A Cost-based Heterogeneous Recovery Scheme for Distributed Storage Systems with RAID-6 Codes

Yunfeng Zhu†, Patrick P. C. Lee*, Liping Xiang†, Yinlong Xu†, Lingling Gao†
† School of Computer Science and Technology, University of Science & Technology of China
* Department of Computer Science and Engineering, The Chinese University of Hong Kong
{*zyfl, xlping, llinggao*}*@mail.ustc.edu.cn, pclee@cse.cuhk.edu.hk, ylxu@ustc.edu.cn*

***Abstract*—Modern distributed storage systems provide large-scale, fault-tolerant data storage. To reduce the probability of data unavailability, it is important to recover the lost data of any failed storage node efficiently. In practice, storage nodes are of heterogeneous types and have different transmission bandwidths. Thus, traditional recovery solutions that simply minimize the number of data blocks being read may no longer be optimal in a heterogeneous environment. We propose a *cost-based heterogeneous recovery (CHR)* algorithm for RAID-6-coded storage systems. We formulate the recovery problem as an optimization model in which storage nodes are associated with generic costs. We narrow down the solution space of the model to make it practically tractable, while still achieving the global optimal solution in most cases. We implement different recovery algorithms and conduct testbed experiments on a real networked storage system with heterogeneous storage devices. We show that our CHR algorithm reduces the total recovery time of existing recovery solutions in various scenarios.**

***Keywords*-distributed storage system, RAID-6 codes, node heterogeneity, failure recovery, experimentation**

I. Introduction

Distributed storage systems provide a scalable platform for storing massive data over multiple storage *nodes*, each of which could be a physical storage server or network drive. They have seen deployment in cloud storage (e.g., GFS [8] and Dynamo [6]) and peer-to-peer backup storage (e.g., Wuala [26]). Since node failures are common [8], a storage system ensures data availability typically by introducing *redundancy* to data storage, for example, via replication (e.g., in [6, 8]) or erasure coding (e.g., in [3, 15]). In addition, to maintain the overall system reliability, it is critical to *recover* node failures, i.e., to retrieve data from existing surviving nodes and reconstruct the lost data in new storage nodes. A key question is how to achieve efficient recovery from node failures in distributed storage systems.

Different metrics can be used to measure the efficiency of a recovery scheme. Since single-node failures are more often than concurrent multi-node failures, prior studies mainly focus on the recovery of a single-node failure. One metric of efficient single-node failure recovery is to *minimize the number of data blocks being read* specifically for XOR-based erasure codes (e.g., RDP [5], EVENODD [4], etc.), in which encoding is based on XOR operations only. Prior studies propose optimal recovery solutions for existing XOR-based erasure codes RDP [27] and EVENODD [24], as well as arbitrary XOR-based erasure codes [13].

On the other hand, extending the recovery solutions for XOR-based erasure-coded storage systems to a *heterogeneous* environment remains an open problem. It is natural for a storage system to be composed of heterogeneous types of storage nodes, for example, due to system upgrades or new node additions. In this case, the storage nodes within the same system may have different capabilities. In a networked setting, it is common that the transmission links that connect different storage nodes may not receive the same bandwidth and latency [18]. It is intuitive to retrieve fewer (or more) data blocks from the surviving storage nodes with lower (or higher) transmission bandwidths. Thus, existing recovery solutions [13, 24, 27] that are based on minimizing the number of data blocks being read may no longer provide an efficient recovery solution for a heterogeneous environment. The key motivation of this work is to *define a new metric of efficient recovery and develop the corresponding recovery solution for XOR-based erasure coded storage systems, such that the metric accounts for general distributed settings.* It is crucial that the number of blocks and which blocks to be fetched from the surviving storage nodes must follow the inherent rules of the specific coding scheme so as to recover the failed node successfully.

The contributions of this paper are three-fold. First, we formulate an optimization model that minimizes the total recovery cost of a single-node failure in a distributed storage system with heterogeneous types of storage nodes. Specifically, we focus on two RAID-6 codes: RDP [5] and EVENODD [4], both of which tolerate two node failures and have seen efficient single-node failure recovery solutions (see [27, 24], respectively). Our main idea is to associate each storage node with a cost component that quantifies the overhead of downloading per unit of data from the storage node, such that the cost component is generic and can be defined differently according to the deployment scenario.

Second, we propose a simplified model that significantly reduces the number of feasible solutions being traversed. This enables us to solve the simplified model in a computationally efficient manner, which is important for the

online recovery scenarios where the costs (e.g., transmission latencies) of storage nodes need to be determined in an on-demand fashion so as to accurately reflect the current network conditions. To this end, we propose a *cost-based heterogeneous recovery (CHR)* algorithm to solve for the optimal solution under the simplified model. We emphasize that the CHR algorithm addresses node heterogeneity, while maintaining the correctness of the failed node recovery. We show via simulations that our CHR algorithm achieves the global optimal solution in most cases. Also, it reduces the recovery costs of the conventional recovery approach (which recovers each lost data block of the failed node independently) and the hybrid recovery approach [27] by up to 50% and 30%, respectively, in a heterogeneous setting.

Third, to show that our CHR algorithm can be feasibly deployed in practice, we conduct testbed experiments on a networked storage system [12] with heterogeneous storage devices. As opposed to disk-based simulations [27], we evaluate the performance of actual read/write operations via real storage nodes during recovery. Our experimental results validate the improvement of our CHR algorithm. For example, compared to the conventional recovery approach for RDP, the existing hybrid approach [27] reduces the total recovery time by up to 20.78%, while our CHR algorithm further reduces the total recovery time by up to 31.19%.

The remainder of the paper proceeds as follows. Section II overviews existing recovery solutions for coded storage systems. Section III formulates our cost-based heterogeneous recovery model and presents the algorithm and its simulation analysis. Section IV presents the testbed experimental results. Section V concludes the paper.

II. Background and Related Work

A. Definitions

We consider a storage system with n physical storage nodes (or disks). We partition the storage system into fixed-size *stripes*, each of which is a two-dimensional array with a fixed number of rows and n columns. Each column corresponds to a unique storage node. Each stripe stores a fixed number of *symbols*. A symbol refers to a fixed-size block (or chunk) in a practical storage system. There are two types of symbols: (i) *data symbols*, which hold the original data, and (ii) *parity symbols* (or simply *parities*), which encode the data symbols with the same stripe. A *parity set* is a set of symbols containing a parity and the corresponding data symbols encoded to form the parity.

B. RAID-6 Codes: RDP and EVENODD

Our work focuses on RAID-6 codes, which are erasure codes that can tolerate any double-node failures. That is, any $n-2$ surviving nodes can be used to recover the original data. There are many implementations of RAID-6 codes. Here, we consider two of them: RDP [5] and EVENODD [4]. They are all XOR-based erasure codes, such

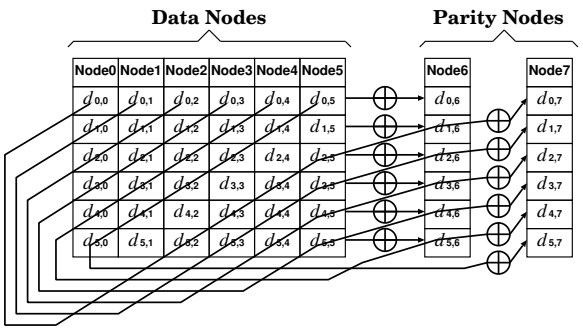


Figure 1. A stripe of RDP with p = 7.

