On the Speedup of Recovery in Large-Scale Erasure-Coded Storage Systems

Yunfeng Zhu, Patrick P. C. Lee, Yinlong Xu, Yuchong Hu, and Liping Xiang

Abstract—Modern storage systems stripe redundant data across multiple nodes to provide availability guarantees against node 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 tolerating failures, a storage system must also provide fast failure recovery to reduce the window of vulnerability. This work addresses the problem of speeding up the recovery of a single-node failure for general XOR-based erasure codes. 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 extend the algorithm to adapt to the scenario where nodes have heterogeneous capabilities (e.g., processing power and transmission bandwidth). We implement our replace recovery algorithm atop a parallelized architecture to demonstrate its feasibility. We conduct experiments on a networked storage system testbed, and show that our replace recovery algorithm uses less recovery time than the conventional recovery approach.

Index Terms—XOR-coded storage system, single-node failure, recovery algorithm

1 Introduction

We have witnessed different implementations of large-scale storage systems, such as GFS [9], Dynamo [8], and Azure [3], in which data is distributed over a collection of *nodes* (or more generally, physical storage devices, which we collectively term as "nodes" in this paper). To ensure data availability, it is necessary to *tolerate* node failures, which are common in large-scale storage systems [9]. Data availability can be achieved by keeping redundant data in multiple nodes based on replication or erasure coding.

In addition to tolerating node failures, it is also necessary to *recover* node failures, so as to preserve the required redundancy level and avoid data unavailability. In this paper, we focus on the recovery of a single-node failure, which occurs more frequently than a concurrent multi-node failure in practice [12], [15]. Recovery of a single-node failure can be achieved by retrieving data from existing surviving nodes and reconstructing the lost data of the failed node in a new node.

To minimize the overall recovery time, one important objective is to minimize the amount of data being read from the surviving nodes. Here, we focus on the recovery problem for a family of special-purpose 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-node failures (e.g., RDP [7], EVENODD [1], X-code [26]), triple-node failures (e.g., STAR [13]), or a general number of failures (e.g., Cauchy Reed-Solomon (CRS) codes [2]). Recent studies have proposed recovery solutions to minimize the amount of data being read for different double-fault tolerant codes, including RDP [25], EVENODD [23], and X-code [27]. However, extending such results for the STAR and CRS codes, which can tolerate more than two node failures, is non-trivial due to the completely different data layouts. For general XOR-based erasure codes, the recovery problem of minimizing the amount of data read is NP-hard [15]. One approach of solving the optimization problem is to enumerate all recovery possibilities [15]. Such an approach, which we call *enumeration recovery*, applies to *any* XOR-based erasure code. However, while enumeration recovery can always find the optimal single-node failure recovery solution for any XOR-based erasure code, enumerating all recovery possibilities is very time consuming. Although the search space can be pruned [15], the search time remains expensive.

In this paper, we focus on speeding up the recovery of a single-node failure for XOR-based erasure-coded storage systems. Instead of constructing new erasure codes to improve recovery performance, we build a recovery mechanism that is applicable for *any* given XOR-based erasure code as in enumeration recovery, while reducing the search complexity. Specifically, our primary objective is to minimize the overall time of the recovery operation, while the recovery solution can be quickly determined. One fundamental requirement for this objective is to minimize the amount of data read from the surviving nodes for recovery. We thus propose a *replace recovery algorithm*, which uses a hill-climbing (greedy) approach [20] to find a near-optimal recovery solution for general cases. Intuitively, it starts with a feasible recovery solution, and incrementally *replaces*

- *Y. Zhu, Y. Xu, and L. Xiang are with University of Science & Technology of China (emails: {zyfl,xlping}mail.ustc.edu.cn, ylxu@ustc.edu.cn)*
- *P. P. C. Lee and Y. Hu are with the Chinese University of Hong Kong, Shatin, N.T., Hong Kong (email: pclee@cse.cuhk.edu.hk, yuchonghu@gmail.com).*

- *An earlier version of this paper appeared in IEEE Conference on Massive Storage Systems and Technologies (MSST) [28]. This journal version extends our prior work to support heterogeneous storage environments.*

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. We further extend our replace recovery algorithm to adapt to the heterogeneous scenario where nodes have different performance capabilities (e.g., processing power and transmission capabilities). This implies that our replace recovery algorithm can be applied in online mode based on the current performance costs of surviving nodes. Note that enumeration recovery is infeasible in doing so due to its exponential complexity.

We implement our replace recovery algorithm atop a parallelized, multi-core architecture, so as to validate its practicality in real deployment. We conduct experiments on a networked storage system testbed of different scales (with up to 21 storage nodes). We validate the recovery time improvement of our replace recovery over the conventional recovery approach (which we define in Section 2.1) for different XOR-based erasure codes.

We summarize our contributions as follows. We propose a replace recovery algorithm that speeds up single-node failure recovery for any XOR-based erasure code. The algorithm addresses the speedup issues in three aspects: (i) it reduces the search time of finding a recovery solution; (ii) it returns a recovery solution that reduces recovery I/Os (and hence recovery time); and (iii) it can be extended for parallelized recovery using multi-core technologies.

