Preventing Silent Data Corruptions from Propagating During Data Reconstruction

Mingqiang Li, *Student Member*, *IEEE*, and Jiwu Shu, *Member*, *IEEE*

Abstract—One recent technical challenge facing the designers of erasure-coded storage systems is how to prevent silent data corruptions from propagating during data reconstruction. This paper proposes a new technique of exploiting erasure-coded storage systems to cope with silent data corruptions during data reconstruction. To develop a data reconstruction method that can prevent silent data corruptions from propagating, we first define the *consistency* of a strip group and then study the impact of silent data corruptions on the consistency of strip groups. Based on the conclusions obtained from the study, an efficient adaptive data reconstruction method is developed for data reconstruction in the presence of silent data corruptions. A performance analysis of our new data reconstruction method is then made using a probabilistic method. Our results show that the overall performance impact of our data reconstruction method is negligible in practical systems. A comparison of techniques for coping with silent data corruptions in erasure-coded storage systems is also made. The comparison shows that the technique based on our data reconstruction method is a better choice to cope with silent data corruptions when periodic validation is used in an erasure-coded storage system.

Index Terms—Data reconstruction, erasure code, silent data corruption, storage system.

1 Introduction

Data availability and reliability have become the primary concern of most business and government organizations. Unexpected data loss or corruption can sometime be seen as a catastrophic incident to these organizations.

However, as today's storage systems grow in size and complexity, they are increasingly confronted with concurrent disk failures [1], [2] together with multiple unrecoverable sector errors [3], [4], [5]. To prevent data loss caused by these disk errors, redundancy technologies are commonly used in modern storage systems. Among these technologies, erasure coding [6] is one of the most attractive technologies because of its high fault tolerance and optimal (or approximately optimal) storage efficiency. It has been widely adopted in RAID subsystems [7], [8] and various fault-tolerant storage systems, such as OceanStore [9], Glacier [10], FAB [11], PASIS [12], [13], RobuSTore [14], Pergamum [15], Cleversafe [16], Allmydata [17], and Permabit [18]. We call such storage systems as *erasure-coded storage systems*. In erasure-coded storage systems, when some detectable errors (including disk failures and unrecoverable sector errors) occur, the lost data caused by them can be reconstructed from the redundant data encoded by erasure codes.

Unfortunately, there also exist a significant number of *silent data corruptions* [19], [20], [21], [22], [23], [24], [25] (where the data is silently corrupted with no indication from the disk array subsystem that an error has occurred) in erasure-coded storage systems. A detailed introduction to silent data corruptions in erasure-coded storage systems will be given in Section 2.2. Silent data corruptions are much more dangerous than detectable errors because they cannot be directly detected and thus will not be recovered by erasure-coded storage systems. More seriously, silent data corruptions can propagate during data reconstruction when we reconstruct lost data using a traditional data reconstruction method (such as the method proposed in [26]) that only considers the issues of detectable errors. The traditional data reconstruction method even can change some detectable errors into silent data corruptions during data reconstruction, giving rise to a ripple effect for silent errors. This further accelerates the increase of silent data corruptions. If the number of silent data corruptions increases to a certain extent, they can even cause unrepairable system errors, resulting in unexpected loss. To prevent this from happening, silent data corruptions should be prevented from propagating during data reconstruction. Meanwhile, they are expected to be recovered simultaneously with detectable errors during data reconstruction. Therefore, one technical challenge facing the designers of erasure-coded storage systems is how to prevent such silent data corruptions from propagating while lost data is being reconstructed.

To cope with this challenge, the traditional method is to co-locate metadata with disk data in an attempt to make silent data corruptions detectable [20], [23], [24]. This metadata can contain various types of extra information, such as checksums or version numbers, which can be used to detect certain types of silent data corruptions. In erasure-coded storage systems, only the information obtained in the RAID-like layer is expected to be contained in this metadata according to the design principle of an abstraction layer framework. However, the metadata method has a remarkable drawback: it requires additional storage overhead to store metadata and thus can involve additional I/O overhead during write operations. Besides,

• *The authors are with the Institute of High-Performance Computing, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China.*
E-mail: lmq06@mails.tsinghua.edu.cn; shujw@tsinghua.edu.cn.

Manuscript received 4 May 2009; revised 9 Oct. 2009; accepted 17 Jan. 2010; published online 10 Feb. 2010.
Recommended for acceptance by C. Bolchini.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2009-05-0191.
Digital Object Identifier no. 10.1109/TC.2010.36.

since colocating metadata with disk data complicates the process of each update operation, it thus can increase the chance of update failures, which may cause new silent data corruptions in reverse. In addition, as will be shown in this paper, the metadata method also has a defect: it cannot detect all kinds of silent data corruptions that occur in the RAID-like layer.

With the limitations of the metadata method, the question is: *Can we find a method that does not require colocating metadata with disk data but can still prevent silent data corruptions from propagating during data reconstruction?* This paper will give an answer to this question.

We noticed that besides the ability to recover from detectable errors, erasure-coded storage systems also have the ability to recover from silent data corruptions. In coding theory [27], it is well known that a k-of-$(k+m)$ erasure code has the ability to tolerate f lost strips together with r silently corrupted strips, where $f+2r \leq m$. Furthermore, our study in this paper reveals that the actual ability of an erasure code to tolerate errors is often much beyond this upper bound. Thus, erasure-coded storage systems also can be exploited to deal with silent data corruptions. Based on this property, we can develop a systematic way to cope with silent data corruptions during data reconstruction.

This paper presents, for the first time, a mathematical formulation and a thorough analysis of the propagation of silent data corruptions during data reconstruction in erasure-coded storage systems. Our analysis shows that the propagation probability of silent data corruptions is very high when we use a traditional data reconstruction method to reconstruct lost data. This reinforces the need for a new data reconstruction method to prevent silent data corruptions from propagating.

In the process of developing such a data reconstruction method, this paper first defines the *consistency* of a strip group, which is a metric indicating whether all the strips contained in the strip group hold the parity computation relations of the adopted erasure code. Then, a systematic study on the impact of silent data corruptions on the consistency of strip groups is presented. Based on the conclusions obtained from the study, an efficient adaptive data reconstruction method that can cope with both lost strips and silently corrupted strips is developed. To the best of the authors' knowledge, this is the first data reconstruction method that addresses the issues of silent data corruptions for erasure-coded storage systems. We then show when we can correctly reconstruct lost data using the new data reconstruction method and that the actual ability of an erasure code to tolerate errors is often much beyond the well-known upper bound that $f+2r \leq m$. To understand the overall performance impact of the new data reconstruction method in practical systems, this paper also makes a performance analysis using a probabilistic method. The results show that the overall performance impact of the new data reconstruction method is negligible in practical systems.

Finally, this paper makes a comparison of techniques for coping with silent data corruptions in erasure-coded storage systems. Two kinds of techniques that can be integrated into the RAID-like layer, i.e., the traditional metadata techniques and the new technique based on the new data reconstruction method, are compared. These two kinds of techniques make different tradeoffs among cost, performance, and effectiveness. The comparison shows that the technique based on the new data reconstruction method is a better choice to cope with silent data corruptions when periodic validation is used in an erasure-coded storage system.

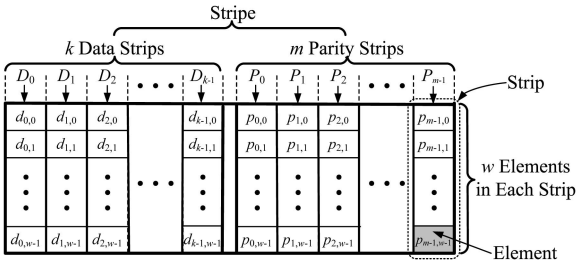

Fig. 1. Element, strip, and stripe in a horizontal erasure code.

