On the Speedup of Single-Disk Failure Recovery in XOR-Coded Storage Systems: Theory and Practice

Yunfeng Zhu†, Patrick P. C. Lee*, Yuchong Hu*, Liping Xiang†, and Yinlong Xu†
†University of Science and Technology of China, *The Chinese University of Hong Kong
{zyfl,xlping}@mail.ustc.edu.cn, pclee@cse.cuhk.edu.hk, ychu@inc.cuhk.edu.hk, ylxu@ustc.edu.cn

***Abstract*—Modern storage systems stripe redundant data across multiple disks to provide availability guarantees against disk failures. One form of data redundancy is based on XOR-based erasure codes, which use only XOR operations for encoding and decoding. In addition to providing failure tolerance, a storage system must also provide fast failure recovery to avoid data unavailability. We consider the problem of speeding up the recovery of a single-disk failure for arbitrary XOR-based erasure codes. We address this problem from both theoretical and practical perspectives. We propose a *replace recovery algorithm*, which uses a hill-climbing technique to search for a fast recovery solution, such that the solution search can be completed within a short time period. We further implement our replace recovery algorithm atop a parallelized architecture to justify its practicality. We experiment our replace recovery algorithm and its parallelized implementation on a networked storage system testbed, and demonstrate that our replace recovery algorithm uses less recovery time than the conventional approach.**

I. Introduction

We have witnessed different implementations of large-scale storage systems for various applications such as data centers, peer-to-peer storage, and cloud storage. Examples of such implementations include OceanStore [18], GFS [10], TotalRecall [1], and Dynamo [8]. In a large-scale storage system, data is distributed over a collection of *disks* (or more generally, physical storage devices). To ensure data availability, it is necessary to *tolerate* disk failures, which are common in large-scale storage systems [10]. Data availability can be achieved by keeping redundant data in multiple disks based on replication or erasure coding. One form of data redundancy is based on the *Maximum Distance Separable (MDS)* codes, which are defined by the parameters n and k. An (n, k) MDS code divides a data object into k equal-size fragments and encodes them into n fragments, such that any k out of the n encoded fragments can be used to reconstruct the original object (we call this the *MDS property*).

In addition to tolerating disk failures, it is also necessary to *recover* disk failures, so as to preserve the required redundancy level and avoid data unavailability. In this paper, we focus on the recovery of a single-disk failure, which occurs more frequently than a concurrent multi-disk failure in practice. Recovery of a single-disk failure can be achieved by retrieving data from existing surviving disks and reconstructing the lost data of the failed disk in a new disk. To minimize the overall recovery time and achieve fast recovery, one important objective is to minimize the amount of data being read from the surviving disks.

Here, we focus on the recovery problem for a family of special-purpose MDS codes called *XOR-based erasure codes*, in which encoding and decoding are purely based on XOR operations. Existing XOR-based erasure codes provide different redundancy levels that can tolerate double-disk failures (e.g., RDP [5], EVENODD [2], X-code [27]), triple-disk failures (e.g., STAR [15]), or a general number of failures (e.g., Cauchy Reed-Solomon (CRS) codes [3]). Recent studies have proposed recovery solutions to minimize the amount of data being read for different double-fault tolerant codes, including RDP [26], EVENODD [25], and X-code [28]. However, extending such results for the STAR and CRS codes, which can tolerate more than two disk failures, is non-trivial due to the very different data layouts. Also, existing results are mainly developed via theoretical analysis and simulation. It remains unclear regarding the practical performance of the recovery solutions when they are deployed in a real storage system. Thus, the motivation of this work is to develop a generalized recovery solution that applies to *arbitrary* XOR-based erasure codes, from both theoretical and practical deployment perspectives.

In this paper, we focus on speeding up the recovery of a single-disk failure for a class of XOR-based erasure codes. Our primary objective is to minimize the amount of data read from the surviving disks for recovery and hence the overall time of the recovery operation, while the recovery solution can be quickly determined. We propose a *replace recovery algorithm*, which uses a hill-climbing (greedy) approach [23] to optimize the recovery solution. It starts with a feasible recovery solution, and incrementally *replaces* the current solution with another one that reads less data. We validate that it provides near-optimal recovery for different variants of STAR and CRS codes. Also, it is shown to achieve polynomial complexity. Note that our replace recovery can be extended for the setting where disks are heterogeneous with different performance costs. This implies that our replace recovery can be applied in online mode based on the current performance costs of surviving disks, while existing enumeration recovery (e.g., [16]) is infeasible in doing so due to its exponential complexity.

We implement our replace recovery atop a parallelized, multi-core-based architecture, so as to justify its practicality in real deployment. We conduct experiments on a networked storage system testbed of different scales (with up to 21 disk nodes). We validate the recovery time improvement of our replace recovery over the conventional recovery approach

(which we define in Section II) for different XOR-based erasure codes. Also, even in the parallel recovery implementation, we still observe the recovery time reduction of our replace recovery. Unlike existing disk-based simulations [26], our testbed experiments capture the actual read/write performance using real storage devices.

The remainder of the paper proceeds as follows. Section II reviews related studies on single-disk failure recovery, and Section III proposes a simplified recovery model. Section IV presents our replace recovery algorithm. Section V shows our implementation of disk recovery based on parallelization. Section VI presents our experimental results. Section VII concludes this paper.

II. Background and Related Work

A. Background: XOR-based Erasure Codes

We first define the vocabularies based on [21]. We consider a storage system employing an XOR-based erasure code. It contains an array of n disks, in which k disks hold *data*, and the remaining $m = n-k$ disks hold coding information (which we call *parity*) encoded from the data. Each disk is partitioned into fixed-size *strips* with ω *symbols* each. A symbol can refer to a fixed-size data block (or chunk) depending on the implementation of the storage system. Each strip in a parity disk is encoded from a strip in each data disk. We call the collection of $n = k + m$ strips that encode together a *stripe*. A *parity set* is a set containing a parity symbol together with the data symbols encoded in the parity symbol. In reality, each stripe is encoded independently, and the data and parity strips are rotated among the disks for load balancing [21]. Thus, the definitions of the data (parity) disks may vary across stripes, depending on where the data (parity) strips are located.

Note that we require the XOR-based erasure code satisfy the MDS property (see Section I), such that the original data can be reconstructed from any k out of n surviving disks. In other words, the storage system can tolerate any $m = n - k$ concurrent disk failures.

We illustrate the above definitions via an example. We consider an XOR-based erasure code called RDP [5], which is a double-fault tolerant code (i.e., $m = 2$) that achieves optimality both in computations and I/Os. The RDP encoding is applied to each stripe, which is a two-dimensional array of size $(p-1)\times(p+1)$, where p is a prime number larger than 2. The first $(p-1)$ columns in the array store data information, while the last two columns store parity information. Figure 1 shows how RDP encoding works for $p = 5$, where $d_{i,j}$ is the i-th symbol in column j. The first four disks (Disks 0 to 3) are the data disks, while the last two disks (Disks 4 and 5) are the parity disks. Disk 4 contains all the *row* parity symbols, e.g., $d_{0,4}$ is the XOR's of symbols $d_{0,0}$, $d_{0,1}$, $d_{0,2}$, $d_{0,3}$. Disk 5 contains all the *diagonal* parity symbols, e.g., $d_{0,5}$ is the XOR's of symbols $d_{0,0}$, $d_{3,2}$, $d_{2,3}$, $d_{1,4}$. In addition to RDP, there are other examples of XOR-based erasure codes that are also double-fault tolerant, such as EVENODD [2] and X-Code [27].

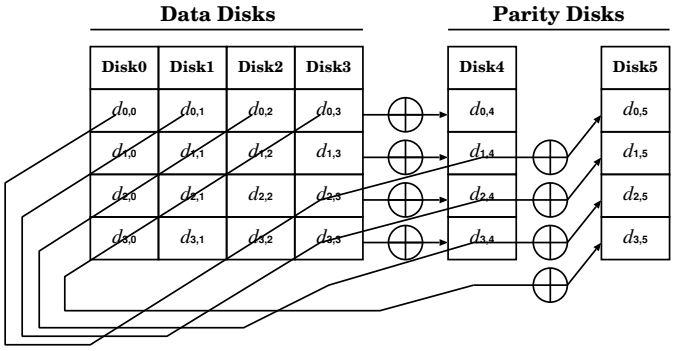


Fig. 1. RDP code with $p = 5$.