The rest of the paper proceeds as follows. Section 2 reviews related studies on single-node failure recovery. Section 3 motivates the need of speeding up single-node failure recovery. Section 4 proposes a simplified recovery model. Section 5 presents our replace recovery algorithm and shows how it can be extended for a heterogeneous environment. Section 6 uses simulations to quantitatively evaluate the efficiency of our replace recovery algorithm. Section 7 presents our experimental results. Section 8 concludes this paper. We also refer readers to our digital supplementary file for additional details of this work.

2 Background and Related Work

In this section, we review the closely related work on the failure recovery problem for XOR-based erasure codes. Further literature review can be found in Section 1 of the supplementary file.

2.1 Background: XOR-based Erasure Codes

We first define the vocabularies based on [18]. We consider a storage system employing an XOR-based erasure code. It contains an array of n nodes, in which k nodes hold *data*, and the remaining $m = n - k$ nodes hold coding information (which we call *parity*) encoded from the data. Each node 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 node is encoded from a strip in each data node. 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 nodes for load balancing [18]. Thus, the definitions of the data (parity) nodes may vary across stripes, depending on where the data (parity) strips are located.

We require the XOR-based erasure codes satisfy the *Maximum Distance Separable (MDS)* property, such that the original data can be reconstructed from any k out of n surviving nodes. In other words, the storage system can tolerate any $m = n - k$ concurrent node failures.

Our work is applicable for large-scale distributed storage systems, including clustered storage systems in local-area networks (e.g., GFS [9] and HDFS [22]), or dispersed storage systems in wide-area networks (e.g., Dynamo [8], Cleversafe [6], Azure [3]). Erasure coding has been deployed in enterprise storage systems [6], [12], [21], such that data is divided into stripes that are distributed across different storage nodes. Here, we use the term "node" in a broad sense to logically refer to a general physical storage device (e.g., a storage server or a network drive) deployed in a storage system. A node can fail and lose all stored data, and we aim to reconstruct the lost data in a new node. Our recovery design is based on the given layout of data/parity symbols across different nodes, and does not depend on the implementation of the underlying storage architecture.

We illustrate the above definitions via an example. We consider an XOR-based erasure code called RDP [7], 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 nodes (Nodes 0 to 3) are the data nodes, while the last two nodes (Nodes 4 and 5) are the parity nodes. Node 4 contains all the *row* parity symbols, e.g., $d_{0,4}$ is the XOR's of symbols $d_{0,0}$, $d_{0,1}$, $d_{0,2}$, $d_{0,3}$. Node 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 [1] and X-Code [26].

We note that every XOR-based erasure code can be represented by a *generator matrix* [18]. To illustrate, we consider the Cauchy Reed Solomon (CRS) codes [2], which can tolerate a general number of node failures. Figure 2 shows the encoding mechanism of a CRS code 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.

To recover node failures, the *conventional recovery* ap-

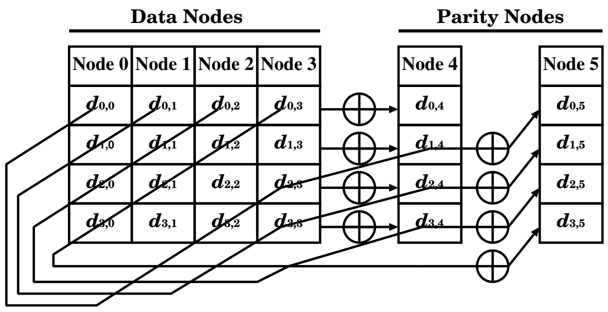


Fig. 1. RDP code with $p = 5$.

proach downloads the data and parity symbols from k nodes, 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 [19] and all XOR-based erasure codes) and any number of node failures no more than m. However, the frequency of a single-node failure is often higher than that of a concurrent multi-node failure. This is generally true when the aggregated node failure rate is lower than the node recovery rate [25]. Thus, using the conventional recovery approach as a baseline, many studies (e.g., [4], [15], [17], [23]–[25], [27], [29]) propose optimal or near-optimal recovery solutions specifically for recovering a single-node failure. In this paper, we seek to *minimize the number of symbols being read (or I/Os) from the surviving nodes for the single-node failure recovery.* In addition, we aim to maintain the recovery efficiency in a heterogeneous storage environment.

In this paper, our algorithm design focuses on recovering a failed data node, since in practice the number of data nodes is larger than that of parity nodes (i.e., $k > m$) in a storage system. If the failed node is a parity node, then we assume that we resort to conventional recovery as in [25], in which we read all original data symbols and encode into the parity symbols. We can actually minimize the number of read symbols for recovering a parity node failure if we use enumeration recovery [15] (see details in Section 2.3). On the other hand, it remains an open issue of how to speed up the search of efficient recovery solutions for parity node failures, and we pose it as future work. In our following discussion, by a single-node failure, we mean the failure of a single data node.

