# Single Disk Failure Recovery for X-code-based Parallel Storage Systems

Silei Xu, Runhui Li, Patrick P. C. Lee, Yunfeng Zhu, Liping Xiang, Yinlong Xu, John C.S. Lui

**Abstract**—In modern parallel storage systems (e.g., cloud storage and data centers), it is important to provide data availability guarantees against disk (or storage node) failures via redundancy coding schemes. One coding scheme is X-code, which is double-fault tolerant while achieving the optimal update complexity. When a disk/node fails, recovery must be carried out to reduce the possibility of data unavailability. We propose an X-code-based optimal recovery scheme called Minimum-Disk-Read-Recovery (MDRR), which minimizes the number of disk reads for single-disk failure recovery. We make several contributions. First, we show that MDRR provides optimal single-disk failure recovery and reduces about 25% of disk reads compared to the conventional recovery approach. Second, we prove that any optimal recovery scheme for X-code cannot balance disk reads among different disks within a single stripe in general cases. Third, we propose an efficient logical encoding scheme that issues balanced disk read in a group of stripes for any recovery algorithm (including the MDRR scheme). Finally, we implement our proposed recovery schemes and conduct extensive testbed experiments in a networked storage system prototype. Experiments indicate that MDRR reduces around 20% of recovery time of the conventional approach, showing that our theoretical findings are applicable in practice.

**Keywords**—parallel storage systems, coding theory, data availability, recovery algorithm.

## 1 Introduction

A fundamental requirement of building large-scale parallel storage systems is to make sure information is reliable and available for a long period of time. To achieve high reliability and availability in the face of component failures, redundancy techniques have been widely used in modern parallel/distributed storage systems, such as cloud storage, data centers, and peer-to-peer storage. For example, GFS [1] and Dynamo [2] use replication, while OceanStore [3], Total Recall [4], and Wuala [5] use erasure codes (e.g., one form of Reed-Solomon code [6]).

Full replication is the simplest way to generate redundant data. Exact copies of the original data are stored in multiple disks (or storage nodes), each of which keeps one copy. However, replication comes with a substantially high storage cost. Another form of redundancy is to use *Maximum Distance Separable (MDS)* codes, defined by parameters $n$ and $k$. An $(n, k)$ MDS code (e.g., Reed-Solomon code [6] and optimal erasure codes) is to divide the original file of size $M$ into $k$ equal-size fragments of size $M/k$ each, and then encode the $k$ data fragments into $n$ fragments, each of which also has the same size $M/k$. The MDS property states that any $k$ out of the $n$ encoded data fragments can be used to reconstruct the original file. The main advantage of MDS codes is that one can achieve optimal tradeoff between storage cost and data reliability. Compared with full replication, MDS codes can achieve orders of magnitude higher reliability with similar storage and bandwidth requirements [7]. Furthermore, full replication implies deploying more disks and that energy cost is one of the major concerns for data center [8], [9]. Thus, we expect that MDS codes are more preferred over full replication in many practical scenarios.

A special family of MDS codes is called *MDS array codes*, which are designed to provide fault-tolerant storage for RAID systems against double-disk failures (e.g., RDP code [10], EVENODD code [11], X-code [12]) or triple-disk failures (e.g., STAR code [13]). MDS array codes have an attractive property that they are computationally efficient, since their encoding and decoding processes use only XOR operations. In this work, we focus on RAID-6 array codes (e.g., RDP, EVENODD, X-code), which tolerate any two concurrent disk failures.

RAID-6 array codes can be categorized into two classes [14]. The first class is the *horizontal codes*, such as RDP and EVENODD, where original data fragments are stored in data disks while encoded fragments (known as *parities*) are stored in dedicated parity disks (also known as P and Q disks). Another class is *vertical codes*, such as X-code, where parities are distributed across all disks.

When a disk failure occurs in a parallel storage system, it is important to recover (or repair) the erased data in the failed disk as quickly as possible to maintain the system reliability guarantees [15]. It is challenging to repair a failed disk when MDS codes, or specifically MDS array codes, are used. A conventional approach is to download the entire file and reconstruct the original data, and then regenerate the data fragments of the failed disk. The conventional approach will cost a great deal of data transmission [16]. The total amount of data


- *S. Xu, Y. Zhu, L. Xiang, and Y. Xu are with School of Computer Science and Technology, University of Science & Technology of China (emails: {xusilei,zyfl,xlping}mail.ustc.edu.cn, ylxu@ustc.edu.cn)*
- *R. Li, P. P. C. Lee, and J. C. S. Lui are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong (emails: {rhli,pclee,cslui}@cse.cuhk.edu.hk).*
- *Corresponding author: Patrick P. C. Lee (pclee@cse.cuhk.edu.hk).*

that must be processed during recovery plays a crucial role in recovery time and affects the system service performance [10], [17]. This is particularly important in parallel storage systems, where network transmission bandwidth is a potential performance bottleneck.

### 1.1 Related Work

Recent research studies (see survey in [18]) propose a new class of MDS codes called the *regenerating codes* to reduce the amount of data needed to recover a failed disk/node in a parallel/distributed storage system. Authors of [19], [20] derive the cut-set lower bound of the amount of data needed for recovery based on the network coding theory [21]. Authors of [22], [23] propose the constructions of new MDS codes that can achieve improved recovery performance. Hu *et al.* [24] further consider cooperative recovery for multi-node failures. However, regenerating codes have yet to see practical deployment, possibly due to several constraints. Most regenerating codes (e.g., [25]) require storage nodes be programmable to support the encoding capability for recovery, thereby limiting the deployable platforms for practical storage systems. Some regenerating codes (e.g., [26]) can be implemented without the encoding capability of a storage node, but generally introduce higher storage overhead than traditional erasure codes. Note that their encoding operations involve linear computations on finite fields, and are more computationally expensive than XOR-based MDS array codes.

Only few research studies (e.g., [27], [28], [29], [30]) consider the recovery problem of RAID-6 array codes. Authors of [27] give lower bounds of repair bandwidth for single disk failure recovery with different codes, and they show that the lower bound is $(3p^2 - 2p + 5)/4$ for X-code. However, they do not present a formal proof for the tight lower bound of X-code. Also, they do not consider the load balancing problem among different disks and provide experimental evaluation. Our work addresses all the above issues. Authors of [28] consider the single-disk recovery problem for a particular type of MDS array codes called the RDP code. By using double parities for single-disk failure recovery, they propose an optimal recovery algorithm that reduces almost 25% of disk reads for recovery. In [28], an optimal recovery algorithm is also proposed for the EVENODD code. However, X-code has some advantages over the RDP and EVENODD codes (see below). Authors of [29] experimentally evaluate the online recovery performance of single disk failure of RAID-6 codes, but they do not present any new recovery algorithm. Authors of [30] propose an efficient recovery scheme called the Path Directed Recovery Scheme (PDRS), which can decrease the disk I/O complexity by up to 25% for all vertical RAID-6 codes like P-code and X-code when recovering a single failed disk. However, they do not formally derive the lower bound of disk reads. Also, PDRS cannot consider the load balancing problem among different disks during recovery. Its performance decreases greatly as the size of storage system increases as their experiments indicated.

Authors of [31] propose an enumeration approach to solve the optimal recovery problem for MDS array codes that tolerate a general number of failures. They also show that the problem is NP-hard. Authors of [32] propose efficient recovery heuristics for general MDS array codes, and authors of [33] propose recovery heuristics for RDP and EVENODD codes in a heterogeneous environment. In this work, we focus on theoretically studying the optimal recovery problem of X-code, which is a RAID-6 array code.

Apart from optimizing the recovery performance based on encoding/decoding-related schemes, other studies propose different techniques on the failure recovery of storage systems, such as exploiting filesystem semantics [34], optimizing reconstruction sequence [35], outsourcing users' workloads during recovery [36], and exploring better cache utilization [37], [38], etc.

### 1.2 Contributions

In this paper, we consider the recovery problem of a single-disk failure for parallel storage systems using a well-known MDS array code called X-code [12], which can tolerate double-disk failures. It has been proven that X-code is *optimal* in computational complexity, update complexity, and space efficiency among all the RAID-6 codes (note that B-code [39] is also shown to be update optimal). Unlike RDP and EVENODD codes, both of which are horizontal codes, X-code is a vertical code that has a *different geometrical structure* where parities are placed in rows rather than columns. Thus, X-code has an advantage of achieving load balancing for data updates within a stripe among different disks, instead of aggregating parities in dedicated parity disks as in RDP and EVENODD codes. Due to the different geometrical structure, the recovery algorithms previously proposed for RDP and EVENODD codes are no longer applicable for X-code.

We observe that during the recovery process, the accessed data from different disks will be transmitted to a disk to generate the failed data. Thus, the transmission overhead is determined by the amount of disk reads. This poses the following open questions: In X-code-based parallel storage systems, is there a way to reduce the amount of data transmission for recovery (or the number of disk reads for recovery)? What is the lower bound of disk reads for recovering a single-disk failure in X-code? According to the specific row-based parity structure of X-code, can we design a recovery algorithm that matches this lower bound? Can the proposed recovery algorithm maintain a load balancing property, such that each disk is issued the same number of reads during recovery? Such questions motivate us to fill the void of achieving optimal recovery performance for an optimal RAID-6 code such as X-code. Thus, *the motivation of this work is to minimize the data transmission in single-disk*

*failure recovery and hence optimize the recovery performance of X-code*. Existing optimal recovery solutions for RAID-6 are designed for the horizontal codes. To the best of our knowledge, this is the first work that addresses the optimal recovery problem for a vertical code in the RAID-6 family.

We show that the amount of disk reads for single-disk failure recovery of X-code can be reduced by carefully designing the recovery approaches for the failed disk/node. The main contributions of this paper are:

1) We formally derive the tight lower bound of disk reads and data transmission for a single-disk failure recovery of X-code-based parallel storage systems and propose a recovery algorithm called *Minimum-Disk-Read-Recovery (MDRR)*, which matches the theoretical lower bound and reduces about 25% of disk reads compared with the conventional approach.
2) We formally prove that in general, disk read cannot be balanced while matching the lower bound of disk reads within a stripe, and cannot be balanced among different disks by simply rotating disks.
3) We propose a *leap rotation* scheme which balances disk reads among different disks within a groups of $p-1$ stripes, where $p$ is the number of disks in a storage system, while matching the lower bound of disk reads. We call this group-based scheme *GMDRR*.
4) To evaluate our proposed recovery schemes, we conduct extensive testbed experiments on a networked storage system called NCFS [40]. Instead of using a disk simulator as in [28], our testbed experiments capture the behavior of actual read/write operations on real storage devices during failure recovery. Our experimental results conform to our theoretical findings. For example, when NCFS is used with heterogeneous types of storage nodes, MDRR and GMDRR reduce the recovery time of the conventional recovery approach by 18.0% and up to 22.0%, respectively.