We note that every XOR-based erasure code can be represented by a *generator matrix* [21]. To illustrate, we consider the Cauchy Reed Solomon (CRS) codes [3], which can tolerate a general number of disk failures. Figure 2 shows the encoding mechanism of CRS codes for $k = 4$, $m = 2$ and $\omega = 3$. The idea is to multiply a $\omega n \times \omega k$ matrix of bits with a column vector of ωk data bits, so as to form a stripe of ωn data and parity bits.

Here, we use the term "disk" in a broad sense to refer to a general physical storage device (e.g., a storage server or a network drive) deployed in a storage system. A disk can fail and lose all stored data, and we aim to reconstruct the lost data in a new disk.

To recover disk failures, the *conventional recovery* approach downloads the data and parity symbols from k disks, such that the amount of information being downloaded per file is equal to the original file size. Note that this conventional recovery approach applies to *all* MDS codes (e.g., Reed-Solomon codes [22] and all XOR-based erasure codes) and any number of disk failures no more than m. However, the frequency of a single-disk failure is often higher than that of a concurrent multi-disk failure. This is generally true when the aggregated disk failure rate is lower than the disk recovery rate [26]. Thus, using the conventional recovery approach as a baseline, many studies (e.g., [16], [25], [26], [28]) propose more effective recovery solutions specifically for recovering a single-disk failure. In this paper, we focus on the following objective of optimizing the recovery solution, namely, we seek to *minimize the number of symbols being read (or I/Os) from the surviving disks for the single-disk failure recovery.*

In this paper, we focus on recovering a failed data disk. If the failed disk is a parity disk, then we assume that its recovery is identical to its corresponding encoding process [26]. Thus, in our discussion, by a single-disk failure, we mean the failure of a single data disk.

Before we discuss the recovery solutions for XOR-based erasure codes, we point out that *regenerating codes* [9] have recently been proposed to minimize the recovery bandwidth in distributed storage systems. The idea is that surviving storage nodes compute and transmit linear combinations of their stored data during failure recovery. On the other hand, in XOR-based erasure codes, we do not require storage nodes (or disks) be equipped with computational capabilities.

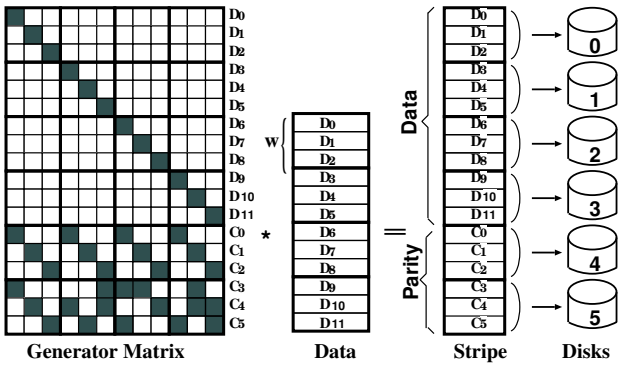


Fig. 2. CRS for $k = 4$, $m = 2$ and $\omega = 3$.

We now overview the related work on the recovery of single-disk failures in XOR-based erasure codes. We classify existing recovery solutions into two families, namely *hybrid recovery* and *enumeration recovery*.

B. Hybrid Recovery

Some studies propose optimal recovery schemes for a single-disk failure specifically for double-fault tolerant XOR-based erasure codes, with an objective of minimizing the number of read symbols. Xiang *et al.* [26] study the optimal recovery of a single-disk failure for RDP, and the recovery solution reduces the number of read symbols by 25% compared to conventional recovery. Wang *et al.* [25] consider a similar single-failure recovery problem in a distributed storage system that uses EVENODD, and prove the same 25% improvement as in RDP. Xu *et al.* [28] also propose optimal single-disk failure recovery for X-code. These studies use the same core idea of *hybrid recovery* for RDP in [26]. Thus, in the following, we use RDP to explain the hybrid recovery idea.

To recover a single-disk failure, the conventional recovery approach in essence recovers each lost symbol in the failed disk *independently*. Specifically, if a data disk is failed, we only use the row parity symbols together with all other surviving data symbols in each row to recover each lost symbol of the failed disk. We illustrate the idea of the conventional recovery approach using Figure 1. For example, if Disk 1 fails, one can read $d_{0,0}, d_{0,2}, d_{0,3}, d_{0,4}$ to recover $d_{0,1}$ of Disk 1. Thus, the total number of read symbols for repairing Disk 1 is 16. In contrast, hybrid recovery [26] uses a *combination of row and diagonal parity sets* of data and parity symbols to repair Disk 1, such as:

- $d_{1,0}, d_{3,3}, d_{2,4}, d_{1,5}$ to recover $d_{0,1}$
- $d_{0,2}, d_{2,0}, d_{3,4}, d_{2,5}$ to recover $d_{1,1}$
- $d_{2,0}, d_{2,2}, d_{2,3}, d_{2,4}$ to recover $d_{2,1}$
- $d_{3,0}, d_{3,2}, d_{3,3}, d_{3,4}$ to recover $d_{3,1}$

Since the above 16 symbols contain four overlapping symbols $d_{3,3}, d_{2,4}, d_{2,0}, d_{3,4}$, the total number of read symbols for repairing Disk 1 is reduced to 12 (i.e., by 25%). Thus, the core idea of hybrid recovery is to find the set of *maximum-overlapping* symbols to minimize the number of read symbols for recovery. Note that this hybrid recovery idea also provides *optimal* recovery solutions for EVENODD [25] and X-code [28].

C. Enumeration Recovery

For general XOR-based erasure codes, one way to achieve optimal recovery is to enumerate all recovery possibilities based on the generator matrix. Such an approach, which we call *enumeration recovery*, applies to *any* XOR-based erasure code and is studied in [16] (an extended version appeared in [17], which considers how to minimize the amount of data being read in degraded read operations as well). Similar approaches are also proposed for non-MDS codes [11].

Here, we use the example of CRS in Figure 2 to explain how enumeration recovery works. To recover a data disk failure in CRS, the conventional recovery approach may select the parity symbols in the first parity disk together with all surviving data symbols. For example, if Disk 0 is failed and data symbols D_0, D_1, D_2 are lost, then we can read parity symbols C_0, C_1, C_2 and data symbols $D_3, D_4, ..., D_{11}$ for recovery. Thus, the total number of symbols being read is 12. Alternatively, we can reduce the number of symbols by enumerating the *recovery equations* [11], [16]. A recovery equation is a collection of symbols in a stripe whose corresponding rows in the generator matrix sum to zero. An example of a recovery equation is D_0, D_3, D_6, D_9, C_0. Obviously, we can recover any lost symbol in one recovery equation from all other surviving symbols in the recovery equation. For example, if D_0 is lost, then the remaining symbols D_3, D_6, D_9, C_0 can be used to recover D_0. Suppose that we enumerate all the recovery equations for the generator matrix. Then we can formulate the optimal recovery problem for Disk 0 as follows: Given three sets E_0, E_1, E_2, where E_i is the set of all the recovery equations for lost symbol D_i ($0 \leq i \leq 2$), we select one equation e_i from each set E_i such that the number of symbols in the union of all e_i's is minimized. In this example, we obtain an optimal solution:

- e_0: $D_0, D_5, D_6, D_7, D_{10}, C_3$,
- e_1: $D_1, D_4, D_7, D_{10}, C_1$,
- e_2: $D_2, D_5, D_8, D_{11}, C_2$.

The total number of symbols in the union of e_0, e_1, e_2 except D_0, D_1, D_2 is equal to the number of read symbols for recovery, which is reduced to 10. However, we point out that the number of recovery equations grows *exponentially* with the number of disks. In general, the problem of finding the optimal recovery solution that minimizes the number of read symbols is NP-hard [16].