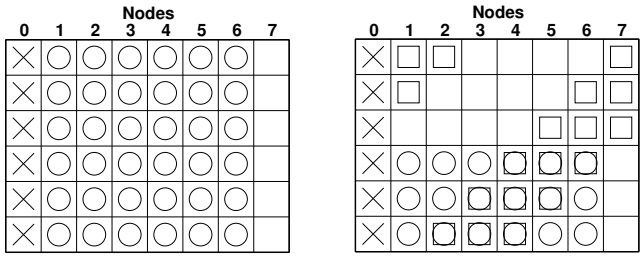
that all parities are encoded from data symbols based on XOR operations only. We specifically consider RDP and EVENODD because they have the corresponding single-node failure recovery solutions that minimize the number of symbols being read per stripe (see [27, 24], respectively). We will elaborate the single-node failure recovery problem in Section II-C.

RDP [5] is one of the most important double-fault tolerant RAID-6 codes. Its variant has seen deployment in commercial applications [19]. Its encoding takes a $(p-1)\times(p+1)$ stripe (i.e., two-dimensional array), where p is a prime number greater than 2. The first $p-1$ columns in the array store data symbols, while the last two store parities. It is proven that RDP achieves optimality both in computations and I/Os [5]. Figure 1 shows how RDP encoding works for $p = 7$, where $d_{i,j}$ is the i-th symbol in column j. Node 6 contains all *row parities*, while Node 7 contains all the *diagonal parities*. For example, let $\oplus$ denote the bitwise-XOR operator. Then the row parity $d_{0,6} = d_{0,0} \oplus d_{0,1} \oplus d_{0,2} \oplus d_{0,3} \oplus d_{0,4} \oplus d_{0,5}$, and the diagonal parity $d_{0,7} = d_{0,0} \oplus d_{5,2} \oplus d_{4,3} \oplus d_{3,4} \oplus d_{2,5} \oplus d_{1,6}$. It is easy to verify that each data symbol belongs to exactly one row parity set and one diagonal parity set [5].

EVENODD [4] is a predecessor of RDP. Its encoding takes a $(p-1)\times(p+2)$ stripe, such that the first p columns store the data symbols and the last two store parities. In EVENODD, one parity node stores row parities and another parity node stores diagonal parities, as in RDP. The only difference is that EVENODD generally involves more XOR computations than RDP in constructing parities.

C. Recovery of Single Node Failure

Since node failures are common in distributed storage systems [8], recovering any node failures becomes crucial to maintain the overall system reliability. In general, single-node failures occur more frequently than concurrent multi-node failures. To improve recovery performance, one fundamental requirement is to *minimize the number of symbol reads required for recovering a single-node failure*. This also corresponds to minimizing the amount of data being transferred among storage nodes in a distributed/networked



(a) Conventional recovery (b) Hybrid recovery

Figure 2. Two recovery approaches for RDP with $p = 7$.

storage system, in which network transmission bandwidth can limit the recovery performance.

Here, we use RDP as an example to illustrate two recovery approaches. To recover a single-node failure, the *conventional recovery* approach is to recover each lost symbol in the failed node *independently* [4, 5]. For example, Figure 2(a) shows the conventional recovery approach when Node 0 fails in RDP with $p = 7$ (i.e., 8 nodes). Each lost symbol in Node 0 can be independently recovered by the corresponding symbols in Nodes 1-6 in the same row. Thus, the total number of symbol reads is 36. Another recovery approach is called the *hybrid recovery* approach [27], whose main idea is to read the symbols in a combination of rows and diagonals that share the most common symbols. Figure 2(b) shows one possible solution of the hybrid recovery approach. Some read symbols can be used to recover more than one lost symbol. The total number of symbols read for repairing Node 0 is reduced to 27 (i.e., by 25%), which is the minimum. Note that [27] also extends the hybrid recovery approach to balance the number of symbols being read among the surviving nodes.

The key idea of the hybrid recovery approach is to maximize the number of shared common symbols being read from the surviving nodes. The hybrid recovery approach proposed in [27] also applies to the optimal single-node failure recovery in EVENODD as studied by [24].

Note that the hybrid recovery approach only addresses the recovery of data symbols rather than parity symbols. As shown in [27], each parity symbol needs to be recovered from all of its encoding original data symbols. Thus, by a single-node failure, we only assume the failure of a data node that stores data symbols.

D. Related Work

We now review related work on the failure recovery in distributed storage systems.

XOR-based erasure codes. In this work, we focus on XOR-based erasure codes, where the encoding/decoding operations are based on XOR operations. We assume that during recovery, there is a centralized controller that reads the required symbols from surviving nodes and reconstructs the lost symbols in the new node, thereby making the deployment feasible. For XOR-based RAID-6 codes, Xiang *et al.* [27] and Wang *et al.* [24] show the optimal single-node failure recovery solutions for RDP and EVENODD, respectively. Wu *et al.* [25] describe an efficient failure recovery approach for a new RAID-6 code that has the I/O load-balancing property. For arbitrary XOR-based erasure codes that can tolerate a general number of failures, Khan *et al.* [13] propose an exhaustive approach for the single-node failure recovery, while Zhu *et al.* [28] propose a greedy heuristic to speed up the search for an efficient recovery solution. However, existing recovery approaches for XOR-based erasure codes focus on minimizing the number of symbols being read from surviving nodes, but do not consider the heterogeneity among storage nodes. Greenan *et al.* [9] discuss the reliability of XOR-based erasure codes on heterogeneous devices. They mainly focus on the placement of data given the reliability costs of storage nodes, but do not address the failure recovery problem. To our knowledge, this is the first work that explicitly seeks to optimize failure recovery for XOR-based erasure codes under heterogeneous settings.

Regenerating codes. The recovery problem in distributed storage system was first discussed in [7], in which a class of codes called *regenerating codes* has been proposed to achieve an optimal tradeoff between the amount of storage space required for storing redundancy and the amount of data traffic being transferred during the recovery process. There are constructions of regenerating codes (e.g., [21, 23]). Existing studies on regenerating codes mainly assume that the same amount of data is retrieved from each surviving node. However, in practical distributed storage systems, the underlying networked environment may have different transmission bandwidths and topologies. Thus, each storage node will have different communication costs, leading to different optimal recovery solutions.

Based on regenerating codes, Li *et al.* [18] study the influence of network topologies on the recovery performance and proposes a tree-structured data regeneration scheme to reduce the communication overhead through exploiting the heterogeneity in node bandwidths. Li *et al.* [17] propose a pipelined regeneration approach that involves fewer surviving nodes during regeneration.

Regenerating codes (e.g., [7, 23]) typically require storage nodes to be programmed with encoding capabilities so as to generate encoded symbols on-the-fly during recovery. This may complicate the deployment of a distributed storage system in practice.

III. Cost-based Heterogeneous Recovery

In this section, we formulate the single-node failure recovery problem via an optimization model that accounts for node heterogeneity in a practical storage environment. We argue that the formulated optimization model has a very

large solution space, making the model difficult to solve in general. Thus, we propose an efficient *cost-based heterogeneous recovery (CHR)* algorithm to solve the problem, and show via simulations that CHR can provide optimal solutions in most cases.

To simplify our discussion, we focus on a storage system based on RDP. We also show how our analysis can be generalized for EVENODD.