The paper proceeds as follows. Section 2 provides an overview of X-code and the conventional recovery scheme of single-disk failure. Section 3 presents a new hybrid recovery scheme which can reduce the amount of data needed for recovery. In Section 4, we theoretically prove the lower bound for the number of disk reads to recover data in a failed disk. Section 5 presents a recovery algorithm which is read-optimal and load balanced. Section 6 presents experimental results and Section 7 concludes the paper.

## 2 Background of X-code

### 2.1 How X-code Works?

We consider a storage system that consists of $p$ disks (or storage nodes)[1], where $p$ is a prime number greater than or equal to 5. X-code [12] takes a $p \times p$ two-dimensional array. The first $p-2$ rows in the array store information symbols[2] and the last two rows store the coded symbols, which are often termed as *parity symbols*, generated from information symbols. A parity symbol in row $p-2$ (row $p-1$) is generated by XOR-summing all the information symbols along the same diagonal of slope $-1$ (slope 1). Fig. 1 shows how X-code stores symbols for $p = 5$, i.e., we have $d_{3,0} = d_{0,2} \oplus d_{1,3} \oplus d_{2,4}$ and $d_{4,0} = d_{0,3} \oplus d_{1,2} \oplus d_{2,1}$.

X-code was originally proposed to tolerate two disk failures in disk array systems. Each disk in the system is divided into *strips* of fixed size and each strip is divided into $p$ segments (denoted as symbols). A *stripe* consists of $p$ strips, one in each disk. X-code is implemented within a stripe. Fig. 1 is an example, where each column is a strip in a disk and a symbol $d_{i,j}$ is the $i$-th segment in disk $D_j$. Note that in X-code, each parity symbol is only a function of information symbols and does not depend on other parity symbols. Thus, each update of an information symbol affects only the minimum number of two parity symbols, thus achieving optimal encoding complexity.

### 2.2 Single-Disk Failure Recovery

X-code aims to tolerate failures of *any* two disks. However, in practice, single-disk failure occurs much more frequently than simultaneously having two disks failed. This motivates us to design efficient recovery schemes for single-disk failure in X-code-based storage systems. In [12], only the recovery for a double-disk failure is considered, but no recovery algorithm is considered specifically for a single-disk failure. A conventional recovery approach for single-disk failure is: (1) the parity symbol of slope $-1$ in row $p-2$ can only be recovered by XOR-summing all symbols along the same diagonal of slope $-1$; (2) other symbols are recovered by XOR-summing all symbols along the same diagonal of slope 1.

Fig. 2 shows an example of the conventional recovery approach when disk $D_0$ fails in an X-code-based system with 7 disks. In Fig. 2, the symbols marked "$\times$" are erased symbols. The parity symbol $d_{5,0}$ marked "$\square$" is recovered by the parity of slope $-1$, while other erased symbols marked "$\bigcirc$" are recovered by the parities of slope 1. A surviving symbol marked "$\square$" (or/and "$\bigcirc$") is used for the recovery as a symbol in a diagonal of slope $-1$ (or/and slope 1).

If we use the conventional approach to recover a single failed disk $D_k$, the total number of disk reads is calculated as: (1) The recovery of the parity symbol $d_{p-2,k}$ reads $p-2$ symbols along a diagonal of slope $-1$; (2) The recovery of each of the other symbols in $D_k$ reads $p-2$ symbols along a diagonal of slope 1, totally $(p-2) \times (p-1)$ symbols. Thus, the conventional approach

1. We use the terms "disks" and "nodes" interchangeably in this paper.

2. We use the term "symbols" to represent device blocks or data chunks. A symbol can also correspond to a set of consecutive sectors of a disk.

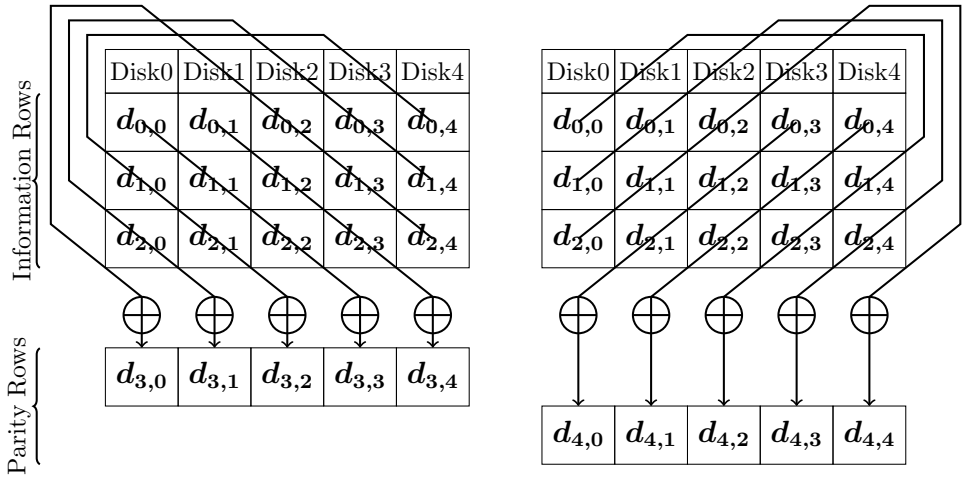



Fig. 1. An example of the encoding strategy of X-code when $p = 5$.

issues a total of $p-2+(p-2)(p-1) = p(p-2)$ disk reads (per stripe).

However, we note that there are $p-3$ *overlapping* symbols among the $p-2$ symbols read along the diagonal of slope $-1$ and the $(p-2)\times(p-1)$ symbols along the diagonals of slope 1. So the number of different symbols read from all disks can be reduced. In Fig. 2(a), $(7-2)\times(7-1)+1 = 31$ different symbols are read from disks. There are 4 *overlapping* symbols marked both "□" and "◯", which are the common symbols read along the diagonal of slope $-1$ and along the diagonals of slope 1. In the conventional approach, these 4 symbols are read *twice* from disks, each for the recovery of two erased symbols. The issue is that the conventional approach seeks to recover each symbol individually, without considering other erased symbols in the same stripe being also recovered.

Typically memory read is significantly faster than disk read. If we store a symbol marked both "□" and "◯" in memory after it is read from disk for the recovery of an erased symbol, it can be read from memory for the recovery of another erased symbol. Thus, we can reduce the number of symbols to be read directly from disk so as to speed up recovery.

Apart from the two parity symbols, each of the other erased symbols can be recovered from a parity of either slope 1 or slope $-1$. Thus, there are *many choices* to recover a single failed disk. Fig. 2(b) shows another recovery choice for the case in Fig. 2(a). In Fig. 2(b), there are 9 overlapping symbols marked both "□" and "◯", so only 26 different symbols are read from disks, which is less than 31 in Fig. 2(a).

The motivation of this work is to use *both* parities of slope 1 and slope $-1$ for single-disk failure recovery so as to *maximize the number of overlapping symbols of parities of slope* $-1$ *and slope* 1. This allows us to maximize the number of symbols read from memory and minimize the number of symbols to be read directly from disks for recovery.

## 3 Hybrid Recovery of Single Disk Failure

In this section, we formally define the different choices for single-disk failure recovery. First, we give some definitions.

**Definition** *1:* We define the following:

- For $0 \le i \le p-1$, define the $i$-th parity set of slope 1 as $L_i = \{d_{m,n} | \langle m+n\rangle_p = \langle i-2\rangle_p, 0 \le m \le p-3$ and $0 \le n \le p-1\} \cup \{d_{p-1,i}\}$. (*Note:* $\langle x\rangle_p = x \mod p$.)
- For $0 \le j \le p-1$, define the $j$-th parity set of slope $-1$ as $R_j = \{d_{m,n} | \langle m-n\rangle_p = \langle p-2-j\rangle_p, 0 \le m \le p-2$ and $0 \le n \le p-1\}$.

$L_i$ consists of parity symbol $d_{p-1,i}$ and all symbols along the $i$-th diagonal of slope 1 which are used to generate $d_{p-1,i}$. $R_j$ consists of parity symbol $d_{p-2,j}$ and all symbols along the $j$-th diagonal of slope $-1$ which are used to generate $d_{p-2,j}$. One can refer to Fig. 1 to understand $L_i$ and $R_j$. From the encoding of X-code, the following lemma states how an erased symbol can be reconstructed.

**Lemma** *1:* Given an erased symbol $d$,

1) If $d \in L_i$, it can be reconstructed by XOR-summing all the symbols in $L_i - \{d\}$.
2) If $d \in R_j$, it can be reconstructed by XOR-summing all the symbols in $R_j - \{d\}$.
3) $d$ can only be recovered by 1) or 2).

For the ease of discussion, we always let $D_k$ be the failed disk, i.e., all symbols in column $k$ are lost and need to be recovered. The following lemma shows the possible recovery choices for each erased symbol in $D_k$.