III. Recovery Model

A. Motivation

While enumeration recovery can find the optimal recovery solution for a single-disk failure in any XOR-based erasure codes, it has a very high computational overhead. One can show that enumeration recovery has a search space of up to $2^{m\omega}$ recovery equations, where m denotes the number of tolerable failures and ω is the strip size. To solve for the optimal recovery solution, one enumeration recovery implementation is to construct a weighted graph containing nodes that represent recovery equations, and then find the shortest path on the

graph [16]. Depending on the implementation of the shortest-path algorithm, the size of the graph can be potentially huge, as there are an exponential number of nodes (i.e., recovery equations). In Section IV-C, we show via simulations the expensive running time performance of enumeration recovery using commodity hardware. In the following, we describe several scenarios where enumeration recovery is *infeasible* to deploy.

Limited hardware resources. Since the number of recovery equations increases exponentially with m and ω, enumeration recovery may consume substantial resources for enumerating all recovery equations when m and ω are large, which correspond to the codes with a large strip size and a higher level of fault-tolerance, respectively. For example, for the STAR code [15], we have $m = 3$ and $\omega = p-1$. If $p = 13$, then at most 2^{36} recovery equations need to be enumerated. In Section IV-C, our simulations show that with $m\omega \geq 20$, our enumeration recovery implementation deployed on commodity hardware cannot be finished within 13 days. Although one can use parallelization to speed up the enumeration process, the computational complexity remains exponential. Therefore, enumeration recovery becomes computationally infeasible when the available hardware resources are limited.

Remote recovery scenario. Nowadays, data recovery of a failed storage system can be outsourced to third-party companies (e.g., DataRecovery [6] and DataTech Labs [7]). However, the recovery service providers usually do not know *in advance* the coding scheme and parameters of the failed storage system. It is also difficult, while not impossible, to determine in advance the optimal solutions for all possible coding schemes and parameters. Thus, finding the optimal recovery solution for a failed storage system can be viewed as an on-demand decision task.

Online recovery scenario. Enumeration recovery aims to minimize the number of read symbols for recovery. We can see that the optimal recovery solution is *deterministic*, meaning that for a given failed disk, a coding scheme, and the corresponding parameters, the set of symbols being read from surviving disks is fixed. Thus, if m and ω are small, one can first determine the optimal solutions *offline* for each possible failed disk. However, practical storage systems are typically composed of storage devices with heterogeneous capabilities (e.g., processing power, connectivity bandwidth) [19], [20]. In order to recover the failed disk effectively, it is intuitive to retrieve fewer (more) data symbols from the surviving disks with lower (higher) capabilities. Disk capabilities may vary, depending on the current usage load of the storage system. Thus, it is important to find the optimal recovery solution *online* based on the current disk capabilities. However, even it is possible to modify the enumeration recovery approach to account for the heterogeneous setting, its exponential complexity makes the online approach infeasible to capture the current disk capabilities in a timely manner.

B. Simplified Recovery Model

Given the high computational complexity of enumeration recovery, we are motivated to find a new recovery approach that can achieve effective recovery performance in a computationally feasible manner. We note that the main bottleneck of enumeration recovery is the huge search space of recovery equations. To narrow down the search space, we make one observation.

Observation: Consider a storage system using an XOR-based erasure code with parameters (n, k, m, ω). Suppose that a strip of ω symbols $D_0, D_1, \ldots, D_{\omega-1}$ is lost. Let E_i be the set of all recovery equations for each lost symbol D_i $(0 \leq i \leq \omega - 1)$. Then it is *likely* that there exists an optimal recovery solution satisfying the corresponding recovery equations $e_0, e_1, \ldots, e_{\omega-1}$ (selected from $E_0, E_1, \ldots, E_{\omega-1}$, respectively) such that this solution has exactly ω parity symbols.

Clearly, there must be a *feasible* recovery solution with exactly ω parity symbols, for example, by using the ω parity symbols in any parity disk and the data symbols in the $k-1$ data disks based on the MDS property (see Section I). Our observation here is to address the optimality condition. Our intuition is that the failure of a single data disk corresponds to the loss of a strip of ω data symbols per stripe, so we need only ω parity symbols to recover the lost ω data symbols. If we use more than ω parity symbols, then it is likely to involve more data symbols to be read. In some situations, we can use exactly ω parity symbols to deduce the optimal recovery solution. To illustrate, in Section II-B, the optimal recovery solution for RDP contains exactly $\omega = 4$ parity symbols $d_{2,4}$, $d_{1,5}$, $d_{3,4}$, and $d_{2,5}$; in Section II-C, the optimal solution contains $\omega = 3$ parity symbols C_1, C_2, and C_3.

Actually, we can always find an optimal recovery solution that has exactly ω parity symbols for RDP codes, as proven in [26], although it remains open if we can always find an optimal recovery solution with ω parity symbols for general codes. Nevertheless, using this observation as our search criterion suffices for practical purposes, based on our simulations (see Section IV) and experiments (see Section VI). We now formulate a simplified recovery model that solves the single-disk failure recovery problem for any XOR-based erasure codes.

Simplified Recovery Model. To recover a failed disk, we aim to choose a collection of parity symbols (together with the corresponding surviving data symbols that encode the parity symbols) to regenerate a strip of ω data symbols per stripe, subject to:

1) The collection of parity symbols is of size ω symbols.
2) The collection of parity symbols (and their encoding data symbols) suffices to resolve the ω lost data symbols.
3) The number of all data symbols encoded in the ω parity symbols is minimum.

The above simplified recovery model now reduces the solution space to the collections of parity symbols of a fixed size (instead of arbitrary collections of parity symbols). This

significantly reduces the computational complexity of recovery as compared to enumeration recovery.

Objectives. Based on the simplified recovery model, our goal is to design a recovery algorithm that achieves the following objectives:

1) *Search efficiency.* The algorithm finds a recovery solution with polynomial complexity.
2) *Effective recovery performance.* The number of read symbols of the resulting recovery solution should be close to that of the optimal solution.
3) *Adaptable to heterogeneous disk capabilities.* The recovery algorithm can be easily converted to handle heterogeneous disk capabilities, and hence can promptly return an effective recovery solution in the online recovery scenario.

IV. Replace Recovery Algorithm

Our simplified recovery model states that there exists a recovery solution that contains exactly ω parity symbols for regenerating ω lost data symbols for each stripe in a single-disk failure. However, it remains computationally expensive to search for the "best" collection of ω parity symbols (out of $\binom{m\omega}{\omega}$ possible candidates) in general. In this section, we propose a computationally efficient *replace recovery algorithm* that seeks to minimize the number of read symbols for single-disk failure recovery, and the algorithm is applicable for *any* XOR-based erasure codes.

The idea of our replace recovery algorithm is as follows. Let $\mathcal{P}_i$ be the set of parity symbols in the ith parity disk, where $1 \leq i \leq m$. Let $\mathcal{X}$ be the collection of ω parity symbols used for recovery, and $\mathcal{Y}$ be the collection of parity symbols that are considered to be included in $\mathcal{X}$. First, we initialize $\mathcal{X}$ with the ω parity symbols of $\mathcal{P}_1$. It can be easily shown that $\mathcal{X}$ can resolve the ω lost data symbols with other $k-1$ surviving disks, due to the MDS property (see Section I). Then we set $\mathcal{Y}$ to be the collection of parity symbols in $\mathcal{P}_2$. Now we *replace* "some" parity symbols in $\mathcal{X}$ with the same number of parity symbols in $\mathcal{Y}$, such that $\mathcal{X}$ still resolves the ω lost data symbols, while reducing the *most* number of read symbols. We repeat this by resetting $\mathcal{Y}$ with $\mathcal{P}_3, \cdots, \mathcal{P}_m$. Finally, we obtain the resulting $\mathcal{X}$. The parity symbols in $\mathcal{X}$, as well as the corresponding encoding data symbols, are retrieved for recovery. In essence, our replace recovery algorithm uses a hill-climbing (greedy) approach [23] to optimize the solution.