A. Model Formulation

We consider an RDP-based distributed storage system with $p > 2$. According to RDP, the storage system has $p+1$ nodes, denoted by V_0, V_1, ..., V_p. Without loss of generality, we let nodes V_{p-1} and V_p store the row and diagonal parities, respectively. Let V_k be the failed (data) node, where $0 \leq k \leq p-2$ (note that we focus on recovering a failed data node only as discussed in Section II-C). Our goal is to recover all lost symbols of V_k. Suppose that the recovery operation reads y_i symbols from node V_i ($i \neq k$) per stripe. We associate a weight w_i with node V_i, where w_i denotes the cost of reading one symbol from node V_i. Our objective is to minimize the *total recovery cost* C, defined as the sum of the costs of reading the symbols from all surviving nodes (i.e., all nodes except V_k) per stripe. We can formulate an optimization problem as follows:

$$\text{Minimize} \quad C = \sum_{i=0, i\neq k}^{p} w_i y_i,$$

subject to the recovery condition that the lost symbols of V_k per stripe can be recovered from the read symbols. Note that different coding schemes have different recovery conditions. For example, to satisfy the recovery condition in RDP, it is necessary that (i) $\sum_{i\neq k} y_i \geq \frac{3(p-1)^2}{4}$ and (ii) $\frac{p-1}{2} \leq y_i \leq p-1$ for all $i \neq k$ [27]. We emphasize that satisfying only these two necessary conditions does not suffice to recover a failed node in RDP.

In general, the total recovery cost C can refer to different physical meanings, depending on how weight w_i is defined. We show several examples of different interpretations of C.

- If w_i = 1 for all i, then C represents the total number of symbols being read from surviving nodes, and the optimization problem is reduced to the traditional homogeneous setting that can be solved by hybrid recovery [27].
- If w_i denotes the inverse of the transmission bandwidth of node V_i, then C represents the total amount of transmission time to download the symbols from all surviving nodes (assuming that each storage node is accessed one-by-one).
- Many companies now outsource data storage to cloud storage providers (e.g., Amazon S3 and Windows Azure). To protect against cloud failures and avoid vendor lock-ins, cloud clients may stripe data across multiple cloud providers using erasure codes (e.g., in [1, 2]) or regenerating codes (e.g., in [11]). We can model w_i as the monetary cost of migrating per unit of data outbound from node V_i, which refers to a cloud storage provider. Then C represents the total monetary cost of migrating data from surviving nodes (or clouds).

Note that the weight information can be specified during the system setup or obtained by probing the performance of storage nodes from time to time. Note that in the latter case, it is crucial that the optimization model is solved *timely*; otherwise, the weight information may become outdated and no longer accurate. This motivates us to look into an efficient scheme to solve the optimization model, as discussed below.

B. Solving the Model

Given the optimization model, we propose several simplification strategies to make the model tractable. Such strategies lay the foundations of constructing our CHR algorithm (see Section III-C).

Reformulating a simplified model. First we need to determine a download distribution for which there exists a recovery strategy that can recover the failed node successfully. Then given a download distribution $\{y_0, \cdots y_{k-1}, y_{k+1}, \cdots, y_p\}$, it is necessary to determine what exactly the symbols need to be retrieved. Here, we reformulate the model based on recovery sequences [27]. A *recovery sequence* $\{x_0, x_1, \cdots, x_{p-2}, x_{p-1}\}$ specifies how each lost symbol (along the column) of a failed node V_k is recovered, such that $x_i = 0$ if $d_{i,k}$ (the i-th symbol of V_k) is recovered from the row parity sets, and $x_i = 1$ if $d_{i,k}$ is recovered from the diagonal parity sets. Note that $d_{p-1,k}$ lies in an imaginary row and is always recovered from its row parity set, meaning that $x_{p-1} = 0$ for any feasible recovery sequence. We point out that we include an imaginary row into a recovery sequence so that we can easily operate on a recovery sequence (see our Lemmas in the following discussion). For example, referring to Figure 2, the recovery sequences for the conventional and hybrid recovery approaches for Node 0 are $\{0000000\}$ and $\{1110000\}$, respectively. Since each lost symbol can be recovered from either the row or diagonal parity sets, a recovery sequence provides a feasible solution of single-node failure recovery.

The download distribution $\{y_0, \cdots y_{k-1}, y_{k+1}, \cdots, y_p\}$ and which symbols to be fetched can both be determined by the corresponding recovery sequence. As proven in [27], for a given recovery sequence $\{x_i\}_{0\leq i\leq p-1}$, the number of symbols being read from node V_j for any $j \neq k$ is given by:

$$y_j = (p-1) - (\sum_{i=0}^{p-1} x_i - \sum_{i=0}^{p-1} x_i x_{\langle i+j-k\rangle_p}), \qquad (1)$$

where $\langle i+j-k\rangle_p$ denotes $(i+j-k)$ modulo p.

We note that there are 2^{p-1} feasible recovery sequences (note that $x_{p-1} = 0$). Instead of enumerating all feasible recovery sequences, we assume that we narrow down our

search space by only considering the recovery sequences that issue the minimum number of symbols being read for recovery as returned by the hybrid recovery approach [27] (we call such recovery sequences to be *min-read recovery sequences*). In Section III-E, we show that with a very high probability, one of the min-read recovery sequences is the optimal solution that minimizes the total recovery cost.

By considering only the min-read recovery sequences, it is shown in [27] that exactly $(p-1)/2$ symbols in the failed node V_k will be recovered from the diagonal parity sets. This implies that $\sum_{i=0}^{p-1} x_i = (p-1)/2$. Thus, Equation (1) becomes:

$$y_j = (p-1)/2 + \sum_{i=0}^{p-1} x_i x_{\langle i+j-k \rangle_p}. \tag{2}$$

By summing the weighted version of Equation (2), the total recovery cost C is given by:

$$\begin{aligned} \sum_{j \neq k} w_j y_j &= \sum_{j \neq k} ((p-1)/2 + \sum_{i=0}^{p-1} x_i x_{\langle i+j-k \rangle_p}) w_j \\ &= \frac{p-1}{2} \sum_{j \neq k} w_j + \sum_{j \neq k} \sum_{i=0}^{p-1} x_i x_{\langle i+j-k \rangle_p} w_j. \end{aligned} \tag{3}$$

As $\frac{p-1}{2} \sum_{j \neq k} w_j$ is a constant value, we only need to minimize the second term of Equation (3). Thus, we reformulate the model as follows:

$$\begin{aligned} \text{Minimize} \quad & \sum_{j \neq k} \sum_{i=0}^{p-1} x_i x_{\langle i+j-k \rangle_p} w_j \\ \text{subject to} \quad & \sum_{i=0}^{p-1} x_i = (p-1)/2, \\ & x_i \in \{0,1\} \text{ for } i = 0, 1, \cdots, p-2, \\ & x_{p-1} = 0. \end{aligned} \tag{4}$$

Traversing unique recovery sequences. We have assumed that we only consider the min-read recovery sequences. However, there are still a large number of min-read recovery sequences, since the number of min-read recovery sequences is $\binom{p-1}{(p-1)/2}$ (i.e., for the set of x_i's where $i = 0, 1, \cdots, p-2$, we set $(p-1)/2$ of them to be 1). For example, when $p = 29$, there are 40,116,600 min-read recovery sequences. As a result, enumerating all min-read recovery sequences for the minimum total recovery cost remains computationally expensive.

Our key observation is that many min-read recovery sequences return the same download distribution $\{y_0, \cdots y_{k-1}, y_{k+1}, \cdots, y_p\}$. For example, referring to the RDP case in Figure 2, we can verify that the two min-read recovery sequence $\{1110000\}$ and $\{0111000\}$ have the same download distribution $\{y_0, \cdots y_{k-1}, y_{k+1}, \cdots, y_p\}$ = {5, 4, 3, 3, 4, 5, 3}. Thus, our goal is to traverse only the *unique* min-read recovery sequences that give distinct download distributions.

To find the number of unique min-read recovery sequences, we enumerate all min-read recovery sequences and determine their corresponding download distributions. Table I shows the results. We can see that the number of the unique min-read recovery sequences is significantly less than that of the min-read recovery sequences.