**Lemma** *2:* Given a failed disk $D_k$,

1) Symbol $d_{i,k}$ $(0 \le i \le p-3)$ can be recovered either by XOR-summing all symbols in $L_{\langle i+k+2\rangle_p} - \{d_{i,k}\}$ or by XOR-summing all symbols in $R_{\langle k-i-2\rangle_p} - \{d_{i,k}\}$.
2) Symbol $d_{p-2,k}$ can only be recovered by XOR-summing all symbols in $R_k - \{d_{p-2,k}\}$.
3) Symbol $d_{p-1,k}$ can only be recovered by XOR-summing all symbols in $L_k - \{d_{p-1,k}\}$.

| Disk0 | Disk1 | Disk2 | Disk3 | Disk4 | Disk5 | Disk6 |
|---|---|---|---|---|---|---|
| $d_{0,0}$ | $d_{0,1}$ | $d_{0,2}$ | $d_{0,3}$ | $d_{0,4}$ | $d_{0,5}$ | $d_{0,6}$ |
| $d_{1,0}$ | $d_{1,1}$ | $d_{1,2}$ | $d_{1,3}$ | $d_{1,4}$ | $d_{1,5}$ | $d_{1,6}$ |
| $d_{2,0}$ | $d_{2,1}$ | $d_{2,2}$ | $d_{2,3}$ | $d_{2,4}$ | $d_{2,5}$ | $d_{2,6}$ |
| $d_{3,0}$ | $d_{3,1}$ | $d_{3,2}$ | $d_{3,3}$ | $d_{3,4}$ | $d_{3,5}$ | $d_{3,6}$ |
| $d_{4,0}$ | $d_{4,1}$ | $d_{4,2}$ | $d_{4,3}$ | $d_{4,4}$ | $d_{4,5}$ | $d_{4,6}$ |
| $d_{5,0}$ | $d_{5,1}$ | $d_{5,2}$ | $d_{5,3}$ | $d_{5,4}$ | $d_{5,5}$ | $d_{5,6}$ |
| $d_{6,0}$ | $d_{6,1}$ | $d_{6,2}$ | $d_{6,3}$ | $d_{6,4}$ | $d_{6,5}$ | $d_{6,6}$ |

(a) Conventional recovery

| Disk0 | Disk1 | Disk2 | Disk3 | Disk4 | Disk5 | Disk6 |
|---|---|---|---|---|---|---|
| $d_{0,0}$ | $d_{0,1}$ | $d_{0,2}$ | $d_{0,3}$ | $d_{0,4}$ | $d_{0,5}$ | $d_{0,6}$ |
| $d_{1,0}$ | $d_{1,1}$ | $d_{1,2}$ | $d_{1,3}$ | $d_{1,4}$ | $d_{1,5}$ | $d_{1,6}$ |
| $d_{2,0}$ | $d_{2,1}$ | $d_{2,2}$ | $d_{2,3}$ | $d_{2,4}$ | $d_{2,5}$ | $d_{2,6}$ |
| $d_{3,0}$ | $d_{3,1}$ | $d_{3,2}$ | $d_{3,3}$ | $d_{3,4}$ | $d_{3,5}$ | $d_{3,6}$ |
| $d_{4,0}$ | $d_{4,1}$ | $d_{4,2}$ | $d_{4,3}$ | $d_{4,4}$ | $d_{4,5}$ | $d_{4,6}$ |
| $d_{5,0}$ | $d_{5,1}$ | $d_{5,2}$ | $d_{5,3}$ | $d_{5,4}$ | $d_{5,5}$ | $d_{5,6}$ |
| $d_{6,0}$ | $d_{6,1}$ | $d_{6,2}$ | $d_{6,3}$ | $d_{6,4}$ | $d_{6,5}$ | $d_{6,6}$ |

(b) Hybrid recovery

Fig. 2. Two recovery approaches for $p = 7$.

| | Disk0 | Disk1 | Disk2 | Disk3 | Disk4 | Disk5 | Disk6 |
|---|---|---|---|---|---|---|---|
| $R_5$ | $d_{0,0}$ | $d_{0,1}$ | $d_{0,2}$ | $d_{0,3}$ | $d_{0,4}$ | $d_{0,5}$ | $d_{0,6}$ |
| $L_3$ | $d_{1,0}$ | $d_{1,1}$ | $d_{1,2}$ | $d_{1,3}$ | $d_{1,4}$ | $d_{1,5}$ | $d_{1,6}$ |
| $L_4$ | $d_{2,0}$ | $d_{2,1}$ | $d_{2,2}$ | $d_{2,3}$ | $d_{2,4}$ | $d_{2,5}$ | $d_{2,6}$ |
| $R_2$ | $d_{3,0}$ | $d_{3,1}$ | $d_{3,2}$ | $d_{3,3}$ | $d_{3,4}$ | $d_{3,5}$ | $d_{3,6}$ |
| $L_6$ | $d_{4,0}$ | $d_{4,1}$ | $d_{4,2}$ | $d_{4,3}$ | $d_{4,4}$ | $d_{4,5}$ | $d_{4,6}$ |
| $R_0$ | $d_{5,0}$ | $d_{5,1}$ | $d_{5,2}$ | $d_{5,3}$ | $d_{5,4}$ | $d_{5,5}$ | $d_{5,6}$ |
| $L_0$ | $d_{6,0}$ | $d_{6,1}$ | $d_{6,2}$ | $d_{6,3}$ | $d_{6,4}$ | $d_{6,5}$ | $d_{6,6}$ |

Fig. 3. Recovery sequence for the failed Disk 0 of $p = 7$.

**Proof:** According to the encoding scheme of X-code, an information symbol $d_{i,k}(0 \le i \le p-3)$ belongs to $L_{\langle i+k+2\rangle_p}$ and $R_{\langle k-i-2\rangle_p}$. Thus, $d_{i,k}$ can be recovered by either XOR-summing all symbols in $L_{\langle i+k+2\rangle_p} - \{d_{i,k}\}$ or XOR-summing all symbols in $R_{\langle k-i-2\rangle_p} - \{d_{i,k}\}$ according to Lemma 1. However, the parity symbol $d_{p-2,k}$ only belongs to $R_k$, it can only be recovered by XOR-summing all symbols in $R_k - \{d_{p-2,k}\}$. Similarly, 3) of Lemma 2 holds. ∎

In the following, we use a *recovery sequence* of $L_i$ and $R_j$ of length $p$ to represent a possible recovery choice. For example, suppose that a disk array of seven disks with disk $D_0$ being failed. $R_5L_3L_4R_2L_6R_0L_0$ is a recovery choice which recovers symbols $d_{1,0}$, $d_{2,0}$, $d_{4,0}$ and $d_{6,0}$ with parities of slope 1 and recovers $d_{0,0}$, $d_{3,0}$ and $d_{5,0}$ with parities of slope $-1$, as illustrated in Fig. 3. Since $d_{p-2,0}$ and $d_{p-1,0}$ are parity symbols of slope $-1$ and slope 1, respectively, the last two of a recovery sequence must be $R_0$ and $L_0$, respectively. Our goal is to find *a recovery sequence with maximum number of overlapping symbols in the parity sets of slope $-1$ and in the parity sets of slope 1 to reduce the symbols read from disks.*

## 4 A Lower Bound of Disk Reads

We now derive a lower bound of the number of disk reads and propose a recovery algorithm which matches this lower bound. We first give a necessary and sufficient condition for two parity sets to share an overlapping symbol.

**Lemma** *3:* We have

1) For any $i, j$, $0 \le i, j \le p-1, i \ne j$, $|L_i \bigcap L_j| = 0$ and $|R_i \bigcap R_j| = 0$.
2) For any $i, j$, $0 \le i, j \le p-1$, if $\langle i-j\rangle_p = 0$ or 2, $|L_i \bigcap R_j| = 0$; otherwise $|L_i \bigcap R_j| = 1$.

**Proof:** Because any two diagonals along slope 1 (or slope $-1$) are parallel, there is no overlapping symbol in a pair of parity sets of slope 1 (or slope $-1$). So 1) of Lemma 3 concludes.

Now we prove 2) of Lemma 3. Let $d_{m,n}$ be an overlapping symbol of $L_i$ and $R_j$, i.e., $d_{m,n} \in L_i \bigcap R_j$. From the definitions of $L_i$ and $R_j$, $\langle m+n\rangle_p = \langle i-2\rangle_p$ and $\langle m-n\rangle_p = \langle p-2-j\rangle_p$. So

$$\langle 2m\rangle_p = \langle p-4+i-j\rangle_p. \tag{1}$$

From Definition 1, a parity symbol only belongs to one parity set. So an overlapping symbol must be an information symbol. From $d_{m,n} \in L_i \bigcap R_j$, $0 \le m \le p-3$ holds. Notice that $p$ is a prime number greater than or equal to 5, therefore

1) If $\langle i-j\rangle_p = 0$, $\langle p-4+i-j\rangle_p = \langle p-4\rangle_p$. Since $p$ is greater than or equal to 5, $\langle p-4\rangle_p = p-4$. If $2m < p$, $\langle 2m\rangle_p = 2m$ is even and Equation (1) does not hold because $p-4$ is odd. Otherwise $2m > p$, and $\langle 2m\rangle_p = 2m-p$. If Equation (1) holds, $2m-p = p-4$ and $m = p-2$. It contradicts to $0 \le m \le p-3$. So Equation (1) does not hold, and $L_i \bigcap R_j = \emptyset$. $|L_i \bigcap R_j| = 0$.
2) If $\langle i-j\rangle_p = 2$, $\langle p-4+i-j\rangle_p = \langle p-2\rangle_p$. Similar to 1), $|L_i \bigcap R_j| = 0$.
3) If $\langle i-j\rangle_p = 1$ or 3, $\langle p-4+i-j\rangle_p = p-4+\langle i-j\rangle_p$ is even. Let $m = (p-4+\langle i-j\rangle_p)/2$ and $n = (i+j)/2$. $d_{m,n}$ is the only symbol in $L_i \bigcap R_j$. $|L_i \bigcap R_j| = 1$.

4) If $\langle i-j\rangle_p$ is no smaller than 4, $\langle p-4+i-j\rangle_p = \langle i-j\rangle_p - 4$. If $\langle i-j\rangle_p - 4$ is even, let $m = (\langle i-j\rangle_p - 4)/2$; otherwise let $m = (\langle i-j\rangle_p + p - 4)/2$. Let $n = (i+j)/2$. $d_{m,n}$ is the only symbol in $L_i \bigcap R_j$. $|L_i \bigcap R_j| = 1$.

So Lemma 3 concludes. ∎

Suppose that $\mathcal{RS}$=$S_0S_1...S_{p-1}$ is a recovery sequence for $D_k$, where $S_i$ is to recover symbol $d_{i,k}$. From Definition 1, if $d_{i,k}$ is recovered by a parity set of slope 1, $S_i = L_{\langle i+k+2\rangle_p}$; otherwise, $d_{i,k}$ is recovered by a parity set of slope $-1$, $S_i = R_{\langle k-i-2\rangle_p}$. Since $d_{p-2,k}$ and $d_{p-1,k}$ are parity symbols of slope $-1$ and slope 1 respectively, $S_{p-2}$ is $R_k$ and $S_{p-1}$ is $L_k$. From Lemma 3, we can conclude that:

1) $|L_k \bigcap R_k| = 0$; i.e. there is no overlapping symbol of parity set $R_k$ for recovering $d_{p-2,k}$ and parity set $L_k$ for recovering $d_{p-1,k}$,
2) For $1 \le i \le p-3$, whether $S_i = L_{\langle i+k+2\rangle_p}$ or $S_i = R_{\langle k-i-2\rangle_p}$, $|(L_k \bigcup R_k) \bigcap S_i| = 1$,
3) Whether $S_0 = L_{\langle k+2\rangle_p}$ or $S_0 = R_{\langle k-2\rangle_p}$, $|(L_k \bigcup R_k) \bigcap S_0| = 0$.