Before presenting our replace recovery algorithm, we need a primitive function that determines if $\mathcal{X}$ is *valid* to resolve the ω data symbols after being replaced with other parity symbols in $\mathcal{Y}$. For each parity symbol, we define an ω-bit encoding vector that specifies how the strip of *lost* data symbols is encoded to the parity symbol. The ith bit (where $1 \leq i \leq \omega$) of the encoding vector of a parity symbol is set to 1 if the ith data symbol is encoding to that parity symbol, or 0 otherwise. For example, referring to the CRS example in Figure 2, the encoding vectors for the parity symbols C_0 and C_1 are (1,0,0) and (0,1,0), respectively. We say $\mathcal{X}$ is valid if the encoding vectors for the ω parity symbols in $\mathcal{X}$ satisfy that: (i) they cover all the ω lost data symbols and (ii) they are linearly independent (e.g., checked by Gaussian Elimination).

Algorithm 1 Replace Recovery Algorithm

```
1: Initialize X = P_1
2: for i = 2 to m do
3:    Set Y = P_i
4:    for each parity symbol Y in Y do
5:       Set f = false;
6:       for each parity symbol X in X do
7:          Set X' = X - {X} + {Y}
8:          if X' is valid and reduces the number of read symbols
            of X then
9:             Compute R_{X,Y} = number of read symbols of X'
10:            Set f = true;
11:         end if
12:      end for
13:   end for
14:   if f == true then
15:      Find (X', Y') = argmin_{(X,Y)} R_{X,Y}
16:      Replace X' with Y' in X
17:      Remove Y' from Y
18:   end if
19:   Repeat Steps 4-18 until Y is empty or f == false
20: end for
21: Return X
```

A. Algorithm Design

Algorithmic details. Algorithm 1 shows the core design of our replace recovery algorithm. We initialize $\mathcal{X}$ with $\mathcal{P}_1$ (Step 1). Then we consider $\mathcal{Y} = \mathcal{P}_i$ ($2 \leq i \leq m$) (Step 3). For each parity symbol in $\mathcal{Y}$, we compute the number of read symbols by replacing every parity symbol in $\mathcal{X}$ (Steps 4-13). Note that we only consider the replacement that is valid and can reduce the number of read symbols of $\mathcal{X}$ (Step 8). We then find the replacement with the minimum number of read symbols, and replace the old parity symbol X' in $\mathcal{X}$ with the new parity symbol Y' in $\mathcal{Y}$ and remove Y' from $\mathcal{Y}$ (Steps 14-18). We repeat Steps 4-18 until $\mathcal{Y}$ is empty or there is no reduction by any replacement (Step 19).

Example. We illustrate Algorithm 1 via the CRS example in Figure 2. First, we initialize $\mathcal{X}$ with $\{C_0, C_1, C_2\}$ of the first parity disk (Disk 4). Note that the number of read symbols of $\mathcal{X}$ is 12 per stripe (i.e., the number of all data symbols in a stripe). Now we consider $\mathcal{Y} = \{C_3, C_4, C_5\}$. We can verify that C_3, C_4, and C_5 can only replace C_0, C_1, and C_2, respectively, to make $\mathcal{X}$ valid. All replacements give the number of read symbols equal to 10 symbols (the maximum reduction achievable). Let us replace C_0 by C_3 (i.e., $\mathcal{X}$ = $\{C_3, C_1, C_2\}$). We now consider again $\mathcal{Y} = \{C_4, C_5\}$, but it does not give any reduction of the number of read symbols. Since m = 2, we finish Algorithm 1 and return $\mathcal{X}$ = $\{C_3, C_1, C_2\}$ for recovery. Note that this is also an optimal solution.

Complexity. We now evaluate the complexity of Algorithm 1 for searching $\mathcal{X}$. We can easily see that the for-loop of Steps 4-13 takes $O(\omega^2)$ time, and Step 19 will repeat the for-loop for at most ω times. We iterate Steps 4-19 for $m-1$ parity disks, so the total search complexity is $O(m\omega^3)$, which is polynomial-time.

Algorithmic enhancements. We can improve the accuracy of Algorithm 1 by searching for more candidate collections of parity symbols for $\mathcal{X}$. This increases the likelihood for our replace search to achieve the optimal point. Here, we propose two enhancements that increase our search space, while maintaining the polynomial complexity.

- *Multiple rounds.* In Step 1, we only use $\mathcal{P}_1$ for the initialization of $\mathcal{X}$. We can repeat Algorithm 1 for additional $m-1$ rounds by using $\mathcal{P}_i$ $(2 \leq i \leq m)$ for initialization.
- *Successive searches.* In Step 2, after we consider $\mathcal{P}_i$, we re-consider the previously considered $i-2$ parity symbol collections $\mathcal{P}_2, \cdots \mathcal{P}_{i-1}$, as they might provide better results. We can replace the for-loop in Step 2 with successive searches as: $\mathcal{P}_2, \mathcal{P}_3, (\mathcal{P}_2), \mathcal{P}_4, (\mathcal{P}_2, \mathcal{P}_3), \cdots, \mathcal{P}_i, (\mathcal{P}_2, \cdots, \mathcal{P}_{i-1}), \cdots$ (this technique is also called *univariate search*).

The successive searches increase the iterations of Step 2 by m times, and the multiple initializations iterate the whole process by another m times. Thus, the search complexity increases to $O(m^3\omega^3)$, which remains polynomial-time.

Generalized recovery cost. Algorithm 1 is designed to minimize the number of read symbols during recovery. We now show how it can be adapted for the scenario where disks have heterogeneous capabilities. Suppose that for the recovery solution $\mathcal{X}$ of failed disk k, the recovery operation reads y_i symbols from disk D_i $(i \neq k)$. Let c_i be the unit recovery cost of downloading a single symbol from disk D_i. We can define the *generalized recovery cost* $C = \sum_{i=0, i\neq k}^{n} c_i y_i$, and substitute the computation of the number of read symbols with that of C in Algorithm 1 (in Steps 8-9). Note that the generalized recovery cost can be tailored for different optimization objectives depending on the definition of the unit cost c_i. For example, if we set c_i to be the inverse of the connectivity bandwidth of disk D_i, then the optimization objective denotes the total recovery time of reading all symbols from other surviving disks one-by-one. Note that the unit cost c_i can be obtained during the storage system setup or be measured online according to the current usage conditions.

In this paper, we mainly focus on minimizing the number of symbols read (i.e., $c_i = 1$ for all i). Recent work [29] discusses the failure recovery for XOR-based erasure codes on heterogeneous storage devices and proposes a cost-based heterogeneous recovery scheme for two RAID-6 (double-fault tolerant) codes RDP and EVENODD. The idea of the recovery scheme is to eliminate the search for the recovery solutions that are known to make no improvements to the resulting recovery performance, thereby improving the efficiency of the solution search process. Note that the search space of the recovery scheme in [29] remains exponential with respect to the number of disks in the system, and it is still an open issue of how to extend the results of [29] for general XOR-based erasure codes. In future work, we plan to evaluate the performance of our replace recovery approach for other optimization objectives, and explore its applicability for general XOR-based erasure codes.

B. Evaluation of Recovery Performance

We now evaluate via simulations the performance of our proposed replace recovery algorithm. Our goal is to show that the number of read symbols returned by replace recovery is very close to that of enumeration recovery. Here, we mainly focus on the STAR and CRS codes, which can tolerate three and a general number of concurrent disk failures, respectively. While replace recovery is applicable for double-fault tolerant codes, we do not consider them as their optimal recovery solutions have been discussed in [26], [25], [28]. Here, our replace recovery algorithm is based on Algorithm 1, with the algorithmic enhancements enabled (see Section IV-A).

Let us first consider STAR [15]. For any prime number p, STAR is composed of $p+3$ disks, where the first p columns are data disks and the remaining three columns are parity disks. We first present a theorem that specifies the lower bound of the optimal recovery solution for STAR. The proof is in Appendix.