Table I
ENUMERATION RESULTS FOR RECOVERY IN RDP.

p	Total # of recovery sequences	# of min-read recovery sequences	# of unique min-read recovery sequences
5	16	6	2
7	64	20	4
11	1024	252	26
13	4096	924	74
17	65536	12870	698
19	262144	48620	2338
23	4194304	705432	28216
29	268435456	40116600	1302688

To help us identify the unique min-read recovery sequences, we propose two lemmas that show the conditions under which two recovery sequences have the same download distribution.

Lemma *1:* (Shift condition). Given a recovery sequence $\{x_i\}_{0 \leq i \leq p-1}$, let $\{x'_i\}_{0 \leq i \leq p-1} = \{x_{\langle i+r \rangle_p}\}_{0 \leq i \leq p-1}$ for any r $(0 \leq r \leq p-1)$. In other words, $\{x'_i\}_{0 \leq i \leq p-1}$ is a shifted recovery sequence of $\{x_i\}_{0 \leq i \leq p-1}$. Then:

1. If $x'_{p-1} = 1$, then $\{x'_i\}_{0 \leq i \leq p-1}$ is not a feasible recovery sequence.
2. If $x'_{p-1} = 0$, then both $\{x'_i\}_{0 \leq i \leq p-1}$ and $\{x_i\}_{0 \leq i \leq p-1}$ have the same download distribution.

Proof. We require that $x'_{p-1} = 0$ as the lost symbol in the last (imaginary) row must be recovered from row parity. Therefore, if $x'_{p-1} = 1$, $\{x'_i\}_{0 \leq i \leq p-1}$ is not feasible.

Suppose that $x'_{p-1} = 0$. From Equation (2), we can derive the download distribution for $\{x'_i\}_{0 \leq i \leq p-1}$ as:

$$\begin{aligned} y'_j &= (p-1)/2 + \sum_{i=0}^{p-1} x'_i x'_{\langle i+j-k \rangle_p} \\ &= (p-1)/2 + \sum_{i=0}^{p-1} x_{\langle i+r \rangle_p} x_{\langle \langle i+r \rangle_p + j-k \rangle_p}. \end{aligned}$$

Let $s = \langle i + r \rangle_p$. If i corresponds to the set of integers $\{0, 1, \cdots, p-1\}$, then we can see that s also corresponds to the same set of integers $\{0, 1, \cdots, p-1\}$. Thus, we can write $y'_j = (p-1)/2 + \sum_{s=0}^{p-1} x_s x_{\langle s+j-k \rangle_p}$. Thus, we have

$y_j = y'_j$. Both $\{x_i\}_{0\leq i\leq p-1}$ and $\{x'_i\}_{0\leq i\leq p-1}$ have the same download distribution. ∎

Remark: For example, let $r = 4$ and $p = 7$. Then both the recovery sequences $\{x_i\}_{0\leq i\leq 6}$ = $\{1110000\}$ and $\{x'_i\}_{0\leq i\leq 6}$ = $\{x_{\langle i+4\rangle_p}\}_{0\leq i\leq 6}$ = $\{0001110\}$ have the same download distribution.

Lemma *2:* (Reverse condition). Given a recovery sequence $\{x_i\}_{0\leq i\leq p-1}$, let $\{x'_i\}_{0\leq i\leq p-1} = \{x_{p-1-i}\}_{0\leq i\leq p-1}$. In other words, $\{x'_i\}_{0\leq i\leq p-1}$ is a reverse recovery sequence of $\{x_i\}_{0\leq i\leq p-1}$. Then:

1. If $x'_{p-1} = 1$, then $\{x'_i\}_{0\leq i\leq p-1}$ is not a feasible recovery sequence.
2. If $x'_{p-1} = 0$, then both $\{x'_i\}_{0\leq i\leq p-1}$ and and $\{x_i\}_{0\leq i\leq p-1}$ have the same download distribution.

Remark: The proof of Lemma 2 is omitted as it is similar to that of Lemma 1. For example, both the recovery sequences $\{x_i\}_{0\leq i\leq 6}$ = $\{0110100\}$ and $\{x'_i\}_{0\leq i\leq 6}$ = $\{x_{p-1-i}\}_{0\leq i\leq 6}$ = $\{0010110\}$ have the same download distribution.

C. Cost-based Heterogeneous Recovery (CHR) Algorithm

In this section, we present the *cost-based heterogeneous recovery (CHR)* algorithm for single-node failure recovery under heterogeneous storage environments. The main idea of CHR is as follows. Given the weight of each surviving node in a storage system, CHR enumerates all the unique min-read recovery sequences and calculates their respective total recovery costs. It then returns the recovery sequence with the minimum total recovery cost. This min-cost recovery sequence will specify the symbols being retrieved from the surviving nodes.

Notation. We summarize the major notation being used in the CHR algorithm, using RDP as an example. Recall that given the prime number p, there are $n = p+1$ storage nodes $V_0, V_1, \cdots, V_p$, and we assume that V_k (where $0 \leq k \leq p-2$) is the failed node. Each node V_i (where $0 \leq i \leq p$) is associated with a weight w_i, which is defined according to the specific optimization objective. We enumerate each recovery sequence R and evaluate its recovery cost C. Our goal is to find the min-cost recovery sequence (denoted by R^*) with the minimum total recovery cost C^*.

Given a feasible recovery sequence $\{x_i\}_{0\leq i\leq p-1}$, we always have $x_{p-1} = 0$, so we can treat the sequence $\{x_{p-2}x_{p-1}\cdots x_0\}$ as a $(p-1)$-bit binary representation of an integer value denoted by v. We let F be a bitmap that identifies whether a min-read recovery sequence has already been enumerated, such that the bit $F[v]$ (where $0 \leq v \leq (2^{p-1}-1)$) is initialized to 0, and will be set to 1 if the recovery sequence with integer value v has been enumerated. Note that the size of F is 2^{p-1} bits, which is exponential in p. Nevertheless, the stripe sizes of systems used in practice tend to stay within a medium range [20]. Thus, the memory overhead remains feasible for a reasonable range of p. For example, for a storage system of at most $n = p + 1 = 30$ nodes, the size of F is at most 32MB, which is manageable for today's commodity systems.

Functions. The CHR algorithm is built on three functions.

- NEXTRS(p, R). The function seeks to generate all min-read recovery sequences, each containing $\frac{p-1}{2}$ 1-bits and $\frac{p+1}{2}$ 0-bits. NEXTRS is derived from Algorithm R [14], which can be used to generate all (s, t) combinations of s 0-bits and t 1-bits in lexicographic order (in our case, $s = \frac{p-1}{2}$ and $t = \frac{p-1}{2}$). We adapt Algorithm R into NEXTRS, such that given p and R, it outputs the next recovery sequence in order, or NULL if all recovery sequences have been enumerated. Note that each invocation of NEXTRS only re-orders a few bits, and has minimal computational overhead in general.
- DUPLICATERS(p, F, R). Given a min-read recovery sequence R, the function generates all its shifted recovery sequences and their corresponding reverse recovery sequences, all of which have the same recovery cost as R according to Lemmas 1 and 2, respectively. For each generated sequence, DEPLICATERS checks if it is feasible (i.e., $x_{p-1} = 0$); if so, it finds the binary representation of the sequence and its corresponding integer value v, and sets $F[v] = 1$.
- DOWNLOADS(R, i). Given the recovery sequence R and the identifier i of surviving node V_i, the function returns the number of symbols read from node V_i.