The paper is organized as follows: Next section gives the necessary background and describes the problem of the propagation of silent data corruptions during data reconstruction. In Section 3, we first define the consistency of a strip group and then study the impact of silent data corruptions on the consistency of strip groups. An efficient adaptive data reconstruction method that can cope with both lost strips and silently corrupted strips is then developed in Section 4. In Section 5, we study the overall performance impact of the new data reconstruction method in practical systems. Section 6 makes a comparison of techniques for coping with silent data corruptions in erasure-coded storage systems. Some related work on silent data corruptions is then discussed in Section 7. Finally, Section 8 concludes the paper.

2 Background and Problem Description

2.1 Erasure Codes—A Brief Overview

In this subsection, we provide a brief overview of erasure codes [6]. We begin with the definition of three basic terms that are widely used in the literature of erasure codes:

Element: A fundamental unit of data or parity that can be a bit, a byte, a sector, or a larger disk block. This is the building block of an erasure code. In coding theory [27], this is the data that is assigned to a bit within a code symbol.

Stripe: A maximum set of data and parity elements that are dependently related by the parity computation relations of an erasure code. This is synonymous with a *code word* [27] in that it is a complete word of an erasure code and is independent of any other word.

Strip[1]: A maximum set of elements within a stripe that are on the same disk. In coding theory [27], this is the data mapped to a code symbol.

Here, we also define the term *strip group* that is often used in this paper:

Strip Group: A group of strips within a stripe; denoted by $\mathbb{G}$.

Fig. 1 shows a representation of our notions of element, strip, and stripe in a typical *horizontal erasure code* (in which data and parity elements within a stripe are stored on separate strips).

1. Compared with the term *chunk* that is often used in striped disk arrays to define the granularity of data or parity from a single disk, *strip* is a stricter term whose application scope is limited to erasure-coded storage systems.

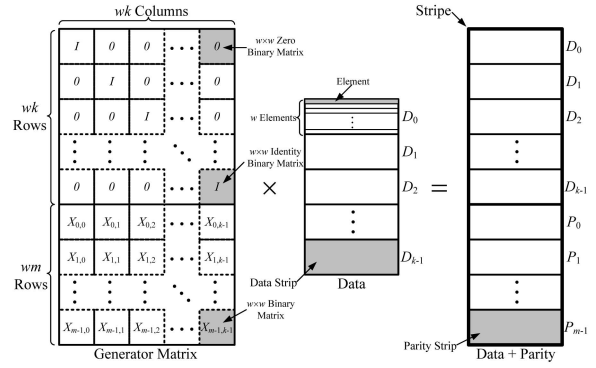

Fig. 2. Encoding using a binary matrix in an erasure code.

Many erasure codes have been proposed over the last 40 years, especially during the last 20 years. Plank [6] classifies them into three categories: Reed-Solomon codes [28], [29], [30], [31], parity array codes [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], and Low Density Parity Check (LDPC) codes [43], [44], [45], [46], [47]. Note that although there also exist other families of erasure codes that do not belong to Plank's classification, such as GRID codes proposed by Li et al. [48], to simplify our discussion, we still use Plank's classification and only consider the three categories of erasure codes mentioned in [6]. Among these three categories of erasure codes, Reed-Solomon codes and parity array codes are much more suitable for storage systems than LDPC codes and have been employed for fault tolerance in most practical systems [7], [8], [9], [10], [15], [16], [17], [18]. Besides, horizontal codes (such as Reed-Solomon codes) seem to be more popular because they store data and parity elements on separate strips and can be implemented more flexibly. Thus, in this paper, we focus our attention on Reed-Solomon codes and horizontal parity array codes. Moreover, we focus our discussion primarily on *Maximum Distance Separable (MDS) codes*[2] since they are typical representatives among all horizontal codes. Therefore, throughout this paper, when referring to erasure codes, we always mean MDS horizontal erasure codes.

In general, a k-of-$(k+m)$ erasure code encodes a stripe of data into $n = k + m$ strips that are of the same size, such that any k strips can be used to reconstruct the original user data. We use $\mathbb{D}$ and $\mathbb{P}$ to denote the set of data strips and the set of parity strips (i.e., $\mathbb{D} = \{D_0, D_1, \ldots, D_{k-1}\}$ and $\mathbb{P} = \{P_0,\ P_1, \ldots, P_{m-1}\}$), respectively. When a write operation updates a data strip, it should update all the corresponding m parity strips at the same time. This process should be atomically implemented. When m or fewer strips are lost, they can be reconstructed from any combination of k strips chosen from the set of the remaining strips.

All erasure codes can be represented by binary matrices that are the basic structures upon which we perform encoding and decoding. In a k-of-$(k+m)$ erasure code, suppose there are w elements in each strip (see Fig. 1). We can use a $w(k+m) \times wk$ binary matrix to perform encoding (see Fig. 2). The transpose of this matrix is called as a *generator matrix*, denoted by G. Then, the binary matrix in Fig. 2 is G^T. As shown in Fig. 2, G^T has a specific format. Its first wk rows form a $wk \times wk$ identity matrix. The next wm rows consist of mk matrices, each of which is a $w \times w$ binary matrix (denoted by $X_{i,j}$). Thus, G^T can be expressed as

$$G^T = \begin{pmatrix} I \\ X \end{pmatrix},$$

where I is a $wk \times wk$ identity matrix, and $X = (X_{i,j})_{m \times k}$.

Let two binary matrices D and P represent the data and parity in a stripe S, respectively. Then, S is expressed as

$$S = \begin{pmatrix} D \\ P \end{pmatrix}.$$

Suppose there are l bits in each element. Then, D and P can be expressed as $D = (D_i)_{k \times 1}$ and $P = (P_j)_{m \times 1}$, respectively, where D_i and P_j are two $w \times l$ binary matrices corresponding to a data strip and a parity strip, respectively. During data encoding, we calculate S by the expression $S = G^T \cdot D$. Then, it is clear that $P = X \cdot D$. Consequently, we have the following binary linear equations:

$$\begin{cases} X_{0,0}D_0 + X_{0,1}D_1 + \cdots + X_{0,k-1}D_{k-1} = P_0, \\ X_{1,0}D_0 + X_{1,1}D_1 + \cdots + X_{1,k-1}D_{k-1} = P_1, \\ \qquad\qquad \vdots \\ X_{m-1,0}D_0 + X_{m-1,1}D_1 + \cdots + X_{m-1,k-1}D_{k-1} = P_{m-1.} \end{cases} \quad (1)$$

Any square submatrix of the coefficient matrix of the above equations should be an invertible matrix. When m or fewer strips are lost, we can reconstruct them by solving the above equations using the matrix method proposed in [26].

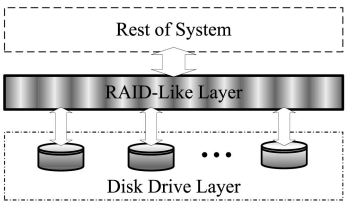

Fig. 3. Two layers of the disk array subsystem in an erasure-coded storage system.

2.2 Silent Data Corruptions in Erasure-Coded Storage Systems

In an erasure-coded storage system, the disk array subsystem consists of two layers: the disk drive layer and the RAID-like layer (see Fig. 3). Here, the RAID-like layer can be implemented either in software or in hardware. These two layers both can cause silent data corruptions.