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

2.2 Hybrid Recovery

Some studies propose optimal recovery schemes for a single-node failure specifically for double-fault tolerant XOR-based erasure codes, with an objective of minimizing the number of read symbols. Xiang *et al.* [25] study the optimal recovery of a single-node failure for RDP, and the recovery solution reduces the number of read symbols by 25% compared to conventional recovery. Wang *et al.* [23] 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.* [27] also propose optimal single-node failure recovery for X-code. Wu *et al.* [24] propose efficient single-node failure recovery for a new horizontal-diagonal parity (HDP) code. These studies use the same core idea of *hybrid recovery* for RDP in [25]. Thus, in the following, we use RDP to explain the hybrid recovery idea.

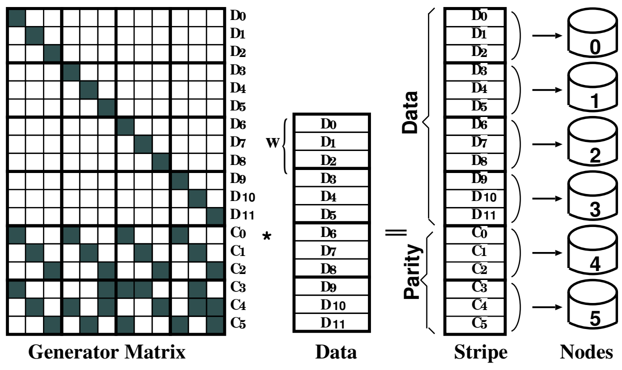


Fig. 2. CRS for $k = 4$, $m = 2$ and $\omega = 3$.

To recover a single-node failure, the conventional recovery approach in essence recovers each lost symbol in the failed node *independently*. Specifically, if a data node 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 node. We illustrate the idea of the conventional recovery approach using Figure 1. For example, if Node 1 fails, one can read $d_{0,0}, d_{0,2}, d_{0,3}, d_{0,4}$ to recover $d_{0,1}$ of Node 1. Thus, the total number of read symbols for repairing Node 1 is 16. In contrast, hybrid recovery [25] uses a *combination of row and diagonal parity sets* of data and parity symbols to repair Node 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 Node 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 [23] and X-code [27].

2.3 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 [15], 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 [10]. Here, we use the example of CRS in Figure 2 to explain how enumeration recovery works. To recover a data node failure in CRS, the conventional recovery approach may select the parity symbols in the first parity node together with all surviving data symbols. For example, if Node 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* [10], [15]. 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 Node 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 nodes. In general, the problem of finding the optimal recovery solution that minimizes the number of read symbols is NP-hard [15].

3 Motivation

While enumeration recovery can find the optimal recovery solution for a single-node failure in any XOR-based erasure codes, it has a very high computational overhead. In an erasure code with parameters (n, k, m, ω), enumeration recovery aims to find ω out of $2^{m\omega} - 1$ recovery equations that minimize the amount of data being read during recovery. Therefore, enumeration recovery has a search space of up to $\binom{2^{m\omega}-1}{\omega}$ combinations of recovery equations. 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 [15]. 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).

We by no means that claim that enumeration recovery should be entirely substituted in real deployment. In practice, we believe that a combination of recovery approaches should be used. When the parameters are deterministic (e.g., when nodes are homogeneous) and the parameter values m and w are small, enumeration recovery should be used since it provides an optimal recovery solution; otherwise, we may require a recovery solution that has efficient search performance while providing near-optimal recovery performance. In the following discussion, we address the latter issue to complement enumeration recovery.

In Section 2 of the supplementary file, we describe several scenarios where enumeration recovery becomes *infeasible* to deploy.

4 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 node and the data symbols in the $k - 1$ data nodes based on the MDS property (see Section 2.1). Our observation is that the failure of a single data node 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 2.2, 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 2.3, 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 [25], 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 6) and experiments (see Section 7). We now formulate a simplified recovery model that solves the single-node failure recovery problem for any XOR-based erasure code.

Simplified recovery model. To recover a failed node, 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 when 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 node capabilities.* The recovery algorithm can be easily converted to handle heterogeneous node capabilities, and hence can promptly return an effective recovery solution in the online recovery scenario.

5 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-node failure. However, it remains computationally expensive to search for the "optimal" 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-node failure recovery, or more generally, to minimize the total recovery cost. Note that the algorithm is applicable for *any* XOR-based erasure code.

The idea of our replace recovery algorithm is as follows. Let $\mathcal{P}_i$ be the set of parity symbols in the ith parity node, 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 nodes, due to the MDS property (see Section 2.1). 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 [20] 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).