Apart from $d_{p-2,k}$ and $d_{p-1,k}$ which must be recovered by $R_k$ and $L_k$ respectively, there are still $p-2$ symbols $d_{0,k}$, $d_{1,k}$, ..., $d_{p-3,k}$ in $D_k$ to be recovered. The above three conclusions show that there are $p-3$ overlapping symbols in $L_k \bigcup R_k$ and the union of parity sets for the recovery of $d_{0,k}$, $d_{1,k}$, ..., $d_{p-3,k}$. The following lemma formally states the number of overlapping symbols in the recovery.

**Lemma** *4:* If a recovery sequence $\mathcal{RS}$ recovers $x$ symbols of $d_{0,k}, d_{1,k}, ..., d_{p-3,k}$ with parity sets of slope 1 and other $p-2-x$ symbols with parity sets of slope $-1$, $\mathcal{RS}$ has

$$N(x,O) = (p-2-x) \times x + (p-3) - |O|, \qquad (2)$$

overlapping symbols, where $O = \{(m,n)|0 \le m < n \le p-3$, $S_m$ and $S_n$ are parity sets, but not both of slope 1 (or slope $-1$), and share no overlapping symbol$\}$.

**Proof:** In Equation (2), $p-3$ is the number of overlapping symbols of $L_k \bigcup R_k$ and the union of other parity sets of $\mathcal{RS}$. If each pair of a parity set of slope 1 and a parity set of slope $-1$ in $\mathcal{RS}$ (excluding $R_k$ and $L_k$) has an overlapping symbol, there are totally $(p-2-x) \times x$ of these overlapping symbols. So Lemma 4 concludes. ∎

In Equation (2), $N(x,O) = \frac{p^2-9}{4}$ is maximized when $x = \frac{p-1}{2}$ or $\frac{p-3}{2}$, and $|O| = 0$. In the following, we prove that $N(x,O) = \frac{p^2-9}{4}$ cannot be matched, and show that some recovery sequences match $N(x,O) = \frac{p^2-9}{4} - 1$. We firstly introduce Lemma 5.

**Lemma** *5:* Define $\mathcal{A}$: $a[0]$, $a[1]$, ..., $a[p-3]$ with

$$\begin{cases} a[i] = i, \ i \text{ is even}, \\ a[i] = p-3-i, \ i \text{ is odd}. \end{cases} \qquad (3)$$

Then $\mathcal{A}$ satisfies that

1) $\{a[i]|0 \le i \le p-3\}$=$\{i|0 \le i \le p-3\}$;
2) Given a pair of parity sets, $S_m$ and $S_n$ ($m \ne n$), suppose that one of them is of slope 1 and the other is of slope $-1$. Then $S_m \bigcap S_n = \emptyset$ if and only if there is $i$, $0 \le i \le p-3$, such that $a[i] = m$, and $a[i+1] = n$ or $a[i-1] = n$.

**Proof:** According to the definition of $\mathcal{A}$, if $i$ is odd, $a[i]$ is odd; otherwise $a[i]$ is even. So 1) of Lemma 5 concludes from $a[i] \ne a[j]$ for $i \ne j$ and $0 \le a[i] \le p-3$.

Now we prove 2) of Lemma 5. Suppose that $S_m \bigcap S_n = \emptyset$. Without loss of generality, suppose that $S_m$ is of slope 1 and $S_n$ is of slope $-1$, i.e. $S_m = L_{\langle k+m+2\rangle_p}$ and $S_n = R_{\langle k-n-2\rangle_p}$. Since $S_m \bigcap S_n = \emptyset$, i.e., $L_{\langle k+m+2\rangle_p} \bigcap R_{\langle k-n-2\rangle_p} = \emptyset$, we can conclude from Lemma 3 that

$$\langle (k+m+2) - (k-n-2)\rangle_p = \langle m+n+4\rangle_p = 0 \text{ or } 2. \qquad (4)$$

Because $0 \le m, n \le p-3$, $4 \le m+n+4 \le 2p-2$. So $m+n+4 = p$ or $m+n+4 = p+2$ from Equation (4), therefore $n = p-2-m$ or $n = p-4-m$.

Because $p$ is an odd number, one of $m$ and $n$ is odd number also and another is even. If $m$ is odd and $n$ is even, we know that $a[p-3-m] = m$, $a[n] = n$ from the definition of $\mathcal{A}$. Let $i = p-3-m$. If $n = p-2-m$, $n = i+1$; otherwise $n = p-4-m$ and $n = i-1$. So there is $i$, $0 \le i \le p-3$, such that $a[i] = m$, and $a[i+1] = n$ or $a[i-1] = n$. The proof of case of $n$ being odd and $m$ being even is similar.

If there is $i$, $0 \le i \le p-3$, such that $a[i] = m$, and $a[i+1] = n$ or $a[i-1] = n$ in $\mathcal{A}$. Without loss of generality, suppose that $m$ is odd and $n$ is even, then $a[p-3-m] = m$, $a[n] = n$ from the definition of $\mathcal{A}$. So $i = p-3-m$, and $n = i+1 = p-2-m$ or $n = i-1 = p-4-m$. We have

1) $n = p-2-m$: then $\langle m+n+4\rangle_p = 2$. Note that one of $S_m$ and $S_n$ is a parity set of slope 1, and the other is of slope $-1$. If $S_m$ is of slope 1 and $S_n$ is of slope $-1$, i.e. $S_m = L_{\langle k+m+2\rangle_p}$ and $S_n = R_{\langle k-n-2\rangle_p}$, $\langle (k+m+2) - (k-n-2)\rangle_p = 2$; otherwise, $S_m$ is of slope $-1$ and $S_n$ is of slope 1, i.e. $S_m = R_{\langle k-m-2\rangle_p}$ and $S_n = L_{\langle k+n+2\rangle_p}$, $\langle (k+n+2) - (k-m-2)\rangle_p = 2$. From Lemma 3, $S_m \bigcap S_n = \emptyset$.
2) $n = p-4-m$: then $\langle m+n+4\rangle_p = 0$, the proof is similar to case of $n = p-2-m$.

From above, 2) of Lemma 5 concludes. ∎

We are now ready to give a lower bound of disk reads so as to recover a single failed disk.

**Theorem** *1:* We have

1) A lower bound of disk reads for the recovery of single disk failure is $\frac{3p^2-8p+13}{4}$.
2) There are four recovery sequences which match the lower bound of disk reads to recover a single failed disk.

**Proof:** Since it needs $p-2$ symbols to recover each of the erased symbols, the number of disk reads is $(p-2) \times p - N(x,O)$. Using the sequence $\mathcal{A}$ defined in Lemma 5, we analyze the upper bound of $N(x,O)$ in Lemma 4.

If $|O| = 0$, $S_{a[i]} \cap S_{a[i+1]} \neq \emptyset$ for $0 \leq i \leq p-4$ from the definition of $O$ and the definition of $\mathcal{A}$ in Lemma 5. From 2) of Lemma 5, both of $S_{a[i]}$ and $S_{a[i+1]}$ are parity sets of slope 1 (or slope $-1$). So all of $S_{a[0]}$, $S_{a[1]}$, ..., $S_{a[p-3]}$ are parity sets of slope 1 (or all of slope $-1$). From 1) of Lemma 5, all of $S_0$, $S_1$, ..., $S_{p-3}$ must be parity sets of slope 1 (or all of slope $-1$), which means that all information symbols must be recovered by parity sets of slope 1 or all by parity sets of slope $-1$, i.e. $x = p-2$ or 0 in Equation (2). Therefore $N(x, O) < \frac{p^2-9}{4}$.

We now show that some recovery sequences match $N(x, O) = \frac{p^2-9}{4} - 1$ overlapping symbols, which maximize $N(x, O)$.

If $|O| = 1$, from the proof of case $|O| = 0$, there is only one $l$ such that one of $S_{a[l]}$ and $S_{a[l+1]}$ is a parity set of slope 1, and the other is of slope $-1$. To satisfy the condition that $x = (p-1)/2$ or $(p-3)/2$ (so as to maximize $N(x, O)$) in Equation (2), $l$ must be $\frac{p-5}{2}$ or $\frac{p-3}{2}$. So we can divide $\mathcal{A}$ into two successive subsequences $a[0]$, ..., $a[l]$ and $a[l+1]$, ..., $a[p-3]$, such that all of $S_{a[0]}$, ..., $S_{a[l]}$ are parity sets of slope 1 (or all of slope $-1$), while all of $S_{a[l+1]}$, ..., $S_{a[p-3]}$ are parity sets of slope $-1$ (or all of slope 1) correspondingly. There are only *four recovery sequences* which have $\frac{p^2-9}{4} - 1 = \frac{p^2-13}{4}$ overlapping symbols. They are:

1) $\mathcal{RS}_1$: $S_{a[i]} = L_{<a[i]+k+2>_p}$, $0 \leq i \leq \frac{p-3}{2}$; $S_{a[i]} = R_{<k-a[i]-2>_p}$, $\frac{p-1}{2} \leq i \leq p-3$;
2) $\mathcal{RS}_2$: $S_{a[i]} = R_{<k-a[i]-2>_p}$, $0 \leq i \leq \frac{p-3}{2}$; $S_{a[i]} = L_{<a[i]+k+2>_p}$, $\frac{p-1}{2} \leq i \leq p-3$;
3) $\mathcal{RS}_3$: $S_{a[i]} = L_{<a[i]+k+2>_p}$, $0 \leq i \leq \frac{p-5}{2}$; $S_{a[i]} = R_{<k-a[i]-2>_p}$, $\frac{p-3}{2} \leq i \leq p-3$;
4) $\mathcal{RS}_4$: $S_{a[i]} = R_{<k-a[i]-2>_p}$, $0 \leq i \leq \frac{p-5}{2}$; $S_{a[i]} = L_{<a[i]+k+2>_p}$, $\frac{p-3}{2} \leq i \leq p-3$;

$\mathcal{RS}_i (1 \leq i \leq 4)$ maximize the number of overlapping symbols, and minimize disk read with $(p-2)\times p - N(x, O) = (p-2) \times p - \frac{p^2-13}{4} = \frac{3p^2-8p+13}{4}$ symbols. ∎

Authors of [19] derive the information theoretic lower bound of the amount of data needed for recovery based on a linear network code defined over a sufficiently large finite field. This lower bound approaches to a factor of 50% for double-fault tolerant codes as the size of the storage system increases. Authors of [41], [42] design new codes which match the information theoretic lower bound. However, the size of the field to realize the codes must be no less than 3, which implies that the codes cannot be implemented only with XOR operations and their computational complexities will be higher than those of the popular XOR-based RAID-6 codes. Authors of [42] also prove that to match the information theoretic lower bound of repair bandwidth, the size of the field to realize the code must be no less than 3. Authors of [43] design a vector code with a repair bandwidth of 50% of the survived data amount, where repair bandwidth is the data amount transmitted in the storage system for the repair, not the data amount read from the disk. The repair scheme in [43] reads all data from each surviving node in the system, encodes the data at each node to an encoded one with 50% of size and then transmits the encoded one to the back-up node.