In the disk drive layer, disk drives can experience errors that are undetected by the drive—which is referred to as *Undetected Disk Errors (UDEs)* in [24]. UDEs are caused by various disk-related firmware and hardware malfunctions. According to when they are caused, Hafner et al. [24] divides UDEs into two classes: *Undetected Read Errors (UREs)* and *Undetected Write Errors (UWEs)*. It should be noted that for either UREs or UWEs, the symptoms for applications are the same—stale or corrupted data[3] returned for a read operation. Typically, the effects of UREs

2. *MDS codes* are the family of erasure codes that attain the Singleton bound [27] and thus can provide optimal storage efficiency.

3. The difference between stale data and corrupted data is that stale data is correct but out-of-date data, while corrupted data is incorrect data.

are transient, while the effects of UWEs persist. Thus, UWEs are more significant problems than UREs and are widely concerned by both the academic and industry research [19], [20], [22], [23], [24].

There are three representative types of UWEs deserving mention here: *misdirected writes* [19], [22], [23] (also called as *off-track writes* in [24]), *lost writes* [20], [22], [23] (also called as *phantom writes* in [19] or *dropped writes* in [24]), and *torn writes* [22], [23]. They are caused as follows:

Misdirected Writes: A buggy disk controller with firmware errors could issue a misdirected write that places the correct data on the disk but in the wrong location.

Lost Writes: A disk head that does not produce a sufficiently strong signal in the right location to override and reset the bits of the data already present can cause a lost write that the drive reports as completed but that never reaches the media.

Torn Writes: A disk drive that is power-cycled in the middle of processing a write request can cause a torn write that writes only a portion of the sectors successfully in the write request.

In the RAID-like layer, one special process that should be atomically implemented is that when a write operation updates a data strip, it should update all the corresponding parity strips at the same time. During this process, if some errors (such as in-memory corruptions, processor miscalculations, or software bugs) occur in the RAID-like layer, some of the update operations may be lost or be misimplemented. Silent data corruptions of this kind are referred to as *Strip Update Errors (SUEs)* in this paper. SUEs can result in *parity inconsistencies*, i.e., the cases of mismatch between data and parity stored in disk drives. Besides, it should be noted that of UDEs, UWEs can also cause parity inconsistencies.

From the above discussion, we can see that there exist two categories of silent data corruptions in erasure-coded storage systems: UDEs and SUEs, which come from two different layers and occur independently from each other. They both can cause silently corrupted strips in erasure-coded storage systems.

2.3 The Propagation of Silent Data Corruptions During Data Reconstruction

After giving a necessary background on silent data corruptions in erasure-coded storage systems, we now discuss the propagation of silent data corruptions during data reconstruction.

In a k-of-$(k+m)$ erasure-coded storage system, when f strips are lost or are identified to be corrupted in a stripe (where $1 \le f \le m$), the common way to handle this error is to first randomly choose a combination of k strips from the set of the remaining $n-f$ strips and then reconstruct lost data from the chosen combination. It is clear that there are $\binom{n-f}{k}$ such combinations.

Now, suppose there exist r silently corrupted strips in the set of the remaining $n-f$ strips, where $0 \le r \le n-f$. If the chosen combination is one of the $\binom{n-f-r}{k}$ combinations consisting of k correct strips, all lost strips can be correctly reconstructed; while if some silently corrupted strips are involved in the reconstruction operation, corrupt data will be produced. Thus, when we reconstruct lost data using a combination of k strips randomly chosen from the set of the remaining strips, the probability of producing corrupt data (denoted by $Pr_{corrupt}$) is given by

$$Pr_{corrupt} = \begin{cases} 0, & \text{if } r = 0; \\ 1 - \frac{\binom{n-f-r}{k}}{\binom{n-f}{k}}, & \text{if } 1 \le r \le m-f; \\ 1, & \text{if } m-f < r \le n-f. \end{cases} \quad (2)$$

TABLE 1
The Value of $Pr_{corrupt}$ for Different Error Scenarios in an Example System with $k = 10$ and $m = 6$

		r						
		0	1	2	3	4	5	≥ 6
f	1	0	0.6667	0.9048	0.9780	0.9963	0.9997	1
	2	0	0.7143	0.9341	0.9890	0.9990	1	1
	3	0	0.7692	0.9615	0.9965	1	1	1
	4	0	0.8333	0.9848	1	1	1	1
	5	0	0.9091	1	1	1	1	1
	6	0	1	1	1	1	1	1

Table 1 shows this probability for different error scenarios in an example system with $k = 10$ and $m = 6$ (such as Cleversafe's widely dispersed storage system [16], which uses these two configuration parameters as its default). It can be seen from this table that the probability of producing corrupt data with the existence of silently corrupted strips is very high and increases rapidly as the number of silently corrupted strips increases.

Because reconstructing lost data in a traditional manner could lead to the propagation of silent data corruptions, we immediately raise a question: *Can we develop an effective data reconstruction method to prevent silent data corruptions from propagating?*

To answer this question, we first recall some properties of erasure codes from coding theory. In coding theory [27], it is well known that if the locations of erroneous strips are not known in advance, a k-of-$(k+m)$ erasure code has the ability to detect and correct up to $m/2$ erroneous strips. Thus, in a k-of-$(k+m)$ erasure-coded storage system, when f strips are lost or are identified to be corrupted in a stripe (where $1 \le f \le m$), any combination of r silently corrupted strips within the same stripe can be detected and corrected as long as the condition $f + 2r \le m$ is satisfied. This property tells us that the propagation of silent data corruptions can be prevented during data reconstruction if we can develop a data reconstruction method specially for coping with silent data corruptions.

Now, the problem becomes how to efficiently reconstruct lost data in the presence of silent data corruptions. We will solve this problem in the following sections.

3 The Impact of Silent Data Corruptions on the Consistency of Strip Groups

Before discussing how to efficiently reconstruct lost data in the presence of silent data corruptions, we investigate the impact of silent data corruptions on the consistency of strip groups in this section.

3.1 The Consistency of Strip Groups

In this section, we first define the consistency of strip groups and then discuss how to determine the consistency.

Consistency($\mathbb{G}$){
(1) When $\mathbb{G}$ contains all the k data strips, for each parity strip in $\mathbb{G}$, first regenerate it from the k data strips and then compare the regenerated value with the original value in $\mathbb{G}$. If all the comparison outcomes are matching results, set $\Phi(\mathbb{G}) \leftarrow 0$; else, set $\Phi(\mathbb{G}) \leftarrow 1$.
(2) When $\mathbb{G}$ does not contain all the k data strips, first choose all the parity strips and a certain number of data strips in $\mathbb{G}$ to form a combination of k strips, then perform a reconstruction operation using the chosen combination, and finally, for each data strip that is not contained in the chosen combination but is contained in $\mathbb{G}$, compare the regenerated value with the original value in $\mathbb{G}$. If all the comparison outcomes are matching results, set $\Phi(\mathbb{G}) \leftarrow 0$; else, set $\Phi(\mathbb{G}) \leftarrow 1$.
(3) **Return** $\Phi(\mathbb{G})$.
}

Fig. 4. An approach to determine the consistency of a strip group $\mathbb{G}$.

Definition 3.1. *The* consistency *of a strip group $\mathbb{G}$ is a metric indicating whether all the strips contained in $\mathbb{G}$ hold the parity computation relations of the adopted erasure code. We use $\Phi(\mathbb{G})$ to denote the consistency of $\mathbb{G}$. If $\mathbb{G}$ is consistent, let $\Phi(\mathbb{G}) = 0$; else, let $\Phi(\mathbb{G}) = 1$.*

Obviously, we have the following two properties:

1. $\Phi(\mathbb{G}_1) = 0 \Rightarrow \forall \mathbb{G}_2 \subseteq \mathbb{G}_1 : \Phi(\mathbb{G}_2) = 0$;
2. $\Phi(\mathbb{G}_1) = 1 \Rightarrow \forall \mathbb{G}_2 \supseteq \mathbb{G}_1 : \Phi(\mathbb{G}_2) = 1$.