Theorem 4.1: The minimum number of read symbols for a single-disk failure recovery for STAR is lower bounded by $(\frac{2}{3}p^2 - p)$ (symbols per stripe).

We use simulations to compare both conventional and replace recoveries with the lower bound. Figure 3 shows the results. We observe that replace recovery is very close to the lower bound for $p < 40$. We also verify that replace recovery can give a result lower than $0.69p^2$ for $p < 1000$.

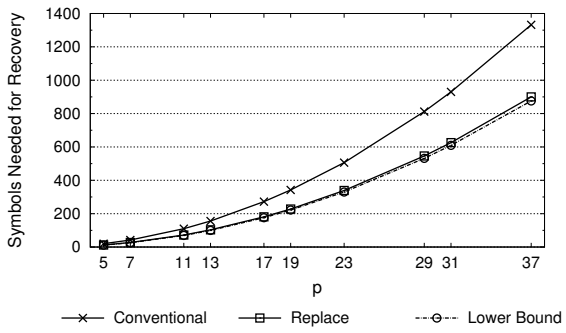
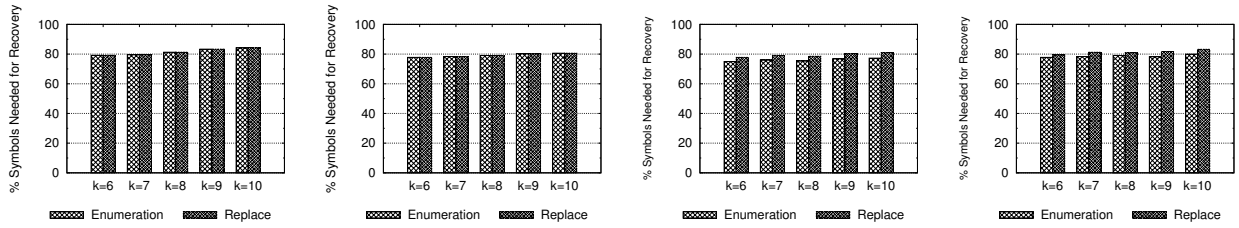


Fig. 3. Number of symbols per stripe needed for a single-disk failure recovery in STAR.

We now consider CRS. We use simulations to evaluate the savings of replace recovery over conventional recovery in terms of the number of read symbols. We also evaluate the savings of enumeration recovery to verify the accuracy of replace recovery. Figure 4 presents the results for different combinations of m, ω, and k. For $m = 2$, we observe that replace recovery achieves the same optimal result as in enumeration recovery, and the savings over conventional recovery are 15.75-25.00%. For $m = 3$, replace recovery achieves near-optimal performance. When compared to conventional recovery, the savings of replace recovery are 16.25-22.22%, while the savings of enumeration recovery are 19.75-25.00%.

C. Evaluation of Search Performance

We now evaluate via simulations the search performance of both enumeration and replace recoveries using commodity



CRS(m=2, ω=4) CRS(m=2, ω=5) CRS(m=3, ω=4) CRS(m=3, ω=5)

Fig. 4. Number of symbols (per stripe) needed for single-disk failure recovery for enumeration and replace recoveries in CRS with m=2 and m=3.

hardware configurations. Our goal is to show that enumeration recovery becomes infeasible for large parameters, while replace recovery can be completed with significantly less computational time.

Our evaluation is conducted on a Linux desktop computer running with 3.2GHz CPU and 2GB RAM. As discussed in Section III, in enumeration recovery, the maximum number of recovery equations being enumerated is $2^{m\omega}$. Here, we consider CRS, which allows us to configure different values of m and ω.

Table I shows the search time for different CRS variants when enumeration and replace recoveries are used. The search time of enumeration recovery increases exponentially with the number of recovery equations. For example, for $k = 10$, $m = 3$, and $\omega = 6$, it takes more than 18 hours to find the optimal solution for recovering one failed disk; for $k = 12$, $m = 4$, and $\omega = 5$, the search cannot be finished within 13 days. On the other hand, the search time of replace recovery can be completed within 0.5 seconds for all the CRS variants that we consider.

TABLE I
SEARCH PERFORMANCE OF ENUMERATION AND REPLACE RECOVERIES.

CRS(k,m,ω)	$m\omega$	Time (Enumeration)	Time (Replace)
CRS(10,3,5)	15	$6m32s$	0.08s
CRS(12,4,4)	16	$17m17s$	0.09s
CRS(10,3,6)	18	$18h15m17s$	0.24s
CRS(12,4,5)	20	$13d18h6m43s$	0.30s

V. DESIGN AND IMPLEMENTATION

In this section, we present the design and implementation for our replace recovery algorithm. We implement it on a parallel architecture that can scale the recovery performance in practice.

A. Recovery Thread

We implement the recovery operation with a *recovery thread*, a process that interconnects all disks and coordinates all data reads and writes with the disks. A recovery thread can be viewed as an intermediate controller process that relays data among the disks. To recover a single-disk failure, the recovery thread performs three steps: (i) reading data from the surviving disks, (ii) reconstructing the lost data, and (iii) writing the reconstructed data to a new disk. Note that the recovery thread implementation only requires the disks support standard read/write functions.

B. Parallel Recovery Architecture

For further performance improvements, we implement a *parallel recovery architecture* that parallelizes the recovery operation via multi-threaded and multi-server designs. Figure 5 shows the architecture. In Section VI, we show that replace recovery outperforms conventional recovery in both single-threaded and parallel implementations.

Multi-threaded recovery. As modern architectures shift toward multi-core, parallelizing the recovery process with multiple recovery threads becomes possible within a multi-core server. Two multi-threaded techniques have been proposed for recovery in RAID systems [12]: *disk-oriented reconstruction (DOR)* and *stripe-oriented reconstruction (SOR)*. We can implement the DOR and SOR as follows. In DOR, we create $n > 1$ recovery threads, each associated with one disk, where n is the total number of disks. Each of the $n-1$ threads reads data chunks from its associated surviving disk, and the remaining thread reconstructs and writes the resulting data chunks to the new disk. In contrast, in SOR, we create multiple recovery threads, each associated with a group of stripes (note that each stripe spans all the disks). Each thread recovers its own group of stripes. Since DOR needs to manage many threads if the number of disks increases, in our implementation, we use SOR as our multi-threaded recovery strategy.

Multi-server recovery. To further boost the recovery performance, we deploy a cluster of recovery servers to extend the scale of parallelism. The cluster is composed of one *dispatcher* and multiple *executers*, as shown in Figure 5. The dispatcher splits the whole recovery operation into different independent *tasks*, each of which corresponds to the recovery of a group of stripes. It then assigns each task to a different executer. An executer notifies the dispatcher after completing its task, so that it can be assigned the next task. Each assigned task can be further decomposed into different groups of stripes, each being processed by a recovery thread based on the SOR approach. Note that in actual deployment, the executors are deployed in different servers, while the dispatcher may be deployed in one of the executor servers, given that its dispatching workload is lightweight in general.

C. Implementation

We implement both conventional and replace recoveries for the following XOR-based erasure codes: RDP [26], EVENODD [25], X-Code [28] (all of which are double-fault tolerant), STAR (triple-fault tolerant), and Cauchy Reed-Solomon

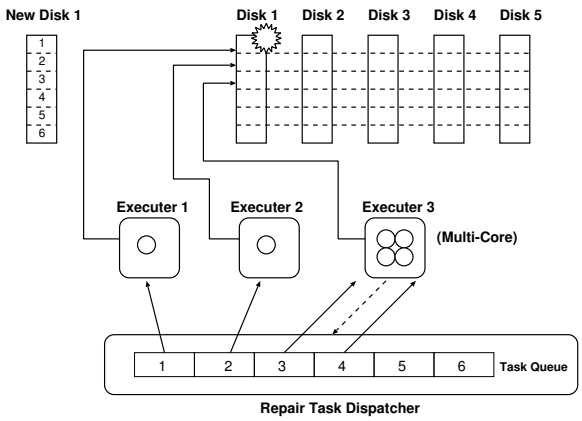


Fig. 5. Our parallel recovery architecture for scaling the recovery performance.