Algorithm details. Figure 3 shows the pseudo-code of the CHR algorithm. First, we initialize different variables (Steps 1-5). We initialize the array F to all 0-bits (Step 2). We also initialize the considered recovery sequence R to have $\frac{p-1}{2}$ 1-bits followed by $\frac{p+1}{2}$ 0-bits (Step 3).

Then we find the min-cost recovery sequence R^* among all min-read recovery sequences (Steps 6-19). For each recovery sequence R we consider, if it has not been enumerated, then we first mark all the shifted and reverse recovery sequences of R as being enumerated using the function DUPLICATERS (Step 11). We then calculate the recovery cost C of R (Step 12), and identify whether it is the min-cost recovery sequence among all the currently enumerated recovery sequences (Steps 13-15). We then select the next recovery sequence R to be considered using the function NEXTRS (Step 18). We repeat the process until all min-read recovery sequences are enumerated.

Finally, we compare R^* that we have found with the baseline conventional recovery approach (Steps 20-25). Our observation is that there are circumstances where the conventional recovery approach outperforms all min-read recovery sequences. Recall that the conventional approach uses only the row parity sets for recovery (see Section II), while the hybrid and CHR approaches uses both row and diagonal parity sets. In some situations, the storage node that stores the diagonal parities may have a significantly high weight

Algorithm: CHR(p, k, $\{w_i\}$)

Input:
p: prime number
k: identifier of the failed node
$\{w_i\}$: set of weights of storage nodes V_i ($0 \leq i \leq p+1$)
Output:
R^*: min-cost recovery sequence

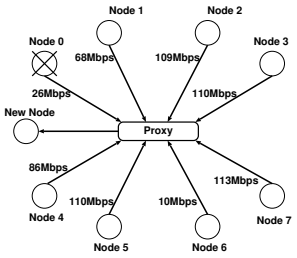
1: /* *Initialization* */
2: Initialize $F[0 \cdots 2^{p-1}-1]$ with 0-bits
3: Initialize R with $\frac{p-1}{2}$ 1-bits followed by $\frac{p+1}{2}$ 0-bits
4: Initialize $R^* = R$
5: Initialize C^* = MAX_VALUE
6: /* *Find R^* among all min-read recovery sequences* */
7: **while** $R \neq$ NULL **do**
8: Convert R into integer value v
9: **if** $\mathcal{F}[v]$ = 0 **then**
10: $\mathcal{F}[v]$ = 1
11: DUPLICATERS(p, F, R)
12: $C = \sum_{i \neq k}$ (DOWNLOADS(R, i) $\times w_i$)
13: **if** $C < C^*$ **then**
14: $C^* = C$
15: $R^* = R$
16: **end if**
17: **end if**
18: $R =$ NEXTRS(p, $tmp\mathcal{R}$)
19: **end while**
20: /* *Compare with the conventional approach* */
21: $R =$ all 0-bits
22: $C = \sum_{i \neq k}$ (DOWNLOADS(R, i) $\times w_i$)
23: **if** $C \leq C^*$ **then**
24: $R^* = R$
25: **end if**
26: **return** R^*

Figure 3. The cost-based heterogeneous recovery (CHR) algorithm.

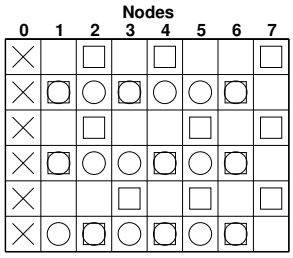
(e.g., with very limited transmission bandwidth). In this case, the conventional recovery approach, even reading more symbols, still has better recovery performance. We first initialize the recovery sequence R with all 0-bits (Step 21), meaning that each lost symbol is recovered from its row parity set. Then we compare whether its recovery cost is no greater than C^*; if so, we switch to use the conventional approach. This baseline testing ensures that the recovery performance of the CHR algorithm is always no worse than that of the conventional approach.

D. Example

We illustrate via an example the main idea of the CHR algorithm and how it improves the recovery performance over the conventional and hybrid approaches. Figure 4(a) shows a star-based distributed storage system where a centralized proxy connects to 8 storage nodes, each of which has link transmission bandwidth following a uniform distribution U(0.3Mbps, 120Mbps), which has also been used in prior work on distributed storage [18] to mimic a PlanetLab-like environment [16]. The storage system uses RDP with $p = 7$ as its coding scheme. Each storage node stores $p-1 = 6$ data



(a) Heterogeneous storage environment



(b) Cost-based Heterogeneous Reocvery Scheme

Figure 4. An example for CHR.

blocks (i.e., symbols) of size α (in the unit of Mb). Suppose now that node 0 fails. The proxy then downloads data from the other surviving nodes and regenerates the lost data of Node 0 in a new node. In this example, we let w_i be the inverse of the link transmission bandwidth of node i (e.g., $w_0 = \frac{1}{26Mbps}$, $w_1 = \frac{1}{68Mbps}$, etc.). The total recovery cost is thus interpreted as the total time of reading data blocks from each surviving node one by one (see Section III-A).

CHR approach. The CHR algorithm works as follows. We initialize the array F with 0-bits, R = $\{1110000\}$, and the recovery cost C = MAX_VALUE. The integer value of R is 7. As $F[7] = 0$, we set $F[7] = 1$. We also find out all shifted and reverse recovery sequences of R. For example, we use the shift condition to find the following: 1110000 $\rightarrow$ 1100001 $\rightarrow$ 1000011 $\rightarrow$ 0000111 $\rightarrow$ 0001110 $\rightarrow$ 0011100 $\rightarrow$ 0111000. The 2nd, 3rd, and 4th recovery sequences are not feasible as $x_{p-1} \neq 0$. Thus, we set $F[56] = F[28] = F[14] = 1$. We also find the reverse of each of the shifted sequences and set the corresponding element of F to be 1.

Now, we calculate the recovery cost C for $R = \{1110000\}$. We can verify that the proxy will (i) read 3 blocks from Nodes 3, 4, and 7, (ii) read 4 blocks from Nodes 2 and 5, and (iii) read 5 blocks from Nodes 1 and 6 for recovery. The recovery cost C will be:

$$\frac{5\alpha}{68} + \frac{4\alpha}{109} + \frac{3\alpha}{110} + \frac{3\alpha}{86} + \frac{4\alpha}{110} + \frac{5\alpha}{10} + \frac{3\alpha}{113}$$
$$= 0.7353\alpha \text{ (in sec)}.$$

Note that this is one of the optimal results of the hybrid recovery approach [27] (see Figure 2(b)), since it corresponds to a min-read recovery sequence that reads the minimum number of blocks.

We now compare the recovery cost C with C^*. Since the former is smaller for the first iteration, we replace C^* with C and R^* with $R = \{1110000\}$. At the end of this loop, we use NEXTRS to operate on the next min-read recovery sequence. Finally, we can find that $R^* = \{1010100\}$ is the min-cost recovery sequence, whose download distribution is

shown in Figure 4(b). The minimum recovery cost is:

$$\frac{3\alpha}{68}+\frac{5\alpha}{109}+\frac{4\alpha}{110}+\frac{4\alpha}{86}+\frac{5\alpha}{110}+\frac{3\alpha}{10}+\frac{3\alpha}{113}$$
$$= 0.5449\alpha \text{ (in sec)}.$$

Note that CHR reduces the recovery cost of the first considered recovery sequence $\{1110000\}$ (which is an optimal result of the hybrid approach) by 25.89%.

Conventional approach. Note that CHR significantly reduces the recovery cost of the conventional approach. Using the conventional approach (see Figure 2(a) in Section II), the proxy reads 6 blocks from each of Nodes 1 to 6. The recovery cost is:

$$\frac{6\alpha}{68}+\frac{6\alpha}{109}+\frac{6\alpha}{110}+\frac{6\alpha}{86}+\frac{6\alpha}{110}+\frac{6\alpha}{10}$$
$$= 0.9221\alpha \text{ (in sec)}.$$

We observe that CHR reduces the recovery cost by 40.91% over the conventional approach.