When a strip group contains not more than k strips, since no inconsistency with the parity computation relations of the adopted erasure code can be detected within the strip group, we regard the strip group to be consistent. Thus, in this paper, we will only consider the consistency of strip groups that contain more than k strips, and when referring to a strip group, we always mean a strip group containing more than k strips.

For a strip group $\mathbb{G}$, we give an approach to determine the consistency of $\mathbb{G}$ in Fig. 4. When $\mathbb{G}$ does not contain all the k data strips, the approach given in Fig. 4 can sufficiently utilize the reconstruction optimization technologies recently proposed in [26].

For a strip group consisting of correct strips, using the approach given in Fig. 4, we immediately obtain the following conclusion:

Theorem 3.1. *A strip group consisting of correct strips is always consistent.*

3.2 The Impact of Silent Data Corruptions

In this section, we study the impact of silent data corruptions on the consistency of strip groups.

As mentioned in Section 2.2, silent data corruptions can cause silently corrupted strips in erasure-coded storage systems. Here, we divide silently corrupted strips into two classes:

Stale Strips: The class of silently corrupted strips that completely consist of correct but out-of-date data caused due to the loss of update operations.

Corrupted Strips: The class of silently corrupted strips that contain incorrect data.

Note that a silently corrupted strip that partially contains stale data is also included in the class of corrupted strips.

TABLE 2
Notations for Describing Silently Corrupted Strips

Notation	Description
$D_i\langle stale\rangle$	A stale data strip of D_i.
$D_i\langle corrupted\rangle$	A corrupted data strip of D_i.
$P_j\langle stale \prec \Re\rangle$	A stale parity strip of P_j caused by the writes to $\Re$.
$P_j\langle corrupted\rangle$	A corrupted parity strip of P_j.

In this table, $0 \le i \le k-1$, $0 \le j \le m-1$, and $\Re$ is a nonempty set of data strips.

We then introduce some notations to describe silently corrupted strips in Table 2. Here, since the updates on parity strips are caused by the writes to data strips, we say a stale parity strip is caused by the writes to a nonempty set $\Re$ of data strips if all the corresponding updates on this parity strip were lost during the write processes of the data strips in $\Re$. For example, $P_j\langle stale \prec \{D_i\}\rangle$ is a stale parity strip of P_j caused due to the loss of the update on P_j during the write process of D_i, while $P_j\langle stale \prec \{D_{i_1}, D_{i_2}\}\rangle$ is a stale parity strip of P_j caused due to the loss of the two updates on P_j during the write processes of D_{i_1} and D_{i_2}.

Now, we begin to discuss the consistency of strip groups that contain silently corrupted strips. In the following discussion, we will focus our attention on when such strip groups can be consistent.

We first present a theorem as follows:

Theorem 3.2. *For a strip group $\mathbb{G}$ containing more than k strips, when all its data strips are correct data strips, and all its parity strips are stale parity strips caused by the writes to the same set of data strips, which does not contain any data strip in $\mathbb{G}$, then $\Phi(\mathbb{G}) = 0$.*

The correctness of Theorem 3.2 can be easily validated using the approach given in Fig. 4.

Example. We consider an example of such a strip group $\mathbb{G} = \{D_0, D_1, \ldots, D_8, P_0\langle stale \prec \Re\rangle, P_1\langle stale \prec \Re\rangle\}$ with $\Re = \{D_9\}$ taken from an erasure-coded storage system with $k = 10$ and $m = 6$. According to Theorem 3.2, we can deduce $\Phi(\mathbb{G}) = 0$. We now validate the correctness of $\Phi(\mathbb{G}) = 0$ using the approach given in Fig. 4 as follows: We perform a reconstruction operation using a combination $\{D_0, D_1, \ldots, D_7, P_0\langle stale \prec \Re\rangle, P_1\langle stale \prec \Re\rangle\}$ chosen from $\mathbb{G}$. In the corresponding stripe, the two data strips (i.e., D_8 and D_9) not contained in the chosen combination are then regenerated by this reconstruction operation. Their regenerated values are D_8 and $D_9\langle stale\rangle$, respectively. For the data strip D_8 in $\mathbb{G}$, its regenerated value is a correct value and thus is equal to the original value in $\mathbb{G}$. This validates the correctness of $\Phi(\mathbb{G}) = 0$.

Similarly, we can deduce the following conclusion:

Theorem 3.3. *For a strip group $\mathbb{G}$ containing more than k strips, when the set of its data strips consists of correct data strips and stale data strips, and all its parity strips are stale parity strips caused by the writes to the same set of data strips, which contains all the stale data strips in $\mathbb{G}$ but does not contain any correct data strip in $\mathbb{G}$, then $\Phi(\mathbb{G}) = 0$.*

Similar to Theorem 3.2, the correctness of Theorem 3.3 can also be easily validated using the approach given in Fig. 4.

We here do not repeat it but just give an example to illustrate when we can use Theorem 3.3 to determine the consistency of strip groups.

Example. We consider a strip group $\mathbb{G} = \{D_0, D_1, \ldots, D_7, D_8\langle stale\rangle, P_0\langle stale \prec \Re\rangle, P_1\langle stale \prec \Re\rangle\}$ taken from an erasure-coded storage system with $k = 10$ and $m = 6$. We wonder when $\mathbb{G}$ is consistent. According to Theorem 3.3, when $D_8 \in \Re$ and $\Re \cap \{D_0, D_1, \ldots, D_7\} = \emptyset$, we can deduce $\Phi(\mathbb{G}) = 0$. For example, when $\Re = \{D_8\}$ or $\Re = \{D_8, D_9\}$, $\Phi(\mathbb{G}) = 0$.

In Theorem 3.2 and Theorem 3.3, we present two special kinds of strip groups that are always consistent. In fact, besides these two kinds of strip groups, we have also checked the consistency of other kinds of strip groups using the approach given in Fig. 4. Unfortunately, all other kinds of strip groups do not have such good property and can rarely be determined to be consistent using the approach given in Fig. 4. Their probability of being determined to be consistent is defined as *consistency probability* (denoted by $Pr_{consistency}$). Then, you may be interested in how to estimate their consistency probability.

For a strip group $\mathbb{G}$, suppose $\widetilde{\mathbb{G}}$ is its maximal subgroup that is always consistent. Then, for each strip contained in $\mathbb{G} \setminus \widetilde{\mathbb{G}}$, we regenerate it from $\widetilde{\mathbb{G}}$. Suppose there are $w \times l$ bits in each strip, where w is the number of elements in each strip, and l is the number of bits in each element. Then, the size of a strip's value space is $2^{w\times l}$. Thus, for each strip contained in $\mathbb{G} \setminus \widetilde{\mathbb{G}}$, the probability of its regenerated value being equal to its original value in $\mathbb{G}$ is $\frac{1}{2^{w\times l}}$. Let $\eta = |\mathbb{G}| - |\widetilde{\mathbb{G}}|$. We can deduce that the consistency probability of $\mathbb{G}$ is

$$Pr_{consistency}(\mathbb{G}) = \frac{1}{2^{w\times l\times \eta}}. \quad (3)$$

We consider an erasure-coded storage system with the configuration parameters $w = 4$ and $l = 512$ B (i.e., the size of a sector). In this system, for a strip group not belonging to the two kinds of strip groups presented in Theorem 3.2 and Theorem 3.3, we can deduce that its consistency probability can be guaranteed to be not higher than $\frac{1}{2^{16384}}$ (i.e., $\frac{1}{2^{4\times(512\times 8)\times 1}}$) according to Eq. (3). Thus, its consistency probability is very negligible in practice.