(CRS) codes (multi-failure tolerant). While there are many existing double-fault tolerant XOR-based erasure codes, we pick RDP, EVENODD, and X-Code because their optimal recovery solutions have been found (see [26], [25], [28], respectively) and we can compare the solutions of our replace recovery approach with their respective optimal results. Note that the numbers of disks (i.e., n) being used for RDP, EVENODD, X-Code, and STAR are mainly determined by a prime number p. For clarity, Table II summarizes the configurations of the codes that we have implemented based on p.

TABLE II
CONFIGURATIONS OF THE CODES THAT WE CONSIDER.

Code	n	k	m	Remarks
RDP	$p+1$	$p-1$	2	prime $p > 2$
EVENODD	$p+2$	p	2	prime p
X-Code	p	$p-2$	2	prime p
STAR	$p+3$	p	3	prime p
CRS	For general $n = k + m$			

We set the unit of XOR of encoding/decoding to be four bytes long, so as to make our implementation compatible with both 32-bit/64-bit machines [21]. The stripe unit in CRS is a ω-bit word, where ω must be large enough so that the total number of disks n is at most 2^ω. Note that in CRS, ω does not need to be a multiple of the machine word length, but should be as small as possible. Here, we select $\omega = 6$ for CRS, meaning that we allow at most 64 disks in the system. Note that from our simulations, we also see the improvements of our replace recovery algorithm for different values of ω (see Figure 4).

In our implementation, we treat each symbol as a chunk, which could be of large size in general. Unlike typical file systems that use small block sizes (for example, the default block size in Linux file systems is 4KB), operating on large chunks is typical in distributed storage systems (e.g., GFS [10] uses the chunk size 64MB). We evaluate how the recovery performance is influenced by the chunk size in our experiments (see Section VI).

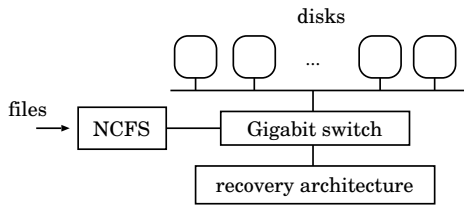


Fig. 6. Our testbed topology.

VI. EXPERIMENTS

We now conduct testbed experiments on the single-disk recovery approaches for different XOR-based erasure codes. Note that replace recovery seeks to minimize the number of read symbols during recovery, and hence the overall time to complete the recovery operation. The goal of our testbed experiments is to demonstrate that our replace recovery algorithm reduces the recovery time over conventional recovery in both single-threaded and parallelized recovery implementations. Unlike disk simulations [26], our experimental results capture the actual I/O performance with real storage nodes.

A. Methodology

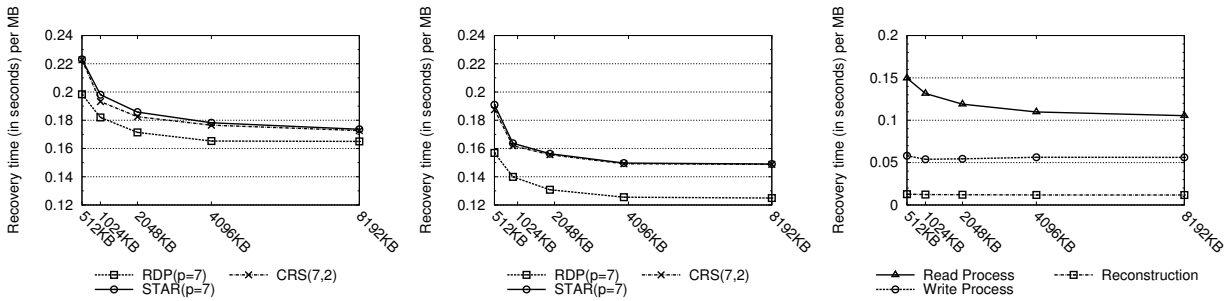
Our experimental testbed is built on an open-source networked storage system called NCFS [14], a networked storage system that interconnects, over a network, different physical storage devices, each of which corresponds to a disk as being considered in our experiments. It transparently stripes data across all disks according to the respective coding scheme. We integrate different coding schemes into NCFS. We deploy our recovery architecture (see Section V) alongside NCFS and the disks, while our architecture implements both conventional and replace recoveries.

Figure 6 shows our testbed. We interconnect all physical entities over a Gigabit Ethernet switch. Both NCFS and the recovery architecture communicate with the storage devices via the ATA over Ethernet protocol [13]. In Experiments 1 and 2, we use only a single recovery thread in our recovery architecture (i.e., without using parallelization), which is deployed on a Linux-based server equipped with an Intel Quad-Core 2.66GHz CPU and 4GB RAM. In Experiment 3, we evaluate the recovery performance using parallelization.

Our storage system consists of a cluster of (logical) disks, each represented by a physical storage device. We experiment different numbers of disks in the cluster, with at most 21 disks depending on the parameters chosen for the coding schemes. The cluster of disks is formed by the combination of the following three storage device types:

- Pentium 4 PCs, each equipped with a 100-Mbps Ethernet interface,
- network attached storage (NAS) devices, each equipped with a 1-Gbps Ethernet interface, and
- Intel Core i5-2400 Quad-Core servers, each also equipped with a 1-Gbps Ethernet interface.

We are mainly interested in the metric *recovery time* (per MB of data being recovered) needed to perform a recovery operation. We obtain the average recovery time as follows.



(a) Conventional recovery (b) Replace recovery (c) Time breakdown for STAR($p = 7$)

Fig. 7. Experiment 1 - Impact of chunk size.

We write 1GB of data into each disk via NCFS using the specified coding scheme (i.e., if there are n disks, then we write a total of nGB). Then NCFS will stripe the data across the disks. We disable one of the data disks in the storage system to make it resemble a failed disk. We then perform the recovery operation, that is, reading data from surviving disks, reconstructing data, and writing data to a new disk that we prepare. The recovery operation is done three times. We repeat this for all data disks, and obtain the overall average. For instance, referring to RDP ($p = 5$) in Figure 1 (see Section II-B), there are four data disks. Thus, we run a total of 4×3 recovery operations, and average the recovery times over the 12 runs.

B. Results

Experiment 1: Impact of chunk size. We first evaluate how different chunk sizes influence the recovery performance. We consider different chunk sizes, ranging from 512KB to 8MB. We focus on various coding schemes that can tolerate different numbers of failures, including RDP($p = 7$), STAR($p = 7$), and CRS($k = 7$, $m = 2$).

Figures 7(a) and 7(b) show the recovery time (per MB) for different chunk sizes using conventional and replace recoveries, respectively. We observe that as the chunk size increases, the recovery time decreases. The reason is that given the same amount of data, the number of accesses decreases for a larger chunk size. The rate of decrease diminishes as the chunk size further increases, so we expect that the recovery time stabilizes for a large enough chunk size. As shown in Figures 7(a) and 7(b), the decrease trend applies to both conventional and replace recoveries. In fact, similar results are observed for all coding schemes and we omitted the results here in the interest of space.

As described in Section V-A, a recovery operation consists of three main parts. In order to evaluate the contribution of each part to the whole recovery performance, we provide a performance breakdown for the recovery operation. Here we take conventional recovery for STAR($p = 7$) as an example. Figure 7(c) shows the breakdown (i.e., reading data from surviving disks, reconstructing lost data, and writing data to a new disk) for STAR. We observe that the reconstruction part contributes less than 10% in STAR, and we believe that the XOR-based encoding/decoding operations in STAR have minimal computational overhead. More importantly, the read part contributes over 60% of the overall recovery time for all chunk sizes. Note that if the number of disks increases, then more data will be read from surviving disks (see Figure 3 in Section IV), so it is expected that the read part will contribute a larger proportion to the overall recovery time. To summarize, the experiment results show that in order to reduce the overall recovery time, it is critical to minimize the number of read symbols, and hence the amount of data read from surviving disks.

