness of our recovery process. In some erasure codes, the numbers of disks are restricted by prime numbers, however we can use the "shorten" method [23] to get rid of this limitation. In our experiments, we use this method to evaluate the recovery speed of these codes with different number of disks.

B. Experiment Results

We considered some RAID-6 codes and the STAR code in our experiments. For each single disk failure situation, we ran the three recovery generation algorithms, their generative recovery schemes were implemented on real disk array to measure the recovery speed; the results are shown in Figure 4.

As shown in Figure 4, our optimization for load balance in C-Scheme brings up to 15.5% improvement on recovery speed compared with Khan's recovery schemes. However, our U-Scheme, which reduce another 16.2% recovery time from C-Scheme, totally improves the recovery speed by as high as 19.9% compared with Khan's recovery scheme. These experiment results are in accord with the theoretical analysis in Figure 3 of Subsection V-A to a large extent.

Note that the read access operations include sequential read and random read. The seek time is involved in the random read, which is slower than sequential read. Our C-Scheme and U-Scheme distribute read data more sparsely in each disk, thus we need more random read. In addition, there may be some I/O merge optimizations for sequential reads in OS. Therefore, the empirical improvement of recovery speed is not as high as the theoretical analysis, but it is still substantial to indicate that the recovery speed in single disk failure situation is significantly improved by the balanced read load.

VII. Conclusion

In this article, we have proposed two load-balance recovery generation algorithms. The first one actually provides an optimization for load balance in the condition of accessing minimal amount of data. The second algorithm minimizes the maximal amount of data read from a single disk, which maximizes load balance without the condition of reading minimal data. In these load-balance recovery generation algorithms, we generate the recovery schemes to minimize read load on the most loaded disk, leading to higher recovery speed. Theoretically, the two generative recovery schemes reduce up to 22.9% and 25.0%, and averages of 9.6% and 16.4% recovery time, respectively, from Khan's scheme, the state-of-the-art scheme for single failure situation. The experiment results from our implementations on a real disk array show that our two load-balance recovery schemes improve recovery speed by as high as 15.5% and 19.9%, respectively, compared with Khan's recovery scheme.

Acknowledgment

The authors would like to thank the reviewers for their constructive comments. We also thank Song Jiang for the kindly help on discussing the idea and the English writing, Si Ma for the help of experiments. This work is supported by the National Natural Science Foundation of China (Grant No. 60925006), the State Key Program of National Natural Science of China (Grant No. 61232003), the National High Technology Research and Development Program of China (Grant No. 2013AA013201), and Shanghai Key Laboratory of Scalable Computing and Systems, and the research fund of Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and Tsinghua University Initiative Scientific Research Program.

References

[1] S. Ghemawat, H. Gobioff and S.-T. Leung. "The Google File System," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, 2003, pp. 29-43.

[2] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong and S. Sankar, "Row-Diagonal Redundant for Double Disk Failure Correction," in *Proceedings of 3rd USENIX Conference on File and Storage Technologies (FAST'04)*, San Francisco, CA, USA, March 31-April 2, 2004.

[3] M. Blaum, J. Brady, J. Bruck and J. Menon, "EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures," *IEEE Transactions on Computer*, Vol. 44, No. 2, February 1995.

[4] C. Huang and L. Xu, "STAR: An Efficient Coding Scheme for Correcting Triple Storage Node Failures," in *Proceedings of 4th USENIX Conference on File and Storage Technologies (FAST'05)*, San Francisco, CA, USA, December, 2005, pp. 197-210.

[5] M. Holland. "On-Line Data Reconstruction in Redundant Disk Arrays," PhD thesis, Carnegie Mellon University, Apr. 1994.

[6] E. Pinheiro, W.-D. Weber and L. A. Barroso, "Failure trends in a large disk drive population," in *Proceedings of 5th USENIX Conference on File and Storage Technologies (FAST'07)*, San Jose, CA, February. 2007, pp. 17-28.

[7] A. L. Drapeau, K. W. Shirriff, J. H. Hartman, E. L. Miller, S. Seshan, R. H. Katz, K. Lutz, D. A. Patterson, E. K. Lee, P. M. Chen, and G. A. Gibson. "RAID-II: A High-Bandwidth Network File Server," in *Proceedings of 21st International Symposium on Computer Architecture (ISCA'94)*, Chicago, IL, USA, April 1994, pp. 234-244.

[8] F. J. MacWilliams and N. J. A. Sloane. "The Theory of Error-Correcting Codes." *New York: North-Holland*, 1977.

[9] C. Huang, M. Chen, and J. Li. "Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems," in *Proceedings of sixth IEEE International Symposium on Network Computing and Applications (NCA'07)*, Cambridge, MA, USA, July, 2007.

[10] K. M. Greenan, X. Li, and J. J. Wylie, "Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs," in *Proceedings of the 26th Symposium on Mass Storage Systems and Technologies (MSST' 10)*, Incline Village, Nevada, USA, May 3-7, 2010, pp. 1-14.