From the above discussion, we can reasonably make an assumption as follows:

Assumption 3.1. *For a strip group $\mathbb{G}$ containing more than k strips, some of which are silently corrupted strips, if it does not belong to the two kinds of strip groups presented in Theorem 3.2 and Theorem 3.3, it is always assumed to be inconsistent in practical systems.*

From what have been discussed in this section, we finally make an observation as follows:

Observation 3.1. *For a strip group $\mathbb{G}$ containing more than k strips, if it is determined to be consistent using the approach given in Fig. 4, it should be in one of the following three states:*

1. *All its strips are correct strips;*
2. *All its data strips are correct data strips, and all its parity strips are stale parity strips caused by the writes to the same set of data strips, which does not contain any data strip in $\mathbb{G}$; or*
3. *The set of its data strips consists of correct data strips and stale data strips, and all its parity strips are stale parity strips caused by the writes to the same set of data strips, which contains all the stale data strips in $\mathbb{G}$ but does not contain any correct data strip in $\mathbb{G}$.*

The above observation is very significant because it reveals that one approach to determine the correctness of a combination of k strips is to determine the consistency of a corresponding larger-size strip group that properly contains the combination. In the following section, we will develop a new data reconstruction method based on this observation.

4 Data Reconstruction in the Presence of Silent Data Corruptions

Having understood the impact of silent data corruptions on the consistency of strip groups, we now study how to efficiently reconstruct lost data in the presence of silent data corruptions in this section. We first propose an efficient adaptive data reconstruction method in the first subsection. Then, we discuss when we can correctly reconstruct lost data using our proposed method in the second subsection.

4.1 An Efficient Adaptive Data Reconstruction Method

In this subsection, we will develop an efficient adaptive method for data reconstruction in the presence of silent data corruptions.

Our data reconstruction method is developed under the general framework of *maximum-likelihood decoding* [49]. As shown in Observation 3.1, for a strip group containing more than k strips, if it is determined to be consistent using the approach given in Fig. 4, all its parity strips should be in one of the following two states: correct or stale. Luckily, as we know, in practical systems, the probability of a parity strip being correct is much higher than that of the parity strip being stale [22]. Thus, in a stripe, the number of correct parity strips is often much larger than that of stale parity strips. Then, during data reconstruction, to avoid encountering a consistent strip group that contains stale parity strips before finding a consistent strip group that consists of correct strips, we first start with the strip groups that contain all the remaining parity strips. During this process, if no consistent strip group can be found, we continue to determine the consistency of strip groups that contain fewer parity strips.

We now consider a stripe in which f strips are lost or are identified to be corrupted, and there also exist some silently corrupted strips. Suppose there are k^* data strips and m^* parity strips contained in the set of the remaining strips, where $k^* + m^* = k + m - f$. We use $\mathbb{D}^*$ and $\mathbb{P}^*$ to denote the corresponding data and parity sets, respectively, where $|\mathbb{D}^*| = k^*$ and $|\mathbb{P}^*| = m^*$. It is clear that $\mathbb{D}^* \subseteq \mathbb{D}$ and $\mathbb{P}^* \subseteq \mathbb{P}$.

To correctly reconstruct lost data in the presence of silent data corruptions, we now present an adaptive data reconstruction algorithm in Fig. 5. It should be noted that the precondition to perform this algorithm is $f < m$. In this algorithm, if TRUE is returned, all the lost strips have been correctly reconstructed, and all the silently corrupted strips have also been detected and corrected; and if FALSE is returned, there exists no consistent strip group containing

An Adaptive Data Reconstruction Algorithm

(0) We use θ to denote the number of parity strips contained in a strip group. Initially, set $\theta \leftarrow m^*$.

(1) For each combination $\mathcal{P}$ of θ parity strips chosen from $\mathbb{P}^*$, determine whether there exists a consistent strip group containing more than k strips in the set of strip groups containing $\mathcal{P}$.

(2) If there exists one such consistent strip group, first reconstruct lost strips from this strip group, then detect and correct silently corrupted strips according to this strip group, and finally **return** TRUE; else, continue to the next step.

(3) If $\theta > k - k^* + 1$, set $\theta \leftarrow \theta - 1$ and go back to Step (1); else, **return** FALSE.

Fig. 5. An adaptive data reconstruction algorithm.

more than k strips, and the error scenario is then considered to be beyond the ability of the adopted k-of-$(k+m)$ erasure code to tolerate errors (this will be discussed in detail in the next subsection). In practical systems, when FALSE is returned, an exception event will be reported to the upper-level file system to invoke higher-level protection mechanisms.

We then discuss the implementation details of the algorithm presented in Fig. 5. Here, in Step (2), to detect and correct silently corrupted strips, we re-encode the stripe after the reconstruction operation and then compare the regenerated values with the original values in the stripe. In addition, in Step (1), one important question is how to efficiently determine whether there exists a consistent strip group containing more than k strips in the set of strip groups containing $\mathcal{P}$. However, we notice that if we use the approach given in Fig. 4 to determine the consistency of each strip group, respectively, many unnecessary reconstruction operations will be involved in this process. Thus, to improve the performance of this process, we need to develop a more efficient approach.

For given $\mathbb{D}^*$ and $\mathcal{P}$, Fig. 6 presents an efficient approach to determine whether there exists a consistent strip group with a size $(k+1)$ in the set of strip groups containing $\mathcal{P}$. Here, we only consider the consistent strip groups with a size $(k+1)$ for the following two reasons:

1. A consistent strip group with a size $(k+1)$ is enough for data reconstruction; and
2. If there exists no consistent strip group with a size $(k+1)$, there will also exist no consistent strip group with a larger size (this can be easily deduced according to the property that $\Phi(\mathbb{G}_1) = 1 \Rightarrow \forall \mathbb{G}_2 \supseteq \mathbb{G}_1 : \Phi(\mathbb{G}_2) = 1$).

In the approach given in Fig. 6, if a strip group is returned, a consistent strip group with a size $(k+1)$ has been found, and then, we can immediately reconstruct all the lost strips and detect and correct all the silently corrupted strips using the consistent strip group; and if NULL is returned, there exists no consistent strip group containing more than k strips in the set of strip groups containing $\mathcal{P}$. We can easily deduce that the approach given in Fig. 6 involves only one reconstruction operation in the best case and $\binom{k^*-1}{k-\theta}$ reconstruction operations in the worst case, where $\theta = |\mathcal{P}|$. Its exact computational complexity depends on the error scenario. However, we believe that it is optimal in terms of computational complexity.

Search-a-Consistent-Strip-Group($\mathbb{D}^*$, $\mathcal{P}$){

(0) Suppose $\mathbb{D}^* = \{D_0^*, D_1^*, \ldots, D_{k^*-1}^*\}$ and $|\mathcal{P}| = \theta$. Let $\tau = k - \theta$. For a combination $\mathcal{D}$ of τ data strips chosen from $\mathbb{D}^*$, we use i_0, i_1, …, $i_{\tau-2}$, and $i_{\tau-1}$ to denote the subscripts of its τ data strips, respectively. Initially, set $i_0 \leftarrow 0, i_1 \leftarrow 1, \ldots, i_{\tau-2} \leftarrow \tau-2$, and $i_{\tau-1} \leftarrow \tau-1$.

(1) Perform a reconstruction operation using $\mathcal{D} \cup \mathcal{P}$.