In an X-code-based storage system, any failed symbol can only be recovered by XOR-summing all other symbols in its parity of slope 1 (or slope $-1$). Given a failed disk $D_k$, suppose that $d_{i,k}$ and $d_{j,k}$ $(i \neq j)$ are two lost symbols in disk $D_k$. Because the diagonals of slope $-1$ which $d_{i,k}$ and $d_{j,k}$ lie at are parallel (the diagonals of slope 1 are also parallel), $d_{i,k}$ does not lie at the two diagonals of slope 1 and slope $-1$ which $d_{j,k}$ lies at, which means that $d_{i,k}$ can not be used to recover $d_{j,k}$. So any recovered symbol can not be used to recover other lost symbols. Therefore the above four recovery sequences are with the minimal disk reads. Theorem 1 is tight for X-code.

In the following, we select $\mathcal{RS}_4$ as an example to present our recovery algorithm, MDRR (Minimum Disk Read Recovery), for single-disk failure which minimizes disk read. $\mathcal{RS}_4$ reduces $\frac{p^2-13}{4}$ symbols read from disks. Compared with $p(p-2)$ disk reads of the conventional recovery algorithm, MDRR reduces approximately 1/4 of disk reads. Fig. 4 gives a comparison of the number of disk reads between the conventional recovery and MDRR with different $p$.

**Algorithm 1** Minimum Disk Read Recovery (MDRR) of Single Disk Failure for X-code

Suppose disk $D_k$ is failed.

1) For $0 \leq i \leq \frac{p-5}{2}$, if $i$ is odd, recover $d_{i,k}$ by XOR-summing all symbols in $L_{\langle k+i+2\rangle_p} - \{d_{i,k}\}$; otherwise, recover $d_{i,k}$ by XOR-summing all symbols in $R_{\langle k-i-2\rangle_p} - \{d_{i,k}\}$.
2) For $\frac{p-3}{2} \leq i \leq p-1$, if $i$ is even, recover $d_{i,k}$ by XOR-summing all symbols in $L_{\langle k+i+2\rangle_p} - \{d_{i,k}\}$; otherwise, recover $d_{i,k}$ by XOR-summing all symbols in $R_{\langle k-i-2\rangle_p} - \{d_{i,k}\}$.

## 5 Disk Read Balancing

In the previous section, we derived a lower bound of disk reads and presented the MDRR algorithm to match the lower bound. But MDRR does not possess the disk read balancing property, i.e., it reads *different number* of symbols from different disks. In case of unbalanced disk read, a disk with a heavier load will slow down the recovery and degrade the availability of the system. Here, we prove that *disk read cannot be balanced in a stripe while matching the lower bound in general cases*. Furthermore, it cannot be balanced by disk rotation. We then present a method which balances disk read in a group of $p-1$ stripes while matching the lower bound.

**Theorem** *2:* If a single disk failure recovery algorithm matches the lower bound of disk reads, then it cannot balance disk read from different disks for the recovery within a stripe in any cases except $p = 7$.

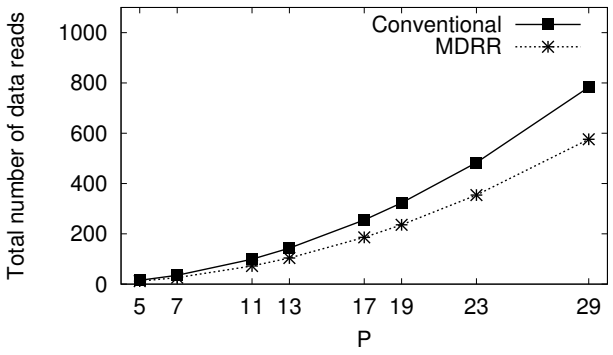



Fig. 4. Numerical comparison of number of disk reads between conventional and hybrid recovery approaches.

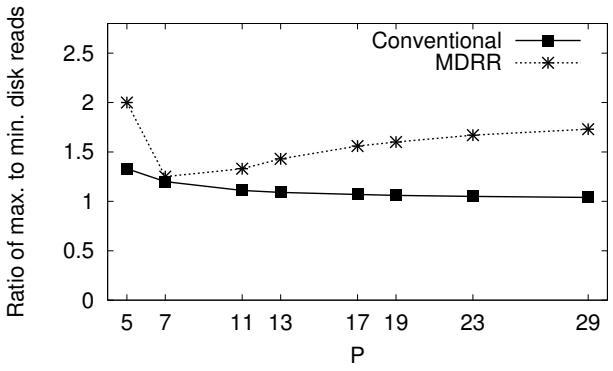



Fig. 5. Numerical comparison of disk reads from different disks.

**Proof:** Only four recovery sequences $\mathcal{RS}_i$ $(1 \leq i \leq 4)$ read minimum number of symbols for the recovery. We prove Theorem 2 with $\mathcal{RS}_1$. The proofs of $\mathcal{RS}_2$, $\mathcal{RS}_3$, $\mathcal{RS}_4$ are similar. We say that disk read is balanced within a stripe if the numbers of disk reads of each disk differ by at most one.

First, $\mathcal{RS}_1$ matches the lower bound of disk reads, i.e., $\frac{3p^2-8p+13}{4}$. On average, it reads $\overline{r} = \frac{3p^2-8p+13}{4\times(p-1)}$ symbols from each disk. However, it reads $p-3$ symbols from disk $D_{\langle k+1\rangle_p}$. When $|(p-3)-\overline{r}| = |\frac{(p-4)^2-17}{4\times(p-1)}|$ is no more than 1, it is possible to balance disk read.

If $p > 7$, $|(p-3)-\overline{r}| > 1$, and disk read cannot be balanced within a stripe. By enumerating all four recovery sequences with minimal disk read, when $p = 5$, Algorithm 1 cannot balance disk read because the numbers of disk read on different disks differ by two; but when $p = 7$, disk read is balanced. ∎

Note that $(p-3)-\overline{r}$ increases rapidly with the increase of $p$ when $p > 7$. Unbalanced disk read becomes more serious as $p$ increases. Fig. 5 gives the ratios of the maximum to the minimum number of disk reads among different disks. From Theorem 2, we know that in general, it is impossible to balance disk reads while matching the lower bound within a stripe. One may think that disk read can be balanced by simply rotating disks in different stripes. In the following, we show that simply rotating disks cannot balance disk read and present a method which balances disk read in a group of $p-1$ stripes. We will firstly introduce the notion of *logical disk*.

**Definition** *2:* Given a disk array system with $p$ disks $D_0$, $D_1$, ..., $D_{p-1}$, which are called *physical disks*. Let $PDs = \{0, 1, 2, ..., p-1\}$ and $\mathcal{T}$:$t_0t_1...t_{p-1}$ be a permutation of $PDs$, define *the logical disk of* $D_j$ *to* $\mathcal{T}$ as $LD_{t_j} = D_j$, $0 \leq j \leq p-1$.

X-code can be understood using the notion of logical disks where $d_{i,j}$ is the symbol stored at row $i$ of logical disk $LD_j$. Fig. 6 shows an example of $p = 5$, where the logical disks correspond to a permutation, 02413. The logical disks of $D_0$, $D_1$, ..., $D_4$ are $LD_0 = D_0$, $LD_2 = D_1$, $LD_4 = D_2$, $LD_1 = D_3$, $LD_3 = D_4$, respectively. For example, $d_{2,3}$ is the symbol at row 2 of logical disk $LD_3$, which is physically stored in disk $D_4$ because $LD_3 = D_4$. Similarly, $d_{4,1}$ is physically stored in disk $D_3$ because $LD_1 = D_3$.

Rotating the physical disks once corresponds to a permutation $12...(p-1)0$ of $PDs$ with $LD_j$=$D_{\langle j-1\rangle_p}$, $0 \leq j \leq p-1$. We will show that recovery sequences $\mathcal{RS}_i$ $(1 \leq i \leq 4)$ cannot balance disk reads by simply rotating the disks.

**Lemma** *6:* X-codes implemented on the following two groups of logical disks are equivalent:

1) *Group 1*: $LD_j = D_j, j = 0, 1, ..., p-1$.
2) *Group 2*: $LD_j = D_{j-1}, j = 0, 1, ..., p-1$.