5.1 Algorithm Design

Algorithmic details. Algorithm 1 shows the replace recovery algorithm whose objective is to minimize the number of read symbols during recovery. We call it *homogeneous replace recovery (HoRR)*. In the algorithm, we first 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 node (Node 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

Algorithm 1 Replace Recovery Algorithm

```
1: Initialize $\mathcal{X}$ = $\mathcal{P}_1$
2: for $i$ = 2 to $m$ do
3:    Set $\mathcal{Y}$ = $\mathcal{P}_i$
4:    for each parity symbol $Y$ in $\mathcal{Y}$ do
5:       Set $f$ = false;
6:       for each parity symbol $X$ in $\mathcal{X}$ do
7:          Set $\mathcal{X}' = \mathcal{X} - \{X\} + \{Y\}$
8:          if $\mathcal{X}'$ is valid and reduces the number of read
            symbols of $\mathcal{X}$ then
9:             Compute $R_{X,Y}$ = number of read symbols of $\mathcal{X}'$
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 $\mathcal{X}$
17:      Remove $Y'$ from $\mathcal{Y}$
18:   end if
19:   Repeat Steps 4-18 until $\mathcal{Y}$ is empty or $f$ == false
20: end for
21: Return $\mathcal{X}$
```

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 nodes, 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.

5.2 Extension to a Heterogeneous Environment

We now show how Algorithm 1 can be modified slightly to adapt to the scenario where nodes have heterogeneous capabilities. Here, we consider a metric called the *recovery cost* [29] for the recovery solution $\mathcal{X}$ of the failed node k. Suppose that the recovery operation of $\mathcal{X}$ reads y_i symbols from Node i ($i \neq k$). Let c_i be the unit recovery cost of fetching a single symbol from Node i. The total recovery cost is then defined as:

$$\sum_{i=0, i\neq k}^{n} c_i y_i.$$

In this paper, we define c_i as the inverse of the transmission bandwidth of Node i. Thus, the total recovery cost can be viewed as the total amount of transmission time to download the symbols from all surviving nodes *per stripe*. Recall from Section 2.1 that a storage system divides data into stripes, each of which is encoded independently. Also, the data and parity strips are rotated across different stripes for load balancing. Since the cost model is defined on a per-stripe basis, we can use multi-threading to parallelize the cost computations across different stripes. Each thread determines, based on the above equation, the symbols to be read from a stripe for recovery, and then reads the symbols accordingly.

To get the heterogeneous version of our replace recovery algorithm (i.e., HeRR), we merely substitute the computation of the number of read symbols for $\mathcal{X}$ in Algorithm 1 (in Steps 8-9) with the computation of the recovery cost for $\mathcal{X}$.

In this work, we focus on finding a cost-effective recovery solution with the values of c_i's as inputs. We assume that c_i can be determined in advance before running our recovery algorithm, say by taking the moving average of periodic measurements [5]. An important future work is to construct an automatic recovery mechanism that integrates real-time measurements of c_i's, node failure detection, and an efficient recovery algorithm.

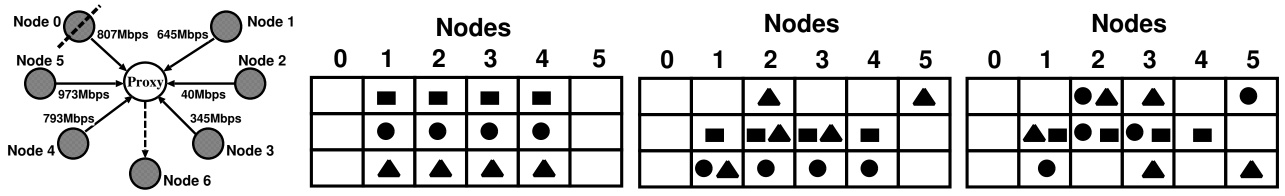
We now illustrate via a contrived example how HeRR significantly improves the recovery performance over the conventional recovery and HoRR in some specific scenarios. Figure 3(a) shows a heterogeneous distributed storage system where a centralized proxy controller connects to 6 nodes. Suppose that the storage system adopts CRS in Figure 2 as its coding scheme. Each node stores 3 symbols of size α each (in unit of Mb). Suppose that Node 0 fails. The proxy downloads data from other surviving nodes and regenerates the lost data in the new node (i.e., Node 6). The unit cost c_i of each node is set to the inverse of the link transmission bandwidth of Node i. We now compute the total recovery costs using the conventional recovery, HoRR, and HeRR, respectively.

Conventional. The proxy reads 3 symbols from each of Nodes 1 to 4 (see Figure 3(b)). Thus, the total recovery cost is:

$$\frac{3\alpha}{645} + \frac{3\alpha}{40} + \frac{3\alpha}{345} + \frac{3\alpha}{793} = 0.0921\alpha \text{ (in sec)}.$$

HoRR. HoRR returns the recovery solution $\mathcal{X} = \{C_3, C_1, C_2\}$ (see Section 5.1). Thus, the proxy downloads the data and parity symbols from 5 nodes (see Figure 3(c)). The total recovery cost is:

$$\frac{2\alpha}{645} + \frac{3\alpha}{40} + \frac{2\alpha}{345} + \frac{2\alpha}{793} + \frac{1\alpha}{973} = 0.0875\alpha \text{ (in sec)}.$$



(a) Heterogeneous setting (b) Conventional approach (c) HoRR approach (d) HeRR approach

Fig. 3. An example for three recovery approaches: conventional, HoRR and HeRR.