In the following experiments, we fix the chunk size to be 512KB, which been chosen by some existing storage systems (e.g., OBFS [24]). Although the 512KB chunk size gives the maximum (worst) recovery time in general, our main goal is to compare the relative recovery performance of conventional and replace recoveries, rather than their actual recovery performance. Using a smaller chunk size enables us to divide the data into more stripes, so that we can better evaluate the recovery performance in SOR-based multi-threaded recovery (see Experiment 3).

Experiment 2: Recovery time performance. We now compare the recovery times of different XOR-based erasure codes using both conventional and replace recoveries, assuming a single recovery thread is used.

Figure 8 shows the results for various double-failure tolerant coding schemes, including RDP, EVENODD, X-Code, CRS(k, $m = 2$). We note that replace recovery reduces the recovery time of conventional recovery in *all* cases. As shown in Experiment 1, the data read part contributes the largest proportion of the recovery time. Since replace recovery aims to reduce the amount of data being read from surviving disks, it reduces the overall recovery time. Take RDP as an example. Replace recovery reduces the recovery time respectively by 20.9% ($p = 7$), 23.9% ($p = 11$), 22.5% ($p = 13$), and 22.0% ($p = 17$). These empirical results also conform to the previous theoretical analysis [26], which shows that the optimal recovery can reduce the number of reads by at about 25%.

Figure 9 shows the results for various coding schemes that tolerate any three or more disk failures. We observe that for STAR, replace recovery reduces the recovery time of conventional recovery by 25-29% (for $p = 11$, 13, and 17), which is consistent with our analysis in Section IV. For CRS($k = 10$, $m = 6$) and CRS($k = 12$, $m = 4$), our experimental results indicate 15.5% and 11.1% reductions of

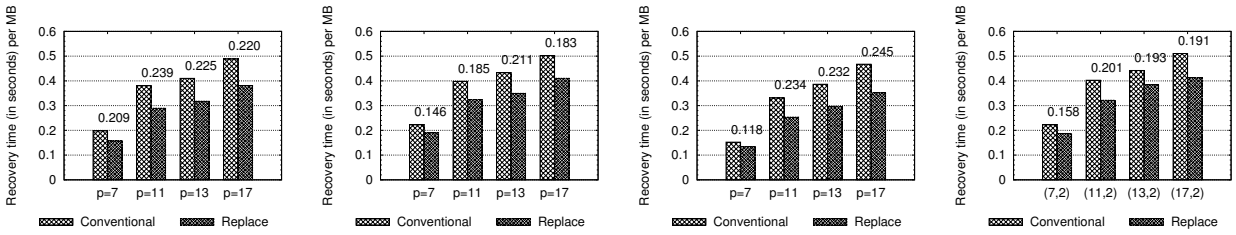


Fig. 8. Experiment 2 - Comparisons between conventional and replace recoveries for different double-fault tolerant codes. We also indicate the percentage decrease in recovery time of replace recovery over conventional recovery for each code.

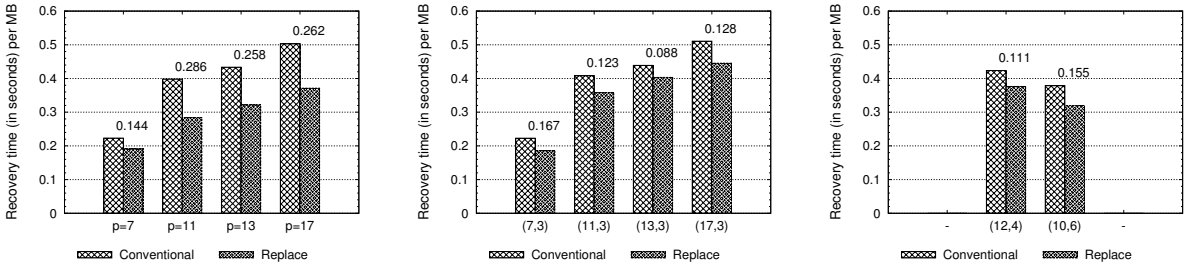


Fig. 9. Experiment 2 - Comparisons between conventional and replace recoveries for different codes that tolerate any three or more disk failures. We also indicate the percentage decrease in recovery time of replace recovery over conventional recovery for each code.

recovery time, respectively. It is important to note that the parameters ($k = 10$, $m = 6$) have been used in a commercial dispersed storage system [4].

Experiment 3: Parallel recovery. We now evaluate the recovery performance based on parallelization (see Section V-B). We aim to show that replace recovery still outperforms conventional recovery in parallelized implementation. Here, we use RDP($p = 13$), STAR($p = 13$), and CRS($k = 12$, $m = 4$) as representatives for different degrees of fault tolerance.

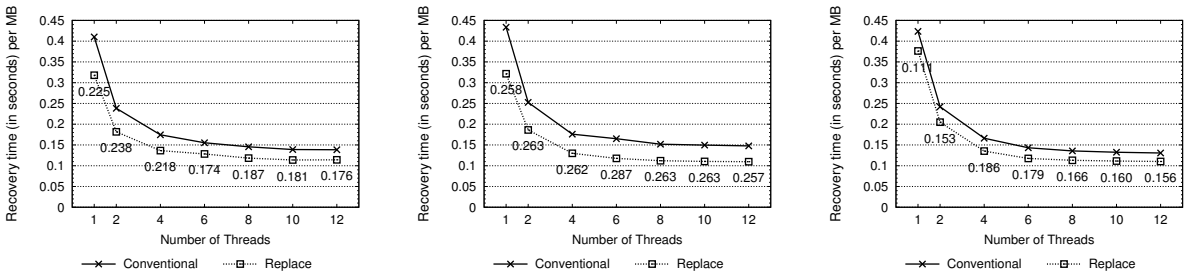
We first evaluate the recovery performance of SOR-based multi-threaded implementation, while we still use a single recovery server. Figure 10 shows the recovery time versus the number of recovery threads being deployed in a single server. As the number of threads increases (with no more than four threads), the recovery times for both conventional and replace recoveries significantly decrease. For example, let us we consider CRS($k = 12$, $m = 4$). When four recovery threads are used, the recovery times of conventional and replace recoveries are reduced by 61% and 64%, respectively when compared to the single-threaded case. However, when the number of threads goes beyond four, the improvement is marginal. The main reason is that the performance gain is bounded by the number of CPU cores (recall that our recovery server is equipped with a Quad-Core CPU).

More importantly, the results presented in Figure 10 show the applicability of replace recovery in parallelized implementation. We observe that replace recovery uses less recovery time than conventional recovery *regardless of the number of recovery threads being used*. For example, in STAR, replace recovery reduces the recovery time of conventional recovery by 25.7-28.7%; in CRS, the recovery time reduction is 15.3-18.6% when more than one recovery thread is used.

We now evaluate the recovery performance when we use multiple recovery servers. Here, we deploy two executors in two separate Quad-Core servers (as opposed to one Quad-Core server in our prior experiments) for parallel recovery. We configure each executor server to run four recovery threads (i.e., we have a total of eight recovery threads). Also, we deploy the dispatcher in one of the executor servers. In multi-server recovery mode, the dispatcher splits the whole recovery process into eight tasks (with a group of stripes). It first dispatches one task to each of the two executers. When one of the executors finishes its assigned task, the dispatcher assigns that executor another task. We verify that in all runs of our experiments, each of the two executers is always assigned four tasks. This is expected, as the executors have the same number of CPU cores.

We now measure the recovery time performance with this parallel setup. Figure 11 compares the recovery times of four recovery approaches: (i) single-server, multi-threaded (conventional), (ii) single-server, multi-threaded (replace), and (iii) multi-server (conventional), and (iv) multi-server (replace). We observe that multi-server implementation reduces the recovery time compared to the single-server implementation. For example, for replace recovery, the multi-server implementation reduces the recovery time by 24.4%, 22.0%, 19.82% for RDP, STAR, and CRS, respectively when compared to the single-server approach. In theory, we should expect 50% reduction, but the coordination overhead between the dispatcher and executors may degrade the actual performance. Nevertheless, the multi-server approach can provide additional recovery time improvements.