[11] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang, "Rethinking Erasure Codes for Cloud File Systems: Minimizing I/O for Recovery and Degraded Reads," in *Proceedings of 10th USENIX Conference on File and Storage Technologies (FAST'12)*, San Jose, CA, February 2012.

[12] L. Xiang, Y. Xu, J. Lui, and Q. Chang, "Optimal Recovery of Single Disk Failure in RDP Code Storage Systems," in *Proceedings of ACM SIGMETRICS'10*, June 14-18, 2010, New York, New York, USA.

[13] L. Xiang, Y. Xu, J. Lui, Q. Chang, Y. Pan, and R. Li, "A Hybrid Approach to Failed Disk Recovery Using RAID-6 Codes: Algorithms and Performance Evaluation," *ACM Transactions on Storage*, Vol. 7, No. 3, October 2011.

[14] 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, August 2008.

[15] J. M. Hafner, V. Deenadhayalan, T. Kanungo and KK Rao, "Performance Metrics for Erasure Codes in Storage Systems," *Technical Report*, RJ 10321, IBM Research, San Jose, CA, 2004.

[16] J. Hafner, V. Deenadhayalan, KK Rao, and J. Tomlin, "Matrix Methods for Lost Data Reconstruction in Erasure Codes", in *Proceedings of 4th USENIX Conference on File and Storage Technologies (FAST'05)*, San Jose, CA, February 2005, pp. 183-196.

[17] M. Blaum, P. Farrell, and H. van Tilborg, "Array codes," in Handbook of Coding Theory, V. Pless and W. Huffman, Eds. Amsterdam, The Netherlands: Elsevier Science B.V., 1998.

[18] M. Blaum, J. Bruck, and A. Vardy, "MDS array codes with independent parity elements," *IEEE Transactions on Information Theory*, Vol. 42, No. 2, pp. 529-542, March. 1996.

[19] J. Plank. "The RAID-6 Liberation Codes," In *Proceedings of 6th USENIX Conference on File and Storage Technologies (FAST'08)*, San Jose, February 2008.

[20] J. Plank. "The RAID-6 Liber8tion Code," *International Journal of High Performance Computing Appilications*, Vol. 23, No. 3, Fall 2009, pp. 242-251.

[21] M. Blaum and R.M. Roth, "On Lowest Density MDS Codes," *IEEE Transactions on Information Theory*, Vol. 45, No. 1, January 1999.

[22] X. Luo, J. Shu, and Y. Zhao, "Shifted Element Arrangement in Mirror Disk Arrays for High Data Availability during Reconstruction," in *Processings of the 41st International Conference on Parallel Processing (ICPP' 12)*, Pittsburgh, PA, USA, September 10-13, 2012, pp. 178-188.

[23] C. Jin, H. Jiang, D. Feng and L. Tian, "P-code: A new RAID-6 code with optimal properties," in *Proceedings of the 23rd International Conference on Supercomputing (ICS'09)*, Yorktown Heights, NY, June 2009, pp. 360-369.

[24] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?" in *Proceedings of 5th USENIX Conference on File and Storage Technologies (FAST'07)*, San Jose, CA, February. 2007, pp. 1-16.

[25] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy and J. Schindler, "An analysis of latent sector errors in disk drives," in *Proceedings of 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'07)*, San Diego, CA, June. 2007, pp. 289-300.

[26] A. Krioukov, L. N. Bairavasundaram, G. R. Goodson, K. Srinivasan, R. Thelen, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Parity Lost and Parity Regained," in *Proceedings of 6th USENIX Conference on File and Storage Technologies (FAST'08)*, San Jose, CA, February 2008, pp.127-141.

[27] B. Schroeder, S. Damouras, and P. Gill, "Understanding Latent Sector Errors and How to Protect Against Them," in *Proceedings of 8th USENIX Conference on File and Storage Technologies (FAST'10)*, San Jose, CA, February 2010, pp. 71-84.

[28] J. L. Hafner, V. Deenadhayalan, W. Belluomini, and K. K. Rao, "Undetected disk errors in RAID arrays," *IBM Journal of Research and Development*, Vol. 52 No. 4/5, July/September 2008, pp. 413-425.

[29] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia. "Above the Clouds: A View of Cloud computing," *Technical report*, EECS Department, University of California, Berkeley, February, 2009.

[30] J. Plank, J. Luo, C. D. Schuman, L. Xu and Z. W. O' Hearn, "A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries For Storage," in *Proceedings of 7th USENIX Conference on File and Storage Technologies (FAST'09)*, San Francisco, CA, USA, February, 2009.

[31] J. Schindler, S. W. Schlosser, M. Shao, A. Ailamaki, G. R. Ganger, "Atropos: A Disk Array Volume Manager for Orchestrated Use of Disks," in *Proceedings of 3rd USENIX Conference on File and Storage Technologies (FAST'04)*, San Francisco, CA, USA, March 31-April 2, 2004.