HeRR. The HeRR algorithm works as follows. We initialize $\mathcal{X}$ with $\{C_0, C_1, C_2\}$. Note that the recovery cost of $\mathcal{X}$ is 0.0623α. We now 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. Among these three, the latter two replacements both reduce the recovery cost to 0.0653α (the maximum reduction achievable). We first try to replace C_1 by C_4 (i.e., $\mathcal{X}$ = $\{C_0, C_4, C_2\}$) and consider $\mathcal{Y} = \{C_3, C_5\}$. We find no replacement gives any smaller recovery cost.

We then rollback to replace C_2 by C_5 (i.e., $\mathcal{X}$ = $\{C_0, C_1, C_5\}$) and consider $\mathcal{Y} = \{C_3, C_4\}$. We now find the replacement of C_0 by C_3 further reduces the recovery cost to 0.0651α, which is also the optimal heterogeneous recovery solution. Thus, HeRR finally returns $\mathcal{X}$ = $\{C_3, C_1, C_5\}$. The proxy then reads symbols (see Figure 3(d)). The total recovery cost is:

$$\frac{2\alpha}{645} + \frac{2\alpha}{40} + \frac{3\alpha}{345} + \frac{1\alpha}{793} + \frac{2\alpha}{973} = \ 0.0651\alpha \text{ (in sec).}$$

To summarize, compared to the conventional recovery, the total recovery cost reduction of HoRR is only 4.99%, while that of HeRR is 29.32%. This example justifies that HeRR works better in a heterogeneous environment.

6 Simulations

We present simulation results for the performance of our proposed replace recovery algorithm. In Section 3 of the supplementary file, we evaluate the recovery performance of HoRR for different coding schemes (e.g., RAID-6, STAR, and CRS) in a homogeneous setting. Here, we focus on the recovery performance of HeRR and the search performance.

6.1 Recovery Performance for HeRR

We evaluate the performance of HeRR in a heterogeneous storage environment. Note that our simulations consider a star-like network topology as in Figure 3(a), where a centralized proxy controller connects to a number of nodes, each having link transmission bandwidth following a uniform distribution U(0.3Mbps, 120Mbps), which is the link bandwidth distribution in PlanetLab [16]. As described in Section 5.2, we use the recovery cost as the metric to evaluate the recovery efficiency in heterogeneous environments, and let c_i be the inverse of the link transmission bandwidth of each Node i.

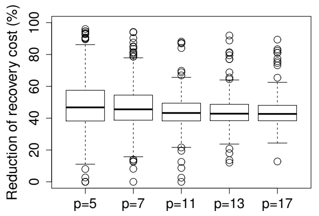
Comparisons with conventional recovery and HoRR. Here, we use STAR and CRS as representative coding schemes, and consider different parameter settings. We conduct 500 simulation runs for each of the parameter settings. We then plot the results via *box plots*. A box plot shows the minimum, lower quartile, median, upper quartile, maximum of all sampled results, and provides the outliers as well.

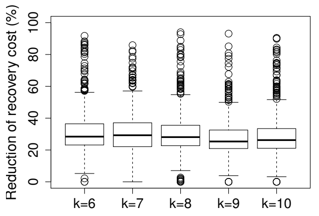
Figure 4 shows the percentage reduction of the recovery cost of HeRR over conventional recovery and HoRR. From Figures 4(a) and 4(b), HeRR can reduce the recovery cost of the conventional approach in both STAR and CRS codes by around 40% and 30%, respectively. Also, from Figures 4(c) and 4(d), HeRR can reduce the recovery cost of HoRR, by around 20% and 10% in STAR and CRS codes, respectively. There are some outlier simulation runs in which HeRR can reduce the recovery cost of HoRR by over 90% (e.g., $p = 5, 7$ for STAR and $k = 6$ for CRS). The main reason is that in those runs, some parity nodes happen to have very small transmission bandwidths. Since HoRR only seeks to minimize the number of symbols read, its recovery performance may suffer from the bottlenecked parity node. On the other hand, in those outlier runs, HeRR will bypass the slow parity node and switch to other parity nodes for recovery.

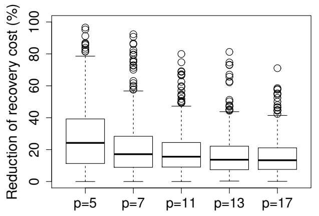
Comparison with enumeration recovery. We further compare HeRR with enumeration recovery [15], which enumerates all possible recovery solutions and returns the one with the minimum recovery cost. Similar to above, we conduct 500 simulation runs and obtain the average results. Here, we briefly summarize our findings for some parameters. For STAR with $p = 5$, enumeration recovery has 6.75% less recovery cost than HeRR; for CRS with $m = 3$ and $\omega = 4$, it has around 14% less recovery cost for $k = 6$ to $k = 10$. On the other hand, we cannot include all optimal results here due to the high computational overhead. We discuss this in the next subsection.