**Proof:** Group 2 is a rotation of all the physical disks of Group 1. From the definition of X-code, we have the following two equations:

$$\begin{cases} d_{p-2,j} = \sum_{k=0}^{p-3} d_{k,\langle j+k+2\rangle_p}, \\ d_{p-1,j} = \sum_{k=0}^{p-3} d_{k,\langle j-k-2\rangle_p}, \end{cases} \tag{5}$$

where $d_{i,j}$ is the symbol at row $i$ of logical disk $LD_j$. Suppose that $d^{'}_{i,j}$, $d^{''}_{i,j}$ are the symbols at row $i$ of logical disk $LD_j$ of Group 1 and Group 2 respectively. From the definition of the two groups, symbols $d^{'}_{i,\langle j-1\rangle_p}$ and $d^{''}_{i,j}(0 \leq i \leq p-3)$ store at row $i$ of the same physical disk, i.e., $d^{'}_{i,\langle j-1\rangle_p} = d^{''}_{i,j}$. So we have the following equalities from the definition of X-code.

$$\begin{cases} d^{''}_{p-2,j} &= \sum_{k=0}^{p-3} d^{''}_{k,\langle j+k+2\rangle_p} = \sum_{k=0}^{p-3} d^{'}_{k,\langle (j-1)+k+2\rangle_p} \\ &= d^{'}_{p-2,\langle j-1\rangle_p}. \\ d^{''}_{p-1,j} &= \sum_{k=0}^{p-3} d^{''}_{k,\langle j-k-2\rangle_p} = \sum_{k=0}^{p-3} d^{'}_{k,\langle (j-1)-k-2\rangle_p} \\ &= d^{'}_{p-1,\langle j-1\rangle_p}. \end{cases} \tag{6}$$

From Equation (6), two parity symbols also satisfy $d^{''}_{i,j}$ = $d^{'}_{i,\langle j-1\rangle_p}$, $i = p-2$ and $p-1$. So X-codes based on logical disks of Group 1 and Group 2 are exactly the same. ∎

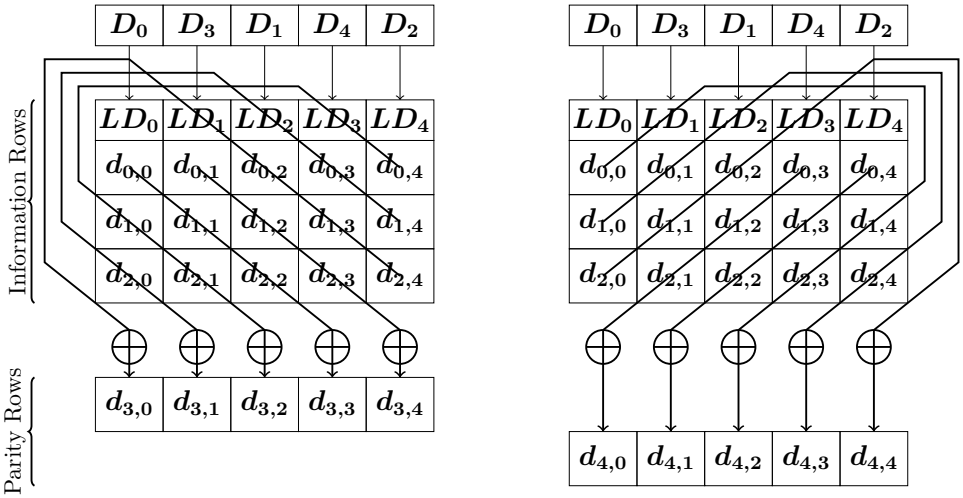



Fig. 6. An example of X-code encoding based on logical disks.

**Definition** *3:* Given two logical disks $LD_{j_1}$ and $LD_{j_2}$, the *logical distance* from $LD_{j_1}$ to $LD_{j_2}$ is defined as $LDI(j_1, j_2) = \langle j_2 - j_1 \rangle_p$.

Note that logical distance is asymmetric, i.e., it is possible that $LDI(j_1, j_2) \neq LDI(j_2, j_1)$. For example, when $p = 7$, $LDI(2, 4) = 2$, but $LDI(4, 2) = 5$.

**Theorem** *3:* If a recovery sequence $\mathcal{RS}$ for X-code system matches the lower bound of disk read, it cannot balance disk reads by simply rotating disks.

**Proof:** If $LDI(j_1, j_2) = LDI(l_1, l_2)$, there is $t$ such that $l_1 = \langle j_1 - t \rangle_p$ and $l_2 = \langle j_2 - t \rangle_p$ from Definition 3. From Lemma 6, we know that the number of symbols read from $LD_{l_2}$ when $LD_{l_1}$ fails equals to the number of symbols read from $LD_{j_2}$ when $LD_{j_1}$ fails by $t$ times of rotation. This implies that the number of symbols to be read in a stripe from physical disk $D_j$ when physical disk $D_k$ fails depends on the logical distance from the logical disk of $D_j$ to the logical disk of $D_k$ in the stripe. Because rotation does not change the logical distance from a physical disk to the failed physical disk, $\mathcal{RS}$ reads the same number of symbols from a physical disk in different stripes by simply rotating. The disk read of $\mathcal{RS}$ is not balanced in a stripe, and it cannot be balanced by simply rotating disks. ∎

In the following, we will provide a method, *Leap Rotation*, to logically number physical disks. Leap rotation will balance disk reads in a group of successive $p - 1$ stripes, which are numbered $1, 2, ..., p - 1$.

**Definition** *4:* Given physical disk $D_0$, $D_1$, ..., $D_{p-1}$, the logical disk of $D_k$ with $l$-th *Leap Rotation* $\mathcal{LP}_l$ is $LD_{\langle k \times l \rangle_p}$, $k = 0, 1..., p - 1, l = 1, 2, ..., p - 1$.

Fig. 7 is an example of leap rotation within $p - 1$ successive stripes with $p = 5$, where the $l$-th stripe is implemented with the $l$-th leap rotation $\mathcal{LP}_l$. To show that leap rotation balances disk read in $p-1$ stripes while matching the lower bound of disk read, we first present some properties of a prime number in Lemma 7.

**Lemma** *7:* Given a prime number $p$, we have

| | $LD_0$ | $LD_1$ | $LD_2$ | $LD_3$ | $LD_4$ |
|---|---|---|---|---|---|
| Stripe 1 | $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ |
| Stripe 2 | $D_0$ | $D_3$ | $D_1$ | $D_4$ | $D_2$ |
| Stripe 3 | $D_0$ | $D_2$ | $D_4$ | $D_1$ | $D_3$ |
| Stripe 4 | $D_0$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ |

Fig. 7. Logical disks in different stripes when $p = 5$.

1) for $1 \leq l \leq p - 1$, $\{\langle l \times j \rangle_p | j = 0, 1, ..., p - 1\} = \{0, 1, ..., p - 1\}$.
2) for any $x, y$ with $0 \leq x, y \leq p - 1$, $x \neq y$, $\{\langle x \times l - y \times l \rangle_p | l = 1, 2, ..., p - 1\} = \{1, 2, ..., p - 1\}$.

**Proof:** If $\langle l \times j_1 \rangle_p = \langle l \times j_2 \rangle_p$ for some $0 \leq j_1, j_2 \leq p - 1$, we can conclude that $\langle l \times (j_1 - j_2) \rangle_p = 0$. Because $p$ is a prime, this equation implies $p|l$ or $p|(j_1 - j_2)$. Because $1 \leq l \leq p - 1$ and $0 \leq j_1, j_2 \leq p - 1$, $j_1 = j_2$. So 1) of Lemma 7 is proved. Similarly, 2) of Lemma 7 concludes. ∎

**Theorem** *4:* A recovery sequence reads the same number of symbols from different disks in a group of $p - 1$ stripes if the $l$-th stripe is X-coded with the logical disks numbered by $l$-th leap rotation $\mathcal{LP}_l$.

**Proof:** 1) of Lemma 7 ensures that $\mathcal{LP}_l$ is reasonable, i.e. the logical numbers of all disks in $l$-th stripe are $0, 1, ..., p - 1$ respectively. So X-code can be implemented based on the logical disks.

From 2) of Lemma 7, for any two physical disks $D_x$ and $D_y$ ($x \neq y$), the logical distances from $D_x$ to $D_y$ in different stripes are different. So the logical distances from $D_x$ to $D_y$ in $p - 1$ stripes are $1, 2, ..., p - 1$ respectively. Suppose that physical disk $D_k$ fails, for

any surviving physical disk $D_j$ ($j \neq k$), the logical distances from $D_j$ to $D_k$ in $p-1$ stripes are $1, 2, ..., p-1$ respectively. So the number of symbols read from $D_j$ in $p-1$ stripes is exactly the sum of the number of symbols read from all surviving disks in one stripe, or the number of symbols read from all surviving disks for the recovery of a stripe, $\frac{3p^2-8p+13}{4}$. So disk read can be balanced in $p-1$ successive stripes with leap rotations. ■

**Remark:** Usually the size of the data in a stripe is much smaller than the memory size of a disk system. So the data in a group of $p-1$ stripes can be read from the disks to the memory in one round of disk reads. Hence, the disk read of a recovery sequence can be balanced in a group of $p-1$ stripes.

With the *Leap Rotation*, we propose a recovery approach called Group-based MDRR (GMDRR) as: (1) In the $l$-th stripe, the X-code is implemented with the logical disks based on $\mathcal{LP}_l$. (2) MDRR is executed in each stripe.

## 6 Experiments on a Parallel Storage Testbed

We empirically evaluate different recovery schemes for X-code in a real networked storage system deployed in a local area network. We compare three recovery algorithms: (i) the *conventional* approach that downloads the entire original file and recovers the lost data, (ii) the MDRR algorithm, and (iii) the GMDRR algorithm. Our goal is to validate our theoretical analysis in a real network environment, by showing that our proposed MDRR and GMDRR algorithms actually improve the recovery performance of X-code in practice. Note that our experiments are different from the simulations used by [28], as we consider read/write operations on real storage devices over a networked environment so as to capture the actual recovery performance in realistic settings.

### 6.1 Methodology

We implement the recovery schemes on NCFS [40], a network-coding-based parallel/distributed file system. NCFS manages all data read/write operations and transparently stripes data across $p$ storage nodes, each corresponding to a disk or a storage device. In the current NCFS implementation, each symbol corresponds to a *chunk*. Note that NCFS supports the recovery operation for a single-node failure. The recovery operation consists of three steps: (i) reading data from surviving nodes, (ii) reconstructing lost data inside NCFS, and (iii) writing data to a new node.