E. Simulation Studies

In this subsection, we use simulation to quantitatively evaluate the efficiency of our proposed CHR algorithm in terms of the computational overhead of the algorithm as well as the recovery performance.

Our simulations consider a star-like network topology as in Figure 4(b), where a centralized proxy connects to a number of nodes, each having link transmission bandwidth following a uniform distribution U(0.3Mbps, 120Mbps) [16]. We again let w_i be the inverse of the link transmission bandwidth of each node i.

Traverse efficiency. We first evaluate the computational time of the CHR algorithm in traversing all recovery sequences. Our goal is to justify that the CHR algorithm can determine the min-cost recovery solution within a reasonable time frame. Recall that the CHR algorithm only computes the recovery costs of the *unique* min-read recovery sequences (see Section III-B). We compare with the *naive traverse* approach that enumerates all of the $\binom{p-1}{(p-1)/2}$ min-read recovery sequences and computes their recovery costs. Here, we evaluate the traverse time using a Linux-based desktop computer running with 3.2GHz CPU and 2GB RAM. All approaches we consider are implemented in C. We consider different values of p from 5 to 29, and obtain the averaged results over 100 runs.

Table I shows the results. By enumerating only the unique min-read recovery sequences, CHR significantly reduces the traverse time of the naive approach by over 90% as p increases. For example, when $p = 29$, the naive traverse time is over 12 minutes (per run), while CHR reduces the traverse time to around 45.4 seconds (per run). As stated in Section III-C, the range of p that we consider is expected to be feasible in actual usage [20].

Table II
TRAVERSE EFFICIENCY COMPARISON FOR RDP WITH DIFFERENT p.

p	Naive traverse time (ms)	CHR's traverse Time (ms)	Improved rate (%)
5	0.0220	0.0100	54.55
7	0.0950	0.0310	67.37
11	2.3160	0.3910	83.12
13	11.9840	1.6150	86.52
17	107.7410	10.0790	90.65
19	455.2760	40.5370	91.10
23	9230.7800	691.2800	92.51
29	752296.2700	45423.5570	93.96

Robustness efficiency. Recall that the CHR approach only considers a simplified recovery model that enumerates only the min-read recovery sequences (see Equation (4) in Section III-B). We would like to see if the CHR approach actually achieves the *global optimal* among all the 2^{p-1} feasible recovery sequences (which read more symbols than the min-read sequences in general).

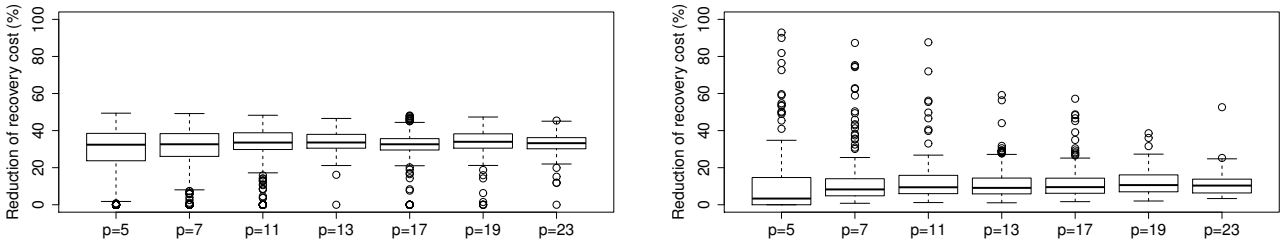
Here, we compute the minimum recovery cost of the global optimal recovery sequence by enumerating all the 2^{p-1} feasible solutions. We compare the global optimal result with that of CHR. We repeat the comparison over 1000 simulation runs.

Table III shows the results, in which we show the probability that CHR actually returns the global optimal result (the 2nd column) and the maximum percentage reduction of the recovery time of the global optimal result over CHR (the 3rd column) for each p. We observe that CHR has a very high probability (over 93%) to hit the global optimal recovery cost. Also, the recovery cost of the global optimal solution is less than that of CHR by at most 6.46% only. Therefore, CHR is robust in returning a global optimal solution in most cases, while significantly reducing the computational time.

Table III
ROBUSTNESS EFFICIENCY FOR CHR.

p	Hit Global Optimal Probability(%)	Global Optimal Max Improvement (%)
5	94.9	6.12
7	94.5	5.54
11	93.6	5.98
13	93.2	6.46
17	92.8	5.97
19	93.1	5.73

Recovery efficiency. We evaluate via simulations the recovery efficiency of CHR in a heterogeneous storage environment. We conduct 100 simulation runs for each value



(a) CHR vs. Conventional

(b) CHR vs. Hybrid

Figure 5. Percentage reduction of recovery cost over (a) the conventional approach and (b) the hybrid approach.

of p and show the results via *box plots*. A box plot shows the minimum, lower quartile, median, upper quartile, maximum of the 100 results, and provides the outliers as well. Figures 5(a) and 5(b) show the box plots of the distributions of the recovery improvements of CHR in terms of the percentage reduction of the total recovery cost over the conventional and hybrid recovery approaches, respectively. We observe that CHR can reduce the recovery cost by up to 50% and 30% (e.g., when $p = 5$) over the conventional and hybrid approaches, respectively.

From Figure 5(b), CHR can reduce the recovery cost of the hybrid approach by over 80% in few simulation runs (e.g., for $p = 5, 7, 11$). The main reason is that in those runs, the diagonal parity node happens to have a very small transmission bandwidth (and hence a large weight w_i). Since the hybrid approach uses diagonal parity sets to minimize the number of blocks read, it has a very high recovery cost. On the other hand, in those runs, CHR will switch to the conventional approach (see the last portion of the algorithm in Figure 3), and its performance improvement over the conventional approach is 0% (see Figure 5(a)).

F. Generalization to EVENODD

Our discussion of CHR thus far focuses on the single-node failure recovery for RDP [5]. For the single-node failure recovery for EVENODD [4], we can apply the CHR approach in a similar way with two observations. First, based on the EVENODD construction, each lost data symbol of EVENODD can be recovered from either its row or diagonal parity sets. This implies that we can construct a recovery sequence as in the RDP case. Second, it is shown that for EVENODD, exactly $(p-1)/2$ lost symbols must be recovered from the diagonal parity sets [27]. Thus, we can propose the same simplified recovery model in Equation (4) (see Section III-B). Note that Lemmas 1 and 2 are also applicable for EVENODD, so we can enumerate only the unique min-read recovery sequences for EVENODD as well.

IV. Testbed Experiments

In this section, we experiment different recovery approaches using a networked storage system testbed deployed in a heterogeneous environment. Specifically, we consider three single-node failure recovery approaches: (i) the *conventional* approach, which recovers each lost symbol independently (see Section II-C), (ii) the *hybrid* approach [24, 27], which minimizes the number of symbols read by using a combination of parity sets in recovery (see Section II-C), and (iii) the *CHR* approach, which seeks to minimize the recovery cost by associating a weight with each storage node (see Section III).

Our goal is to validate the performance improvement of CHR over the conventional and hybrid approaches in a real, heterogeneous network environment. We emphasize that our testbed experiments are distinct from our numerical simulations (see Section III-E) and disk-based simulations [27], since we experiment the actual read/write operations via physical storage devices. The resulting recovery performance is determined by the hardware configurations of the testbed. We expect that testbed experiments can provide more realistic results than simulations.