6.2 Search Performance

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


(a) STAR, over conventional


(b) CRS($m = 3$, $\omega = 4$), over conventional


(c) STAR, over HoRR

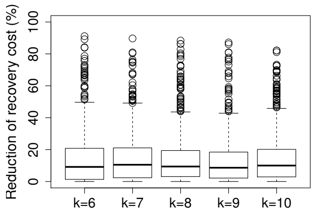

(d) CRS($m = 3$, $\omega = 4$), over HoRR

Fig. 4. Percentage reduction of recovery cost of HeRR over conventional ((a) and (b)) and HoRR ((c) and (d)).

Our evaluation is conducted on a Linux desktop computer running with 3.2GHz CPU and 2GB RAM. As discussed in Section 3, in enumeration recovery, we need to search for a maximum of $\binom{2^{m\omega}-1}{\omega}$ combinations of recovery equations. Here, we consider CRS, which allows us to configure different values of m and ω.

Table 1 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 node; 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 1
Search times 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

7 Experiments

In Section 4 of the supplementary file, we describe how we implement our replace recovery algorithm and how we extend the algorithm to support parallelization.

Based on our implementation, we conduct testbed experiments on the single-node recovery approaches for different XOR-based erasure codes. 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 [25], our experimental results capture the actual I/O performance with real storage nodes.

7.1 Methodology

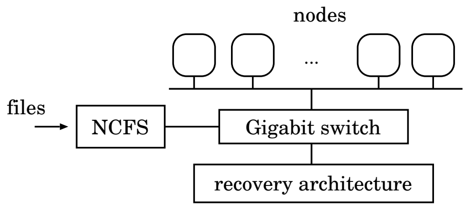


Fig. 5. Our testbed topology.

Our experimental testbed is built on an open-source networked storage system called NCFS [11]. Figure 5 shows our testbed. NCFS interconnects, over a network, different physical storage devices, each of which corresponds to a node as being considered in our experiments. It transparently stripes data across all nodes according to the respective coding scheme. We integrate different coding schemes into NCFS. We deploy our recovery architecture alongside NCFS and the nodes, while our architecture implements both conventional and replace recoveries. Our architecture is deployed on a Linux-based server equipped with an Intel Quad-Core 2.66GHz CPU and 4GB RAM. 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.

Our storage system consists of a cluster of (logical) nodes, each represented by a physical storage device. We experiment different numbers of nodes in the cluster, with at most 21 nodes depending on the parameters chosen for the coding schemes. Each logical node corresponds to a physical PC which has a SATA disk installed. For simplicity, we assume that the logical nodes have identical hardware configurations, although their actual hardware configurations (e.g., CPU speed, RAM capacity) may vary. We do not specifically optimize our implementation to demonstrate the absolute recovery performance. Instead, our goal is only to evaluate the *relative* performance of our replace recovery compared to the conventional approach under fair conditions.

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. We write 1GB of data into each node via NCFS using the specified coding scheme (i.e., if there are n nodes, then we write a total of nGB). Then NCFS will stripe the data across the nodes. We disable one of the data nodes in the storage system to make it resemble a failed node. We then perform the recovery

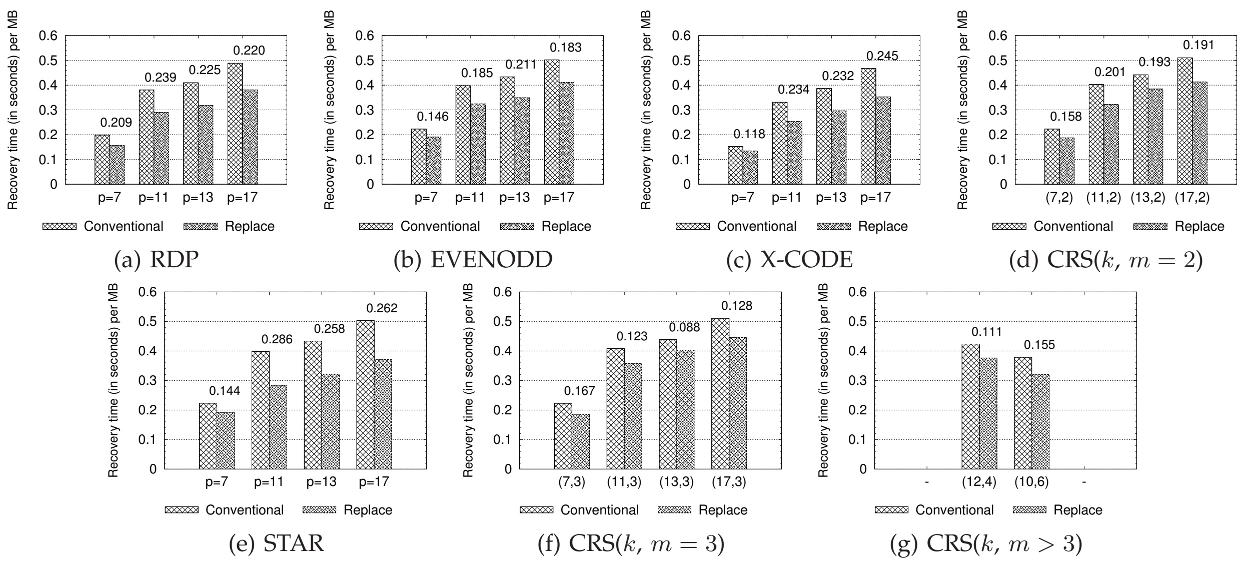


Fig. 6. Experiment 1 - Comparisons between conventional and replace recoveries for different codes that have different fault tolerance levels. We also indicate the percentage decrease in recovery time of replace recovery over conventional recovery for each code.