Fig. 8 shows the network topology considered in our experiments. We deploy NCFS on a Linux-based server equipped with Intel Quad-Core 2.66GHz, 4GB RAM, and a harddisk with the SATA interface. NCFS interconnects multiple storage nodes via a 1-Gbps switch. There are

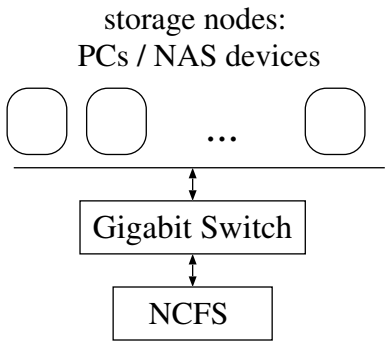



Fig. 8. Topology used in our testbed. Each storage node corresponds to a disk.

two types of storage nodes: (i) Pentium 4 PCs, each equipped with a 100-Mbps Ethernet interface, and (ii) network attached storage (NAS) devices, each equipped with a 1-Gbps Ethernet interface. We consider two types of topologies: (i) *homogeneous*, in which all storage nodes are PCs, and (ii) *heterogeneous*, in which the storage nodes are a mixture of PCs and NAS devices. In each topology, we also have a spare Pentium 4 PC that serves as the new node where the recovery operation places the recovered data. Furthermore, we configure the storage volume of each storage node as 1GB in our experiments.

We first write 40 files of size 100MB each to NCFS, which then stripes the data across the storage nodes. We choose to store large files to mimic the file patterns in real-life distributed storage systems (e.g., [1]). We then disable one node, and activate the recovery operation of NCFS to recover the data of the failed node in a new node. We consider both *offline* and *online* recovery modes, i.e., no files are read and files are being read during recovery, respectively. For each recovery scheme, we evaluate the overall *recovery time per megabyte of data*. Our results are averaged over five runs.

### 6.2 Results

**Summary of results.** Experiments show that MDRR and GMDRR reduce the recovery time of the conventional approach by around 20%, conforming to the theoretical findings. Specifically, in the heterogeneous topology, MDRR and GMDRR can reduce the recovery time by 18.0% and 22.0%, respectively, and GMDRR generally uses less recovery time than MDRR regardless of which node (column) is failed. The improvements of MDRR and GMDRR are similar in both offline and online recovery modes.

**Experiment 1 (Impact of chunk size on the recovery performance).** We first evaluate the impact of chunk size on the recovery performance. Here, we vary the chunk size from 4KB to 1024KB, which is configurable in the current NCFS implementation. Fig. 9 shows the recovery times for the conventional approach and MDRR. We observe that in both approaches, the recovery time decreases as the chunk size increases, mainly because

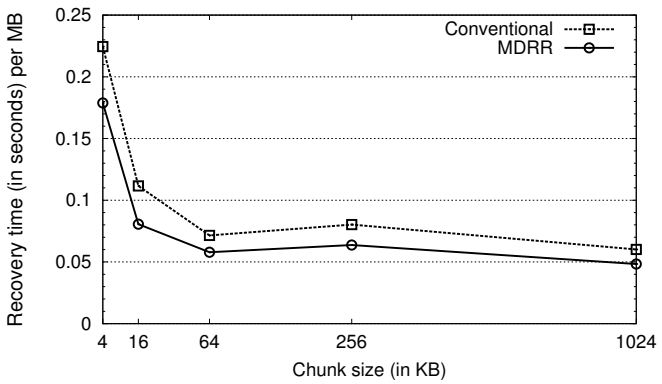



Fig. 9. Experiment 1: Impact of chunk size.

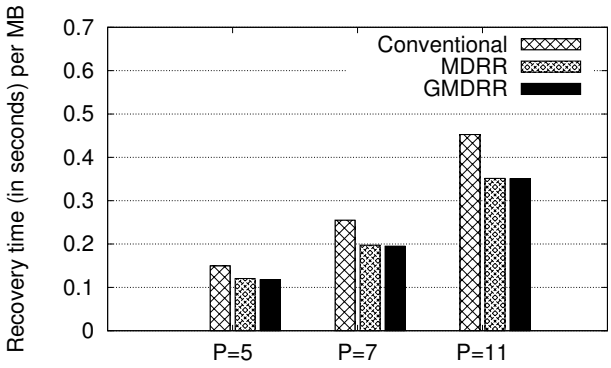



Fig. 10. Experiment 2: Recovery time in the homogeneous topology (Column 0 is failed).

the number of disk I/O requests decreases with a larger chunk size. Note that the rate of increase diminishes as the chunk size further increases. Thus, we expect that the recovery time stabilizes for a large chunk size. Nevertheless, for all chunk sizes that we choose, our proposed MDRR approach outperforms the conventional approach consistently.

In the following experiments, we fix the chunk size to be 4KB, which is the default disk block size in existing Linux extended file systems.

**Experiment 2 (Recovery in the homogeneous topology).** We now evaluate the recovery performance in the homogeneous topology. We consider $p$ = 5, 7, and 11 storage nodes. Fig. 10 shows the recovery time (in offline recovery mode) when the failure is in Column 0. We observe that both MDRR and GMDRR reduce the recovery time of the conventional approach, for example, by 22.3% and 22.5% when $p$ = 11, respectively. Note that the difference between MDRR and GMDRR is very small in the homogeneous setting.

We also look into the performance breakdown (not shown in the figure), and find that the step of reading data from the survival nodes accounts for more than 90% of the overall recovery time. This also justifies our objective of minimizing the data reads during recovery.

We point out that our experimental results are consistent with our theoretical findings. In theory, the improvements of MDRR over the conventional approach are 20% ($p$ = 5), 25.7% ($p$ = 7), and 27.3% ($p$ = 11), while our experiments indicate that the improvements are 19.5% ($p$ = 5), 22.6% ($p$ = 7), and 22.32% ($p$ = 11). We observe a slight drop in improvements in our experiments, mainly because of the additional overhead of reconstructing data and writing data to a new node.

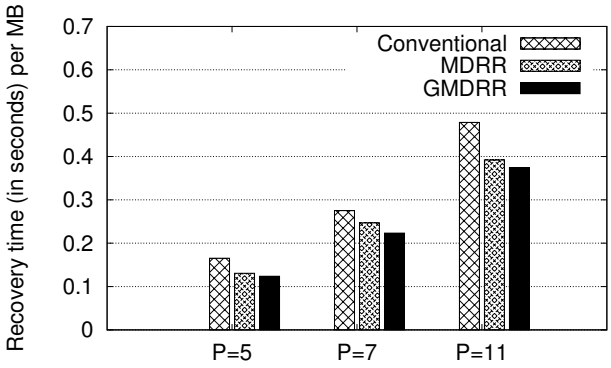



Fig. 11. Experiment 3: Recovery time in the heterogeneous topology.

**Experiment 3 (Recovery in the heterogeneous topology).** We now evaluate the recovery performance when the storage nodes are of different types. We consider three setups: (i) $p$ = 5, with 3 PCs in Columns 0-2 and 2 NAS devices in Columns 3-4, (ii) $p$ = 7, with 4 PCs in Columns 0-3 and 3 NAS devices in Columns 4-6, and (iii) $p$ = 11, with 8 PCs in Columns 0-7 and 3 NAS devices. Recall that the NAS devices have a higher access speed (with 1-Gbps interface) than the PCs (with only 100-Mbps interface).

Fig. 11 shows the recovery time (in offline recovery mode) when the failed node is in Column 0. Both MDRR and GMDRR reduce the recovery time of the conventional method, say, by 18.0% and 22.0% when $p$ = 11, respectively. The advantage of GMDRR over MDRR is more obvious in this case since it seeks to balance the number of disk reads and will not download more chunks from the nodes that have a slower access speed (e.g., PCs in our case).

To further evaluate GMDRR, suppose now that the failed node appears in another different column (i.e., aside from Column 0). Fig. 12 shows how the location of the failed node affects the recovery time performance when $p$ = 5. Both MDRR and GMDRR still reduce the recovery time of the conventional approach regardless of which node (column) is failed, while the improvement of GMDRR is more significant than MDRR in some cases. In particular, if the failed node is located in Column 1, GMDRR further reduces the recovery time of MDRR by at least 9%.

**Experiment 4 (Online recovery).** In this experiment, we study the online recovery mode, in which we repair a failed node while files are being read. During the recovery process, we also independently download a number of files (of size 100MB each) from NCFS. This

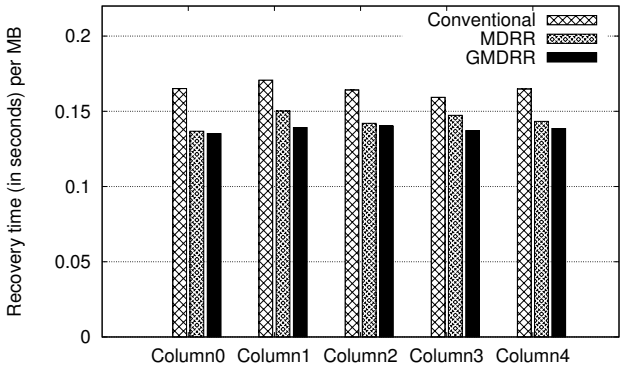



Fig. 12. Experiment 3: Recovery time in the heterogeneous topology when the failed node is located in another different column ($p$ = 5).

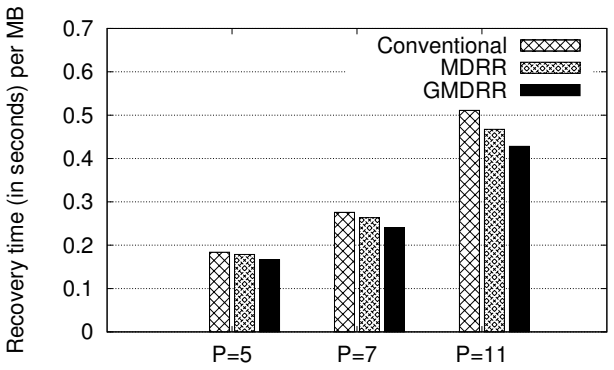



Fig. 13. Experiment 4: Recovery time in online recovery mode.

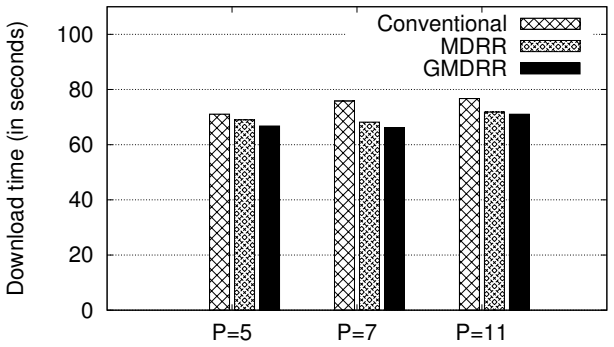



Fig. 14. Experiment 4: File download time during online recovery.

mimics a read-intensive access pattern. Here, we focus on the heterogeneous topology as in Experiment 2.