(2) For each data strip with a subscript larger than $i_{\tau-1}$ in $\mathbb{D}^*$, compare its regenerated value with its original value in $\mathbb{D}^*$. If there exists a data strip D_ξ^* whose regenerated value is equal to its original value in $\mathbb{D}^*$, **return** the consistent strip group $\mathcal{D} \cup \mathcal{P} \cup \{D_\xi^*\}$; and if there is no such data strip, continue to the next step.

(3) If $i_0 = k^* - 1 - \tau$, **return** NULL; else, increase the subscript vector $(i_0, i_1, \ldots, i_{\tau-1})$ by one in lexicographic order and go back to Step (1).

}

Fig. 6. An efficient approach to determine whether there exists a consistent strip group with a size $(k+1)$ in the set of strip groups containing $\mathcal{P}$.

In this section, we have proposed an efficient adaptive method for data reconstruction in the presence of silent data corruptions. Unlike the existing data reconstruction methods, such as the matrix methods proposed in [26], our data reconstruction method here can cope with both lost strips and silently corrupted strips.

4.2 When Can We Correctly Reconstruct Lost Data?

In this subsection, we will discuss when we can correctly reconstruct lost data using our data reconstruction method proposed in Section 4.1.

4.2.1 The Conditions to Correctly Reconstruct Lost Data

Basically, in a k-of-$(k+m)$ erasure-coded storage system, when f strips are lost or are identified to be corrupted in a stripe, there should exist more than k correct strips in the set of the remaining strips so as to form a consistent strip group containing more than k strips. Suppose there exist r silently corrupted strips in the set of the remaining strips. Then, we should have $n - f - r \geq k + 1$. Since $n = k + m$, we can deduce a necessary condition that $f + r \leq m - 1$.

Now, under the condition that $f + r \leq m - 1$, we begin to discuss when we can correctly reconstruct lost data using our data reconstruction method proposed in Section 4.1. Meanwhile, to make our discussion more easily understood, we will give some example scenarios taken from an erasure-coded storage system with $k = 10$ and $m = 6$.

When some strips are lost or are identified to be corrupted in a stripe, it is clear that if the number of correct parity strips is larger than that of silently corrupted parity strips in the set of the remaining strips, our data reconstruction method can always correctly reconstruct lost data. Here, the "if" condition is often looser than the well-known condition that $f + 2r \leq m$ in coding theory [27]. This advantage results from the probabilistic characteristics of the consistency of strip groups exploited in Section 3.2.

Example. In an erasure-coded storage system with $k = 10$ and $m = 6$, we consider a stripe in which one data strip and one parity strip are lost, and one data strip and two parity strips are silently corrupted. In this stripe, although $f + 2r \leq m$ is not satisfied, since $f + r \leq m - 1$ is satisfied, and the number of correct parity strips

is larger than that of silently corrupted parity strips in the set of the remaining strips, our data reconstruction method can still correctly recover all the faulty strips.

Now, let us consider the more complex case where the number of correct parity strips is not larger than that of silently corrupted parity strips in the set of the remaining strips. According to Observation 3.1, it is clear that if the set of correct parity strips is larger than any set of stale parity strips caused by the writes to the same set of data strips, our data reconstruction method can always correctly reconstruct lost data. Here, it should be noted that even when the "if" condition is not satisfied, our data reconstruction method can still correctly reconstruct lost data as long as no consistent strip group containing more than k strips can be formed by stale parity strips.

Example. In an erasure-coded storage system with $k = 10$ and $m = 6$, we consider a stripe in which one data strip and one parity strip are lost, and three parity strips are stale parity strips caused by the writes to the same set $\Re$ that contains at least two remaining data strips. In this stripe, although the set of correct parity strips is smaller than the set of stale parity strips caused by the writes to $\Re$, since no consistent strip group containing more than k strips can be formed by the three stale parity strips, our data reconstruction method can still correctly recover all the faulty strips.

4.2.2 The Limitations of Our Data Reconstruction Method

Until now, we have discussed the conditions to correctly reconstruct lost data using our data reconstruction method proposed in Section 4.1. Then, someone may ask in what case we cannot correctly reconstruct lost data using our data reconstruction method. We will give the answer to this question in the following discussion.

When $f + r \geq m$, there exist not more than k correct strips. Then, in the set of strip groups containing more than k strips, there exists no consistent strip group that consists of correct strips. At this time, the corresponding error scenario is beyond the ability of the adopted k-of-$(k + m)$ erasure code to tolerate errors.

Then, under the condition that $f + r \leq m - 1$, we begin to discuss when we cannot correctly reconstruct lost data using our data reconstruction method.

When some strips are lost or are identified to be corrupted in a stripe, in the set of the remaining strips, if there exists a set of stale parity strips caused by the writes to the same set of data strips, which meets the following two conditions:

1. this set of stale parity strips is larger than the set of correct parity strips, and
2. a consistent strip group containing more than k strips can be formed by this set of stale parity strips,

our data reconstruction algorithm presented in Fig. 5 will then reconstruct lost data from the consistent strip group formed by this set of stale parity strips. Thus, some corrupt strips will be produced.

In the above two conditions, if the first condition is replaced by a new condition that the size of this set of stale parity strips is equal to that of the set of correct parity strips, our data reconstruction algorithm presented in Fig. 5 will possibly reconstruct lost data from a consistent strip group consisting of correct strips or from a consistent strip group formed by this set of stale parity strips. Then, the correctness of the data reconstruction process depends on the distribution of lost strips and silently corrupted strips in the stripe.

To make the above discussion more easily understood, we now give two example scenarios taken from an erasure-coded storage system with $k = 10$ and $m = 6$.

Example. We first consider a stripe in which one data strip is lost, and four parity strips are stale parity strips caused by the writes to the same set $\Re$ that contains not more than two remaining data strips. In this stripe, since the set of stale parity strips caused by the writes to $\Re$ is larger than the set of correct parity strips, and a consistent strip group containing more than k strips can be formed by the four stale parity strips caused by the writes to $\Re$, some corrupt strips will be produced by our data reconstruction algorithm presented in Fig. 5.

Example. We then consider another stripe in which one data strip is lost, and three parity strips are stale parity strips caused by the writes to the same set $\Re$ that contains not more than one remaining data strips. In this stripe, a consistent strip group containing more than k strips can also be formed by the three stale parity strips caused by the writes to $\Re$. However, the number of stale parity strips caused by the writes to $\Re$ here is equal to that of correct parity strips. Thus, our data reconstruction algorithm presented in Fig. 5 will possibly reconstruct lost data from a consistent strip group consisting of correct strips or from a consistent strip group formed by the three stale parity strips caused by the writes to $\Re$. The correctness of the data reconstruction process depends on the distribution of lost strips and silently corrupted strips in the stripe.

For the corrupt strips produced by our data reconstruction method, we notice that all of them are stale strips. An example is as follows:

Example. In an erasure-coded storage system with $k = 10$ and $m = 6$, we consider a stripe in which D_9 and P_5 are lost, and the set of the remaining strips is $\{D_0, D_1, \ldots, D_8, P_0, P_1, P_2\langle stale \prec \Re\rangle, P_3\langle stale \prec \Re\rangle, P_4\langle stale \prec \Re\rangle\}$ with $\Re = \{D_0, D_9\}$. It is clear that all the three stale parity strips are caused during the write processes of D_0 and D_9. If we reconstruct this stripe using our data reconstruction method proposed in Section 4.1, a set of strip values (i.e., $D_0\langle stale\rangle$, D_1, $\ldots$, D_8, $D_9\langle stale\rangle$, $P_0\langle stale \prec \Re\rangle$, $P_1\langle stale \prec \Re\rangle$, $P_2\langle stale \prec \Re\rangle$, $P_3\langle stale \prec \Re\rangle$, $P_4\langle stale \prec \Re\rangle$, and $P_5\langle stale \prec \Re\rangle$) will be produced from the consistent strip group $\mathbb{G} = \{D_1, \ldots, D_8, P_2\langle stale \prec \Re\rangle, P_3\langle stale \prec \Re\rangle, P_4\langle stale \prec \Re\rangle\}$. Here, for the three strips (i.e., D_0, P_0, and P_1) not contained in $\mathbb{G}$, since their original values are not consistent with the values regenerated from $\mathbb{G}$, they are then mistakenly regarded as silently corrupted strips in the process of data reconstruction and are thus replaced by the regenerated values.