operation, that is, reading data from surviving nodes, reconstructing data, and writing data to a new node that we prepare. The recovery operation is done three times. We repeat this for all data nodes, and obtain the overall average. For instance, referring to RDP ($p = 5$) in Figure 1 (see Section 2.1), there are four data nodes. Thus, we run a total of 4×3 recovery operations, and average the recovery times over the 12 runs.

7.2 Results

We present results of our testbed experiments. Here, we fix the chunk size to be 512KB and use only a single recovery thread in our recovery architecture (i.e., without using parallelization). In Section 5 of the supplementary file, we also evaluate the impact of chunk size on the recovery performance, and the recovery performance using parallelization.

Experiment 1: 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.

Figures 6(a)-(d) 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 our experimental results (see the supplementary file), 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 nodes, 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 [25], which shows that the optimal recovery can reduce the number of reads by at about 25%.

Figures 6(e)-(g) show the results for various coding schemes that tolerate any three or more node 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 6. For CRS($k = 10$, $m = 6$) and CRS($k = 12$, $m = 4$), our experimental results indicate 15.5% and 11.1% reductions of recovery time, respectively. It is important to note that the parameters ($k = 10$, $m = 6$) have been used in a commercial dispersed storage system [6].

Experiment 2: Heterogeneous environment. We now evaluate the recovery performance of our heterogeneous version of replace recovery (i.e., HeRR) in a heterogeneous setting. Here, we consider STAR($p = 7$) and CRS($k = 7$, $m = 3$), assuming only single-threaded recovery is used.

In order to mimic a heterogeneous environment, we run intensive I/O applications in 3 nodes (namely, Node 1, 3, 8). We further use a standard storage benchmark IOzone [14] to measure the read bandwidth for each node. The results show that the actual read bandwidth of the above 3 nodes is roughly 23Mbps, while other nodes have 65Mbps of bandwidth. We then use HeRR to find out the efficient heterogeneous recovery solutions for STAR and CRS, respectively.

Figure 7 shows the total recovery time performance of different recovery approaches for STAR and CRS in our heterogeneous storage environment. We observe that HoRR reduces the recovery time by 7.2% and 4.4%, respectively. On the other hand, HeRR achieves 14.3% and 13.4% reductions of recovery time of conventional recovery. This validates that HeRR outperforms HoRR in a real heterogeneous environment. We also validate that our experimental results conform to our theoretical findings based on the analysis in Section 5.2. Note that in some extreme scenarios, the improvement of HeRR

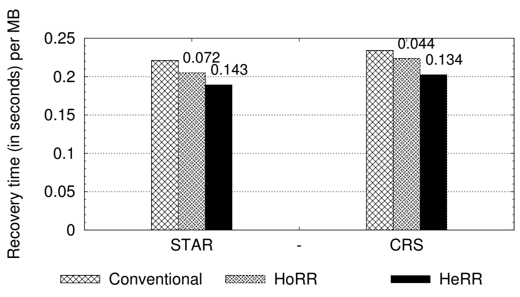


Fig. 7. Experiment 2 - Comparisons between three recovery approaches for STAR ($p = 7$) and CRS($k = 7$, $m = 3$), respectively.

over HoRR can be more significant (see Section 6.1).

8 Conclusions

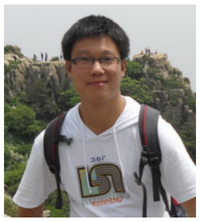
We propose mechanisms for speeding up single-node failure recovery for large-scale storage systems that use XOR-based erasure codes. Our objective is to minimize the amount of data, or the number of symbols, being read from surviving nodes during the recovery operation. We propose a replace recovery algorithm that provides near-optimal recovery performance for different coding schemes, while the algorithm has a polynomial computational complexity. We also extend replace recovery to adapt to heterogeneous storage environments. We 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. The source code of our implementation is available at **http://ansrlab.cse.cuhk.edu.hk/software/zpacr**.

Acknowledgment

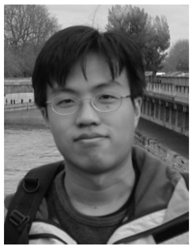
This work is supported by National Nature Science Foundation of China under Grant No. 61073038, Key Science and Technology Program of Anhui Province, China under Grant No. 1206c0805003, and grants from the University Grants Committee of Hong Kong (AoE/E-02/08 and ECS CUHK419212).