Note that even in multi-server implementation, we still observe the improvements of replace recovery over conventional recovery, as the recovery time is reduced by 25.3%,



(a) RDP($p = 13$) (b) STAR($p = 13$) (c) CRS($k = 12$, $m = 4$)

Fig. 10. Experiment 3 - Multi-threaded recovery based on SOR.

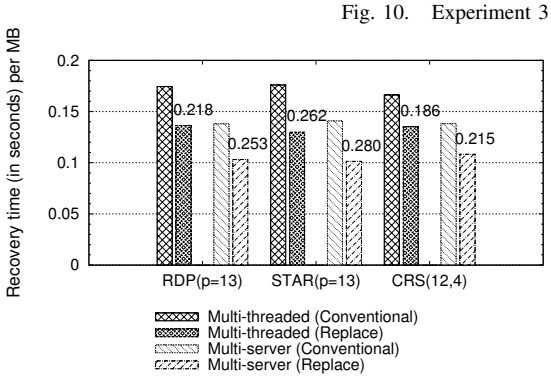


Fig. 11. Experiment 3 - Parallel recovery. Each number denotes the overall percentage decrease of recovery time compared to conventional recovery.

28.0%, 21.50% for RDP, STAR, and CRS, respectively. The results also validate the applicability of replace recovery in parallelized implementation.

VII. Conclusions

We address the problem of speeding up single-disk failure recovery for large-scale storage systems that use XOR-based erasure codes. Our objective is to minimize the amount of data, or the number of symbols, being read from surviving disks during the recovery operation. We address the problem from both theoretical and practical perspectives. From a theoretical perspective, we propose a replace recovery algorithm that provides near-optimal recovery performance for STAR and CRS codes, while the algorithm has a polynomial computational complexity. From a practical perspective, we design and implement our replace recovery algorithm on top of a parallel recovery architecture for scalable recovery performance. Experiments on a networked storage system testbed show that our replace recovery significantly reduces the recovery time over conventional recovery in both single-threaded and parallel recovery implementations.

Appendix

Proof of Theorem 4.1. We define a *parity set* as the set that contains a parity symbol and the data symbols encoded into that parity symbol. To recover a failed data disk in STAR, we can select x parity sets of slope 0, y parity sets of slope -1, and z parity sets of slope 1. We can select $p-1$ parity sets from any combinations of x, y, and z, such that $x+y+z = p-1$. Thus, the number of symbols in the selected parity sets is $xp + (y + z)(p - 1)$, in which the number of overlapping symbols is at most $x(y + z) + yz$. Thus, the number of read symbols for recovery per stripe (denoted by R) will be:

$$\begin{aligned}
R &\geq xp + (y+z)(p-1) - x(y+z) - yz \\
&= (p-1)p - (y+z) - x(y+z) - yz \\
&= (p-1)p - (y+z) - (p-1-y-z)(y+z) \\
&\quad - yz \\
&= (p-1)p + (y+z)^2 - p(y+z) - yz \\
&\geq (p-1)p + \frac{3}{4}(y+z)^2 - p(y+z) \\
&= \frac{3}{4}[(y+z)^2 - \frac{4}{3}p(y+z) + \frac{4}{9}p^2] + (p-1)p \\
&\quad - \frac{1}{3}p^2 \\
&\geq (p-1)p - \frac{1}{3}p^2 \\
&= \frac{2}{3}p^2 - p \text{ (symbols per stripe).}
\end{aligned}$$

We can see that R is lower bounded by $(\frac{2}{3}p^2 - p)$ symbols.

Acknowledgment

This work is supported by the National Nature Science Foundation of China under Grant No. 61073038 and the 111 Project under Grant No. B07033.

References

[1] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. Voelker. Total Recall: System Support for Automated Availability Management. In *Proc. of NSDI*, 2004.

[2] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures. *IEEE Trans. on Computers*, 44(2):192–202, 1995.

[3] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. An XOR-Based Erasure-Resilient Coding Scheme. *International Computer Sciences Institute Technical Report ICSI TR-95-048*, 1995.

[4] CLEVERSAFE. Cleversafe Dispersed Storage. http://www.cleversafe.org/downloads, 2008.

[5] 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 USENIX FAST*, 2004.

[6] DataRecovery. http://www.datarecovery.com/.

[7] DataTech Labs. http://www.datatechlab.com/.

[8] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's Highly Available Key-Value Store. In *Proc. of ACM SOSP*, 2007.

[9] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *IEEE Trans. on Information Theory*, 56(9):4539–4551, 2010.

[10] S. Ghemawat, H. Gobioff, and S. Leung. The Google File System. In *Proc. of ACM SOSP*, 2003.

[11] K. Greenan, X. Li, and J. Wylie. Flat XOR-Based Erasure Codes in Storage Systems: Constructions, Efficient Recovery, and Tradeoffs. In *Proc. of IEEE MSST*, 2010.

[12] M. Holland, G. Gibson, and D. Siewiorek. Architectures and Algorithms for On-line Failure Recovery in Redundant Disk Arrays. *Distributed and Parallel Databases*, 2(3):295–335, 1994.

[13] S. Hopkins and B. Coile. AoE (ATA over Ethernet). http://support.coraid.com/documents/AoEr11.txt, Feb 2009.

[14] Y. Hu, C.-M. Yu, Y.-K. Li, P. P. C. Lee, and J. C. S. Lui. NCFS: On the Practicality and Extensibility of a Network-Coding-Based Distributed File System. In *Proc. of NetCod*, July 2011.

[15] C. Huang and L. Xu. STAR: An Efficient Coding Scheme for Correcting Triple Storage Node Failures. *IEEE Trans. on Computers*, 57(7):889–901, 2008.

[16] O. Khan, R. Burns, J. Plank, and C. Huang. In Search of I/O-Optimal Recovery from Disk Failures. In *Proc. of USENIX HotStorage*, 2011.

[17] O. Khan, R. Burns, J. S. Plank, W. Pierce, and C. Huang. Rethinking Erasure Codes for Cloud File Systems: Minimizing I/O for Recovery and Degraded Reads. In *Proc. of USENIX FAST*, 2012.

[18] 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 ACM ASPLOS*, 2000.

[19] J. Li, S. Yang, and X. Wang. Building Parallel Regeneration Trees in Distributed Storage Systems with Asymmetric Links. In *Proc. of CollaborateCom*, pages 1–10. IEEE, 2010.

[20] J. Li, S. Yang, X. Wang, and B. Li. Tree-Structured Data Regeneration in Distributed Storage Systems with Regenerating Codes. In *Proc. of IEEE INFOCOM*, 2010.

[21] J. Plank, J. Luo, C. Schuman, L. Xu, and Z. Wilcox-O'Hearn. A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries for Storage. In *Proc. of USENIX FAST*, pages 253–265, 2009.

[22] I. Reed and G. Solomon. Polynomial Codes over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[23] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2009.

[24] F. Wang, S. Brandt, E. Miller, and D. Long. OBFS: A File System for Object-based Storage Devices. In *Proc. of IEEE MSST*, pages 283–300. Citeseer, 2004.

[25] Z. Wang, A. Dimakis, and J. Bruck. Rebuilding for Array Codes in Distributed Storage Systems. In *IEEE GLOBECOM Workshops*, 2010.

[26] 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 Trans. on Storage*, 7(3):11, 2011.

[27] L. Xu and J. Bruck. X-code: MDS Array Codes with Optimal Encoding. *IEEE Trans. on Information Theory*, 45(1):272–276, 1999.

[28] S. Xu, R. Li, P. Lee, Y. Zhu, L. Xiang, Y. Xu, and J. Lui. On the Recovery of Single-node Failure for X-code-based Distributed Storage Systems. Technical report, University of Science and Technology of China, 2011.

[29] Y. Zhu, P. P. C. Lee, L. Xiang, Y. Xu, and L. Gao. A Cost-based Heterogeneous Recovery Scheme for Distributed Storage Systems with RAID-6 Codes. In *Proc. of IEEE/IFIP DSN*, 2012.