From the above example, we can see that all the corrupt strips produced by our data reconstruction method are stale strips. In fact, if more than half of parity strips fail to be

updated during the write processes of a set of data strips, there is no way to guarantee not producing stale strips during the subsequent data reconstruction process unless an additional version mirroring technique [23] is used. This is easily understood according to the majority principle. Then, the corresponding error scenario is considered to be beyond the ability of an erasure code to tolerate errors.

From what have been discussed in this subsection, we can see that our data reconstruction method proposed in Section 4.1 can fully exploit the fault-tolerance potential of an erasure code. Our data reconstruction method cannot correctly reconstruct lost data only in the case where the error scenario is beyond the theoretical ability of the adopted erasure code to tolerate errors.

5 The Performance Impact of Our Data Reconstruction Method

When our data reconstruction method proposed in Section 4.1 is adopted in erasure-coded storage systems, one important concern is its performance impact. In this section, we will show that the overall performance impact of our data reconstruction method is negligible in practical systems.

As we know, our data reconstruction method not only reconstructs lost strips in a stripe but also detects and corrects silently corrupted strips in the same stripe. Consequently, all the strips in the stripe should be accessed during data reconstruction. Here, if we use a traditional data reconstruction method (that does not consider the issues of silent data corruptions) to perform the similar functions as our data reconstruction method (though the correctness of the data reconstruction process then cannot be guaranteed), all the strips in the stripe should also be accessed during data reconstruction. Thus, our data reconstruction method does not cause additional I/O overhead. Then, we only need to investigate the additional computational overhead caused by our data reconstruction method in this section.

In our data reconstruction method, the one and only one kind of complex and time-consuming operation is the reconstruction operation. Thus, we will use the number of reconstruction operations (denoted by N_{op}) involved in the data reconstruction process as an evaluation metric in the following discussion.

5.1 The Computational Complexity

Before considering the overall performance impact of our data reconstruction method in practical systems, we first estimate the computational complexity of our data reconstruction method.

In a k-of-$(k+m)$ erasure-coded storage system, we consider a stripe in which f strips are lost or are identified to be corrupted, and there also exist r silently corrupted strips in the set of the remaining strips. Suppose there are k^* data strips and m^* parity strips contained in the set of the remaining strips, where $k^* + m^* = k + m - f$.

When we use our data reconstruction method to reconstruct the stripe, it is clear that $N_{op} = 1$ in the case where $k^* + m^* = k + 1$. However, the case where $k^* + m^* > k + 1$ is much more complex. We discuss it as follows:

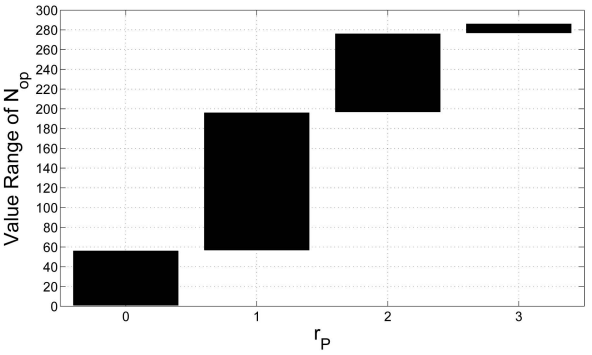

Fig. 7. The value range of N_{op} for different values of r_P in the case where $k^* = 9$ and $m^* = 5$ in an erasure-coded storage system with $k = 10$ and $m = 6$.

In the case where $k^* + m^* > k + 1$, when $r = 0$, it is clear that $N_{op} = 1$. When $r > 0$, suppose there are r_D data strips and r_P parity strips contained in the set of silently corrupted strips, where $r_D + r_P = r$. Then, under the conditions to correctly reconstruct lost data given in Section 4.2.1, the value range of N_{op} can be estimated as follows:

1. When $r_P = 0$,

$$N_{op} \in \left[1, \binom{k^* - 1}{k - m^*}\right];$$

2. When $1 \le r_P \le k^* + m^* - k - 1$,

$$N_{op}(r_P) \in \left[\sum_{i=0}^{r_P - 1} \binom{m^*}{i} \binom{k^* - 1}{k - (m^* - i)} + 1, \sum_{i=0}^{r_P} \binom{m^*}{i} \binom{k^* - 1}{k - (m^* - i)}\right].$$

Remark. According to the condition that $f + r \le m - 1$ given in Section 4.2.1, we can deduce that $r_P \le k^* + m^* - k - 1$. Thus, we here only consider the case where $r_P \in [0, k^* + m^* - k - 1]$.

To further understand the estimated value range of N_{op}, we give an example as follows:

Example. In an erasure-coded storage system with $k = 10$ and $m = 6$, we consider a stripe in which one data strip and one parity strip are lost (i.e., the case where $k^* = 9$ and $m^* = 5$). Under the conditions to correctly reconstruct lost data given in Section 4.2.1, the value range of N_{op} is shown in Fig. 7. From this figure, we can see that when we use our data reconstruction method to reconstruct a stripe, the number of reconstruction operations involved in the data reconstruction process increases with the number of silently corrupted parity strips. This characteristic is decided by the structure of our data reconstruction algorithm presented in Fig. 5. However, the exact value of the number of reconstruction operations depends on the distribution of lost strips and silently corrupted strips in the stripe.

Finally, it should be noted that in the case where FALSE is returned by our data reconstruction algorithm presented in Fig. 5, $N_{op} = \sum_{i=0}^{k^*+m^*-k-1} \binom{m^*}{i} \binom{k^*-1}{k-(m^*-i)}$, which is the upper bound of N_{op}.

5.2 The Overall Performance Impact

Now, we employ our data reconstruction method in practical systems. We will study its overall performance impact using a probabilistic method in this subsection.

According to the values of r_D and r_P, N_{op} is roughly estimated as follows:

1. When $r_P = 0$ and $r_D = 0$ (i.e., $r = 0$),
$$N_{op} = 1;$$
2. When $r_P = 0$ and $r_D > 0$,
$$N_{op} \leq \binom{k^* - 1}{k - m^*};$$
3. When $1 \leq r_P \leq m^*$,
$$N_{op} \leq \sum_{i=0}^{k^*+m^*-k-1} \binom{m^*}{i} \binom{k^* - 1}{k - (m^* - i)}.$$

Suppose the probability of a strip being silently corrupted is p. Then, for a stripe, the corresponding probability (denoted by Pr) for different values of r_D and r_P can be estimated as follows:

1. For $r_P = 0$ and $r_D = 0$ (i.e., $r = 0$),
$$Pr \approx (1-p)^{k^*+m^*};$$
2. For $r_P = 0$ and $r_D > 0$,
$$Pr \approx (1-p)^{m^*} \times [1 - (1-p)^{k^*}];$$
3. For $1 \leq r_P \leq m^*$,
$$Pr(r_P) \approx (1-p)^{m^*-r_P} \times p^{r_P}.$$

Using the results obtained above, we can estimate the average value of the number of reconstruction operations (denoted by $\overline{N_{op}}$) involved in the data reconstruction process. Then, according to the value of $\overline{N_{op}}$, we can estimate the overall performance impact of our data reconstruction method in practical systems.