References

[1] 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.
[2] 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.
[3] B. Calder et al. Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In *Proc. of ACM SOSP*, 2011.
[4] Q. Cao, S. Wan, C. Wu, and S. Zhan. An Evaluation of Two Typical RAID-6 Codes on Online Single Disk Failure Recovery. In *Proc. of IEEE NAS*, pages 135–142. IEEE, 2010.
[5] M. Chowdhury, S. Kandula, and I. Stoica. Leveraging Endpoint Flexibility in Data-Intensive Clusters. In *Proc. of ACM SIGCOMM*, 2013.
[6] CLEVERSAFE. Cleversafe Dispersed Storage. http://www.cleversafe.org/downloads, 2008.
[7] 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.
[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] S. Ghemawat, H. Gobioff, and S. Leung. The Google File System. In *Proc. of ACM SOSP*, 2003.
[10] 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.
[11] 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.
[12] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure Coding in Windows Azure Storage. In *Proc. of USENIX ATC*, Jun 2012.
[13] 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.
[14] IOzone Filesystem Benchmark. http://www.iozone.org.
[15] 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.
[16] S. Lee, P. Sharma, S. Banerjee, S. Basu, and R. Fonseca. Measuring Bandwidth Between PlanetLab Nodes. In *Proc. of PAM*, 2005.
[17] S. Li, Q. Cao, J. Huang, S. Wan, and C. Xie. PDRS: A New Recovery Scheme Application for Vertical RAID-6 Code. In *Proc. of IEEE NAS*, pages 112–121. IEEE, 2011.
[18] 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.
[19] 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.
[20] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2009.
[21] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. XORing Elephants: Novel Erasure Codes for Big Data. *Proceedings of the VLDB Endowment*, 2013.
[22] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *Proc. of IEEE MSST*, May 2010.
[23] Z. Wang, A. Dimakis, and J. Bruck. Rebuilding for Array Codes in Distributed Storage Systems. In *IEEE GLOBECOM Workshops*, 2010.
[24] C. Wu, X. He, G. Wu, S. Wan, X. Liu, Q. Cao, and C. Xie. HDP Code: A Horizontal-Diagonal Parity Code to Optimize I/O Load Balancing in RAID-6. In *Proc. of IEEE/IFIP DSN*, 2011.
[25] 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.
[26] L. Xu and J. Bruck. X-code: MDS Array Codes with Optimal Encoding. *IEEE Trans. on Information Theory*, 45(1):272–276, 1999.
[27] S. Xu, R. Li, P. Lee, Y. Zhu, L. Xiang, Y. Xu, and J. Lui. Single Disk Failure Recovery for X-Code-Based Parallel Storage Systems. *IEEE Trans on Computers*, 2013. (To appear).
[28] Y. Zhu, P. Lee, Y. Hu, L. Xiang, and Y. Xu. On the Speedup of Single-Disk Failure Recovery in XOR-Coded Storage Systems: Theory and Practice. In *Proc. of IEEE MSST*, 2012.
[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.

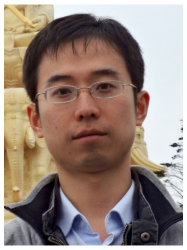


Yunfeng Zhu received his B.S. from the School of Computer Science, University of Science and Technology of China, Anhui, China, in 2008. He is currently working toward the Ph.D. degree at the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China. His research interests include distributed storage system, cloud storage and data deduplication.



Patrick P. C. Lee received the B.Eng. degree (first-class honors) in Information Engineering from the Chinese University of Hong Kong in 2001, the M.Phil. degree in Computer Science and Engineering from the Chinese University of Hong Kong in 2003, and the Ph.D. degree in Computer Science from Columbia University in 2008. He is now an assistant professor of the Department of Computer Science and Engineering at the Chinese University of Hong Kong. His research interests are in cloud storage, distributed systems and networks, and security/resilience.

Yinlong Xu received his B.S. in Mathematics from Peking University in 1983, and MS and Ph.D in Computer Science from University of Science and Technology of China(USTC) in 1989 and 2004 respectively. He is currently a professor with the School of Computer Science and Technology at USTC. Prior to that, he served the Department of Computer Science and Technology at USTC as an assistant professor, a lecturer, and an associate professor. Currently, he is leading a group of research students in doing some networking and high performance computing research. His research interests include network coding, wireless network, combinatorial optimization, design and analysis of parallel algorithm, parallel programming tools, etc. He received the Excellent Ph.D Advisor Award of Chinese Academy of Sciences in 2006.



Yuchong Hu received the B.S. degree in Computer Science and Technology from the School for the Gifted Young, University of Science & Technology of China, Anhui, China, in 2005. He received the Ph.D. degree in Computer Science and Technology from the School of Computer Science, University of Science & Technology of China, in 2010. He was a postdoctoral fellow at the Institute of Network Coding, the Chinese University of Hong Kong. His research interests include network coding and distributed storage.

Liping Xiang received her B.S. from the Department of Information and Computational Science, Anhui University, China, in 2007. She is currently working toward the Ph.D. degree at the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China. Her research interests include distributed storage system, data recovery, and network coding.