Fig. 13 shows the recovery time (now in online recovery mode) for different recovery schemes when the failed node is Column 0. In general, the recovery time for each scheme is larger than that in offline recovery mode (see Fig. 10), yet the improvements of MDRR and GMDRR over the conventional approach still exist. For example, GMDRR reduces the recovery time of the conventional approach by 16.26% when $p$ = 11. Note that GMDRR outperforms MDRR (e.g., the recovery time is reduced by 8.4%) by balancing the data reads among the storage nodes.

While the emphasis of our work is on improving the recovery performance, MDRR and GMDRR are also beneficial to file downloads during recovery. We measure the per-file download time for the files whose entire downloads occur during the time window of the recovery operation. Fig. 14 plots the per-file download time for different recovery schemes. Both MDRR and GMDRR reduce the per-file download time compared to the conventional approach (e.g., by at least 6% when $p$ = 11), since they minimize the data reads during the recovery process. We emphasize that we here only provide a preliminary study on how the online recovery using MDRR and GMDRR can improve normal usage, while the actual improvements depend on the workload of file access patterns. We pose the further analysis as future work.

## 7 Conclusions

We study the optimal recovery problem of a single-disk/node failure in X-code-based parallel storage systems. Since existing optimal recovery solutions are mainly designed for RAID-6 horizontal codes, to our knowledge, this is the first work that addresses the optimal recovery problem of RAID-6 vertical codes. We propose a recovery algorithm MDRR which reduces the disk reads about 25% compared with the conventional recovery scheme, and matches the theoretical lower bound of disk reads for the recovery. Because MDRR issues unbalanced disk read among different disks and its disk read cannot be balanced by disk rotation, we present a leap rotation scheme which makes sure that MDRR issues balanced disk read among disks in a group of $p-1$ stripes. The principle of leap rotation is implementing data encoding based on logical number of disks, and this rotation scheme can be applied to balance disk reads in storage systems with different codes, which is one of our future work.

## References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proc. of ACM SOSP*, 2003, pp. 29–43.

[2] 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, pp. 205–220.

[3] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," in *Proc. of ASPLOS*, 2000, pp. 190–201.

[4] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management," in *Proc. of USENIX NSDI*, 2004, pp. 337–350.

[5] Wuala, http://www.wuala.com.

[6] I. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[7] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Proc. of IPTPS*, 2002, pp. 328–338.

[8] B. Battles, C. Belleville, S. Grabau, and J. Maurier, "Reducing Data Center Power Consumption Through Efficient Storage," NetApp, Tech. Rep. WP-7010-0207, Feb 2007.

[9] J. Hamilton, "Cost of power in large-scale data centers," Nov 2009, http://perspectives.mvdirona.com.

[10] 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, pp. 1–14.

[11] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 192–202, Feb 1995.

[12] L. Xu and J. Bruck, "X-code: MDS array codes with optimal encoding," *IEEE Trans. on Information Theory*, vol. 45, no. 1, pp. 272–276, 1999.

[13] C. Huang and L. Xu, "STAR: An efficient coding scheme for correcting triple storage node failures," in *Proc. of USENIX FAST*, 2005.

[14] C. Jin, D. Feng, H. Jiang, and L. Tian, "A comprehensive study on raid-6 codes: Horizontal vs. vertical," in *Proc. of IEEE NAS*, 2011.

[15] M. Baker, M. Shah, D. S. H. Rosenthal, M. Roussopoulos, P. Maniatis, T. Giuli, and P. Bungale, "A fresh look at the reliability of long-term digital storage," in *Proc. of EuroSys*, 2006, pp. 221–234.

[16] A. G. Dimakis, P. B. Godfrey, M. J. Wainwright, and K. Ramchandran, "The benefits of network coding for peer-to-peer storage systems," in *Proc. of NetCod*, January 2007.

[17] R. R. Muntz and J. C. S. Lui, "Performance analysis of disk arrays under failure," in *Proc. of VLDB*, 1990, pp. 162–173.

[18] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *IEEE Proceedings*, vol. 99, pp. 476–489, Mar 2011.

[19] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 56, pp. 4539–4551, 2010.

[20] Y. Wu, A. Dimakis, and K. Ramchandran, "Deterministic regenerating codes for distributed storage," in *Proc. of Allerton Conference on Control, Computing, and Communication*, 2007.

[21] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. on Information Theory*, vol. 46, no. 4, pp. 1204–1216, Jul 2000.

[22] Y. Wu, "Existence and construction of capacity-achieving network codes for distributed storage," *IEEE J.Sel. A. Commun.*, vol. 28, pp. 277–288, February 2010.

[23] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, "Regenerating codes for distributed storage networks," in *Proc. of international conference on Arithmetic of Finite Fields*, 2010, pp. 215–223.

[24] Y. Hu, Y. Xu, X. Wang, C. Zhan, and P. Li, "Cooperative recovery of distributed storage systems from multiple losses with network coding," *IEEE J.Sel. A. Commun.*, vol. 28, pp. 268–276, Feb 2010.

[25] C. Suh and K. Ramchandran, "Exact-repair mds codes for distributed storage using interference alignment," in *Proc. of IEEE ISIT*, 2010.

[26] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," in *Proc. of Allerton Conference*, 2009.

[27] Z. Wang, A. G. Dimakis, and J. Bruck, "Rebuilding for array codes in distributed storage systems," in *Proc. of ACTEMT*, 2010.

[28] L. Xiang, Y. Xu, J. C. Lui, Q. Chang, Y. Pan, and R. Li, "A hybrid approach of failed disk recovery using RAID-6 codes: Algorithms and performance evaluation," *ACM Transactions on Storage*, vol. 7, no. 3, Oct 2011.

[29] Q. Cao, S. Wan, C. Wu, and S. Zhan, "An evaluation of two typical raid-6 codes on online single disk failure recovery," in *Networking, Architecture and Storage (NAS), 2010 IEEE Fifth International Conference on*. IEEE, 2010, pp. 135–142.

[30] S. Li, Q. Cao, J. Huang, S. Wan, and C. Xie, "Pdrs: A new recovery scheme application for vertical raid-6 code," in *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*. IEEE, 2011, pp. 112–121.

[31] 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.

[32] Y. Zhu, P. P. C. 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.

[33] 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.

[34] M. Sivathanu, V. Prabhakaran, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Improving storage system availability with d-graid," *ACM Transactions on Storage (TOS)*, vol. 1, no. 2, pp. 133–170, 2005.

[35] L. Tian, D. Feng, H. Jiang, K. Zhou, L. Zeng, J. Chen, Z. Wang, and Z. Song, "Pro: A popularity-based multi-threaded reconstruction optimization for raid-structured storage systems," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, 2007, pp. 277–290.

[36] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao, "Workout: I/o workload outsourcing for boosting raid reconstruction performance," in *Proccedings of the 7th conference on File and storage technologies*. USENIX Association, 2009, pp. 239–252.

[37] S. Wan, Q. Cao, J. Huang, S. Li, X. Li, S. Zhan, L. Yu, C. Xie, and X. He, "Victim disk first: an asymmetric cache to boost the performance of disk arrays under faulty conditions," in *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*. USENIX Association, 2011, pp. 13–13.

[38] T. Xie and H. Wang, "Micro: A multilevel caching-based reconstruction optimization for mobile storage systems," *IEEE Transactions on Computer*, vol. 57, no. 10, pp. 1386–1398, 2008.

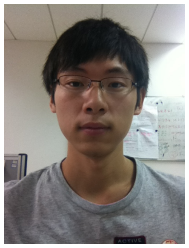
[39] J. B. Lihao Xu, Vasken Bohossian and D. G. Wagner, "Low-density mds codes and factors of complete graphs," *IEEE Transactions on Information theory*, vol. 45, no. 6, pp. 1817–1826, 1999.

[40] 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.

[41] D. S. Papailiopoulos, A. G. Dimakis, and V. R. Cadambe, "Repair optimal erasure codes through hadamard designs," in *Proc. of Allerton*, 2011.

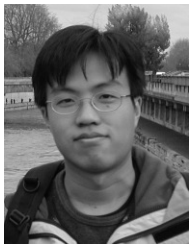
[42] I. Tamo, Z. Wang, and J. Bruck, "Mds array codes with optimal rebuilding," in *Proc. of IEEE ISIT*, 2011.

[43] K. Shanmugam, D. S. Papailiopoulos, A. G. Dimakis, and G. Caire, "A Repair Framework for Scalar MDS Codes," in *Proc. of Allerton*, 2012.

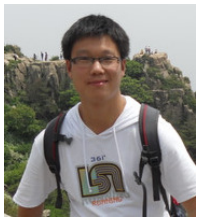



**Silei Xu** received his B.S. from the School of Computer Science, University of Science and Technology of China, Anhui, China, in 2012. He is going to work toward to his M.phil. degree in Computer Science and Engineering at the Chinese University of Hong Kong. His research interests include distributed storage system, data recovery and smart phone.

**Runhui Li** received his B.S. from the School of Computer Science, University of Science and Technology of China, Anhui, China, in 2011. He is currently working toward the Ph.D. degree in Computer Science and Engineering at the Chinese University of Hong Kong. His research interests include distributed storage system, cloud computing and security.



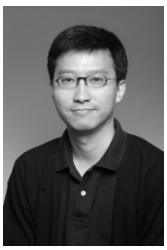
**Patrick P. C. Lee** received the B.Eng. degree (first-class honors) in Information Engineering from the Chinese University of Hong Kong in 2001, the M.Phil. degree in Computer Science and Engineering from the Chinese University of Hong Kong in 2003, and the Ph.D. degree in Computer Science from Columbia 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 various applied/systems topics including cloud computing and storage, distributed systems and networks, operating systems, and security/resilience.



**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.

**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.

**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.



**John C. S. Lui** received his Ph.D. in Computer Science from UCLA. He is currently a Professor with the Department of Computer Science and Engineering at the Chinese University of Hong Kong. He was the chairman of the Department from 2005 to 2011. His current research interests are in data networks, system and applied security, multimedia systems, network sciences and cloud computing. He is a Fellow of the Association for Computing Machinery (ACM), a Fellow of IEEE, a Croucher Senior Research Fellow, and an elected member of the IFIP WG 7.3.