A. Methodology

Our experiments are based on an open-source networked storage system called NCFS [12]. It is a networked file system that interconnects different storage nodes and stripes data across the nodes according to the underlying coding scheme. In our experiments, we focus on RDP and EVENODD, and integrate them into the NCFS implementation.

NCFS includes a recovery utility that can recover a single-node failure. The recovery operation of NCFS includes three steps: (i) reading data from other surviving storage nodes, (ii) reconstructing the lost data, and (iii) writing data to a new storage node. We implement the three recovery approaches (i.e., conventional, hybrid, and CHR) in NCFS. Note that both the hybrid and CHR approaches only focus on optimizing the read part (i.e., step (i)), but we argue (in Experiment 1) that it remains a good approximation in practice since the read part accounts for the majority of the total recovery time.

We set up the testbed topology as follows. We deploy NCFS on a Linux server with Intel Quad-Core 2.66GHz CPU and 4GB RAM. We connect the NCFS server to a

number of storage devices (or nodes) over a Gigabit Ethernet switch using the ATA over Ethernet protocol [10]. The set of storage nodes is composed of a mixture of personal computers (PCs) or network-attached storage (NAS) devices. Each storage node is equipped with an Ethernet interface card with a different network speed that is either 100Mbps and 1Gbps. Table IV shows the compositions of storage nodes in our testbed for the RDP case for different values of p that we consider (i.e., the total number of nodes is $n = p + 1$). Note that for EVENODD, the total number of nodes is $n = p + 2$, and we add another storage node of 100Mbps to our testbed. The testbed is configured to have a heterogeneous setting, as the storage nodes have different transmission speeds.

Table IV
COMPOSITIONS OF STORAGE NODES FOR RDP IN OUR TESTBED.

p	Total # of of nodes ($n = p + 1$)	# of nodes with 100Mbps	# of nodes with 1Gbps
5	6	2	4
7	8	3	5
11	12	5	7
13	14	6	8
17	18	9	9

Default settings. By default, we let each symbol refer to a chunk of size 1MB, which has been considered in some existing distributed file systems (e.g., [22]). The chunk size is larger than a typical file system block size (e.g., the default block size of Linux file systems is 4KB). We expect that it is natural for distributed storage systems to operate on large-size chunks (e.g., 64MB in GFS [8]). We will evaluate the impact of different chunk sizes (see Experiment 2). In addition, to trigger a recovery operation, we disable the storage node in the leftmost column (i.e., Node 0 in Figure 1). We will also evaluate the impact of disabling other nodes (see Experiment 3).

Metric. We consider the metric *recovery time* (per MB of data being recovered) needed to perform a recovery operation. The average recovery time is obtained over 10 runs. In each run, we write nGB of data to NCFS, where n is the number of storage nodes that we consider. NCFS will stripe the data across the storage nodes. We disable a storage node, and reconstruct the lost data in another spare storage node of network speed 1Gbps. The recovery utility reads the data symbols from each surviving node one by one.

In CHR, we associate each storage node with a weight given by the inverse of the speed of its Ethernet interface card. Note that the weight can be more accurately determined, for example, by probing measurements. Nevertheless, we validate in the following experiments that such simple weight assignments suffice to bring improvements in the recovery performance. Here, the total recovery cost reflects the total recovery time (see Section III-A).

B. Results

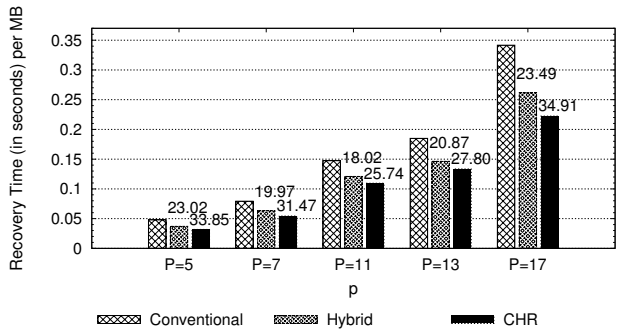
Experiment 1 (Recovery time performance for different numbers of storage nodes). We first evaluate the recovery time performance of different recovery approaches by varying the value of p, and hence the number of storage nodes. Recall that both the hybrid and CHR approaches seek to improve the read performance in the recovery process. That is, the hybrid approach minimizes the number of symbols (or chunks) read from surviving nodes, and CHR further improves the read performance by taking into account the network transmission bandwidths. Figures 6(a) and 6(b) show the performance for the read part in the recovery operation for RDP and EVENODD, respectively. Compared to the conventional approach, the hybrid approach reduces the time of the data read part by 18.02% to 23.49%, while CHR further reduces the time by 25.74% to 34.91%. Specifically, CHR improves the hybrid approach by up to 14.92% and 20.03% for RDP and EVENODD when $p = 17$, respectively.

We also consider the overall performance of the complete recovery operation, which also includes the steps of reconstructing the lost data in NCFS and writing the reconstructed data to a new node. Figures 7(a) and 7(b) show the total recovery time performance of different recovery approaches for RDP and EVENODD, respectively. We again observe that CHR constantly outperforms the hybrid and conventional approaches for different values of p. Take RDP for $p = 11$ as an example. Compared to the conventional approach, the total recovery times for the hybrid and CHR approaches are reduced by 15.28% and 21.45%, respectively. In particular, CHR reduces the overall recovery time of the hybrid approach by up to 13.15% and 16.89% for RDP and EVENODD, respectively.

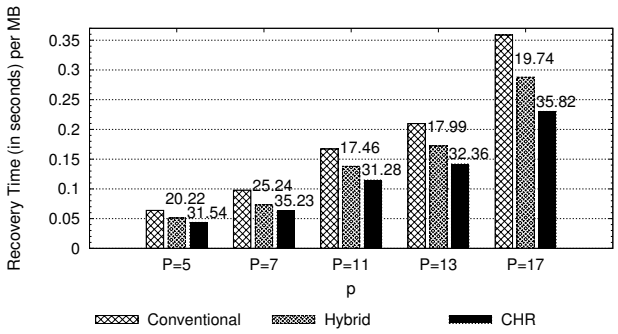
From both Figures 6 and 7, we observe that the read time (in Figure 6) accounts for at least 68.61% of the total recovery time (in Figure 7). This also justifies why the CHR and hybrid approaches [27] seek to optimize the read part only. Also, CHR works better in a heterogeneous environment compared to the hybrid approach.

Experiment 2 (Performance with different chunk sizes). We now evaluate the impact of chunk size on the recovery time performance. We vary the chunk size from 256KB to 4MB. We fix $p = 11$ and hence the number of storage nodes.

Figures 8(a) and 8(b) show the total recovery times for RDP and EVENODD, respectively. We observe that as the recovery time decreases as the chunk size increases. In addition, the improvements of both the hybrid and CHR approaches over the conventional approach increase with the chunk size. The main reason is that with a larger chunk size, the number of I/O accesses decreases. Thus, the recovery

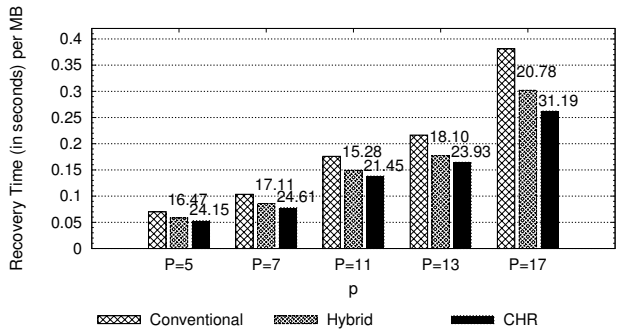


(a) Read Part Comparison for RDP

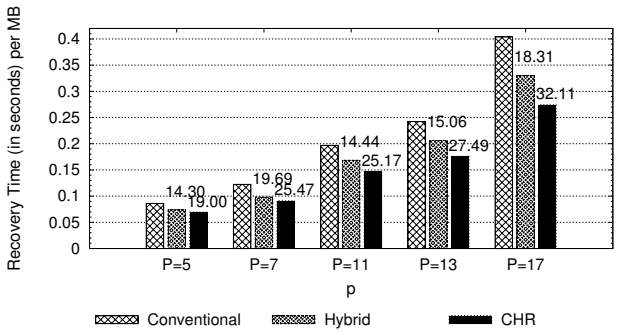


(b) Read Part Comparison for EVENODD

Figure 6. Experiment 1: Read part performance comparison during recovery for p ranging from 5 to 17. Each number corresponds to the percentage reduction over the conventional approach (same for other figures).



(a) Overall Comparison for RDP



(b) Overall Comparison for EVENODD

Figure 7. Experiment 1: Overall recovery time performance comparison for p ranging from 5 to 17.

time performance becomes more dominated by the amount of data being read. Take RDP with chunk size 4MB as an example. Compared to the conventional approach, the total recovery time reduction of the hybrid approach is 19.82%, while that of CHR is 25.66%. It is important to note that the improvement of CHR over the hybrid approach is consistent over all chunk sizes.

Experiment 3 (Performance with different failed nodes). We now evaluate the recovery time performance when the failed node is in a different column. We set the chunk to be 1MB, and fix $p = 11$. Figure 9 shows the total recovery time for RDP, while similar results are observed for EVENODD and omitted here in the interest of space. Both the hybrid and CHR approaches have varying recovery performance improvements over the conventional approach across different failed nodes. Nevertheless, CHR outperforms the hybrid approach regardless of which node fails. Compared to the recovery approach, the total recovery time of the hybrid approach is reduced by 10.04-19.44%, while the total recovery time of CHR is reduced by 17.90-25.12%.

V. Conclusions

We address single-node failure recovery of a RAID-6-coded storage system with heterogeneous types of storage nodes. We formulate an optimization model, and propose a cost-based heterogeneous recovery algorithm to minimize the total recovery cost of single-node failure recovery. We simplify the model to improve the computational efficiency of the algorithm, while still effectively minimize the total recovery cost. Through extensive simulations and testbed experiments, we justify the effectiveness of the CHR algorithm in achieving efficient single-node failure recovery in a heterogeneous storage environment. The source code of the CHR algorithm is available at: **http://ansrlab.cse.cuhk.edu.hk/software/chr**.

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.

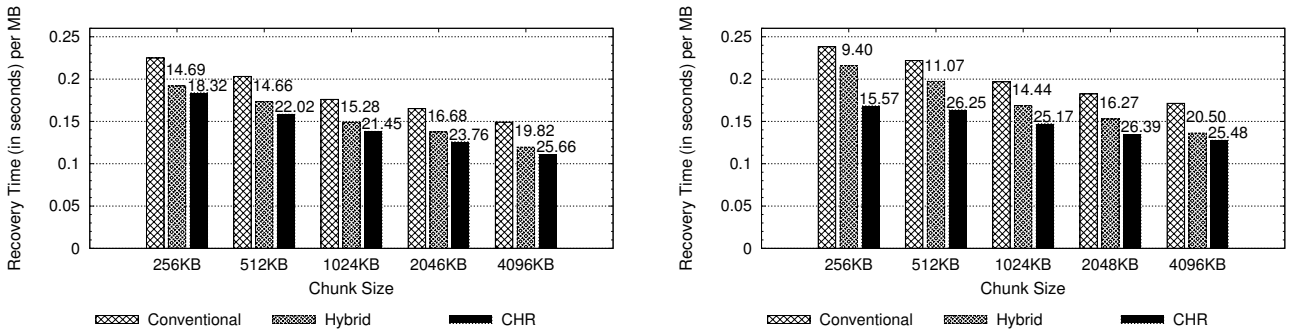
References

[1] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon. RACS: A Case for Cloud Storage Diversity. In *Proc. of ACM SOCC*, 2010.

[2] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. DEPSKY: Dependable and Secure Storage in a Cloud-of-Clouds. In *Proc. of ACM EuroSys*, 2011.

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

[4] 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.



(a) RDP($p = 11$)

(b) EVENODD($p = 11$)

Figure 8. Experiment 2: Total recovery time versus different chunk sizes (from 256KB to 4MB).

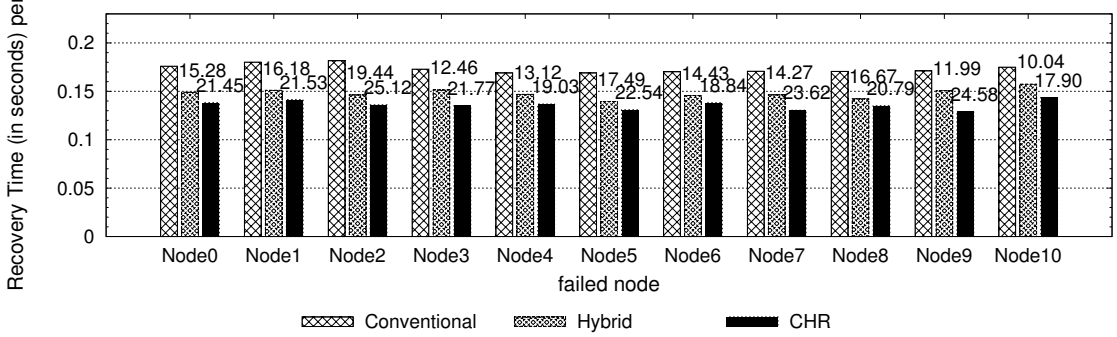


Figure 9. Experiment 3: Total recovery time versus different failed nodes for RDP.

[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] 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.

[7] 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.

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

[9] K. Greenan, E. Miller, and J. Wylie. Reliability of Flat XOR-based Erasure Codes on Heterogeneous Devices. In *Proc. of IEEE DSN*, 2008.

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

[11] Y. Hu, H. Chen, P. Lee, and Y. Tang. NCCloud: Applying Network Coding for the Storage Repair in a Cloud-of-Clouds. In *Proc. of USENIX FAST*, 2012.

[12] Y. Hu, C. Yu, Y. Li, P. Lee, and J. Lui. NCFS: On the practicality and extensibility of a network-coding-based distributed file system. In *Proc. of NetCod*, July 2011.

[13] 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.

[14] D. E. Knuth. The Art of Computer Programming, 1999.

[15] 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.

[16] S. Lee, P. Sharma, S. Banerjee, S. Basu, and R. Fonseca. Measuring Bandwidth Between PlanetLab Nodes. In *Proc. of PAM*, 2005.

[17] J. Li, X. Wang, and B. Li. Pipelined Regeneration with Regenerating Codes for Distributed Storage Systems. In *Proc. of NetCod*, 2011.

[18] 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.

[19] C. Lueth. RAID-DP: Network Appliance Implementation of RAID Double Parity for Data Protection. Technical report, NetApp, 2006.

[20] 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*, 2009.

[21] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran. Explicit Construction of Optimal Exact Regenerating Codes for Distributed Storage. In *Allerton Conference*, 2009.

[22] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *Proc. of USENIX FAST*, 2002.

[23] C. Suh and K. Ramchandran. Exact-Repair MDS Codes for Distributed Storage Using Interference Alignment. In *Proc. of IEEE ISIT*, 2010.

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

[25] 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.

[26] Wuala. http://www.wuala.com.

[27] 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.

[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.