Here, the lower bound of $\overline{N_{op}}$ is 1, which is also equal to the number of reconstruction operations involved in a traditional data reconstruction method (that does not consider the issues of silent data corruptions). A smaller $\overline{N_{op}}$ means a smaller overall performance impact. If $\overline{N_{op}}$ can be estimated to very approach to 1, the overall performance impact can then be regarded to be negligible in practical systems.

In the following discussion, we will give some examples, taken from an erasure-coded storage system with the configuration parameters $k = 10$, $m = 6$, $w = 4$, and $l = 512$ B (i.e., the size of a sector), to show that the overall performance impact of our data reconstruction method is negligible in practical systems.

TABLE 3
The Upper Bound of $\overline{N_{op}}$ Estimated for Different Erasure Scenarios in an Example System with $k = 10$, $m = 6$, $w = 4$, and $l = 512$ B

Erasure scenario			The upper bound of $\overline{N_{op}}$	
f	k^*	m^*	Nearline disks	Enterprise class disks
1	10	5	$1 + 3.6798 \times 10^{-8}$	$1 + 3.6798 \times 10^{-9}$
	9	6	$1 + 2.6477 \times 10^{-8}$	$1 + 2.6476 \times 10^{-9}$
2	10	4	$1 + 1.8219 \times 10^{-8}$	$1 + 1.8219 \times 10^{-9}$
	9	5	$1 + 1.2714 \times 10^{-8}$	$1 + 1.2714 \times 10^{-9}$
	8	6	$1 + 9.0440 \times 10^{-9}$	$1 + 9.0439 \times 10^{-10}$
3	10	3	$1 + 6.7666 \times 10^{-9}$	$1 + 6.7664 \times 10^{-10}$
	9	4	$1 + 4.9971 \times 10^{-9}$	$1 + 4.9970 \times 10^{-10}$
	8	5	$1 + 3.6209 \times 10^{-9}$	$1 + 3.6208 \times 10^{-10}$
	7	6	$1 + 2.5887 \times 10^{-9}$	$1 + 2.5886 \times 10^{-10}$
4	10	2	$1 + 1.4582 \times 10^{-9}$	$1 + 1.4581 \times 10^{-10}$
	9	3	$1 + 1.1633 \times 10^{-9}$	$1 + 1.1632 \times 10^{-10}$
	8	4	$1 + 9.0112 \times 10^{-10}$	$1 + 9.0110 \times 10^{-11}$
	7	5	$1 + 6.7174 \times 10^{-10}$	$1 + 6.7173 \times 10^{-11}$
	6	6	$1 + 4.7514 \times 10^{-10}$	$1 + 4.7513 \times 10^{-11}$
5	$k^* + m^* = 11$		1	1

Here, since the probability of producing silent data corruptions is about an order of magnitude smaller than that for unrecoverable sector errors [22], [23], referring to the unrecoverable bit-error probability estimated in [4] and [5], the probability of a bit being silently corrupted (denoted by p_{bit}) can be estimated to be 10^{-15} for nearline (SATA) diks and 10^{-16} for enterprise class (Fibre Channel) disks. Then, the equivalent probability of a strip being silently corrupted is $p \approx p_{bit} \times w \times l$, which is 1.6384×10^{-11} in the case of nearline diks and 1.6384×10^{-12} in the case of enterprise class disks.

In Table 3, we estimate the upper bound of $\overline{N_{op}}$ for different erasure scenarios in the cases of both nearline and enterprise class disks. Here, we only consider the erasure scenarios where $f < m$, which is the precondition to use our data reconstruction method.

From Table 3, we can see that $\overline{N_{op}}$ is equal to 1 in the case where $k^* + m^* = k + 1$ and is very close to 1 in the case where $k^* + m^* > k + 1$. Thus, we can make a conclusion that the overall performance impact of our data reconstruction method is negligible in practical systems.

6 A Comparison of Techniques for Coping with Silent Data Corruptions

In erasure-coded storage systems, according to the design principle of an abstraction layer framework, the RAID-like layer is in the best position to correct silent data corruptions it detects and locates when such corrections are theoretically and practically possible. Thus, in this section, we consider the following two kinds of techniques that can be integrated into the RAID-like layer: the traditional metadata techniques (such as the family of four metadata techniques proposed in [24]) and the new technique based on our data reconstruction method. Before comparing these two kinds of techniques, we first give a brief overview of metadata techniques.

6.1 Metadata Techniques—A Brief Overview

Metadata techniques colocate some forms of metadata (that may contain checksums, version numbers, or other information of the data) along with data/parity strips. They are often used in an ad hoc manner, i.e., choosing different forms of metadata for protection against different types of silent data corruptions. As mentioned in Section 2.2, there exist two categories of silent data corruptions, i.e., Undetected Disk Errors (UDEs) and Strip Update Errors (SUEs), in erasure-coded storage systems. We discuss how to use metadata to cope with them as follows:

To protect against UDEs, we use the metadata that contains checksums to cross-check each strip. The metadata for cross-checking is stored separately from the strip it checks and thus has to be updated by a separate write. Here, someone may consider to use the metadata that contains simpler version numbers to self-check each strip. The metadata for self-checking is stored colocated with the strip it checks and thus can be updated by the same write to the strip. Although the metadata for self-checking involves much less additional I/O overhead than the metadata for cross-checking, it cannot detect silent data corruptions caused by lost writes. In contrast, the metadata for cross-checking can detect all kinds of silent data corruptions. In addition, to cross-check a strip, there should be at least two metadata copies stored separately from each other in the corresponding stripe. If there is only one metadata copy, then when the metadata copy and the strip it cross-checks are detected to be inconsistent, there is no direct way to determine which one is silently corrupted. However, if there are more than one metadata copy, the problem can then be solved according to the majority principle. Furthermore, the number of metadata copies should be carefully decided. Suppose there are α metadata copies for cross-checking a strip in a metadata solution. Then, when more than $\alpha - 2$ strips are lost or are identified to be corrupted in a stripe, the metadata solution cannot work. Generally, in a k-of-$(k+m)$ erasure-coded storage system, the number of metadata copies for cross-checking a strip can be set to be $m+1$, and then, f lost strips together with r silently corrupted strips caused by UDEs can be tolerated in a stripe as long as $f + r \leq m$.

For SUEs, metadata techniques cannot always work. The SUEs caused due to the loss of update operations can be detected using a metadata technique called as *version mirroring* [23]. However, for other kinds of SUEs (such as the SUEs caused due to processor miscalculations), there is no effective metadata technique that can detect them.

6.2 Making the Best Choice to Cope with Silent Data Corruptions

There are several evident differences between the traditional metadata techniques and the new technique based on our data reconstruction method. We list these differences as follows:

- The former cannot detect SUEs except those caused due to the loss of update operations, while the latter can cope with all kinds of silent data corruptions.
- The necessary condition for the former to tolerate errors is $f + r \leq m$ (resulting in at least $m = 1$ for $r = 1$), while that for the latter is $f + r \leq m - 1$ (resulting in at least $m = 2$ for $r = 1$).
- The former requires additional storage overhead to store metadata, while the latter does not need additional disk space. Consequently, the former involves additional I/O overhead during write operations, while the latter does not affect write performance.
- When a read operation on a strip is validated, the former should read only two metadata copies from two other strips in the same stripe, while the latter should read at least k other strips in the same stripe.

We can see that these two kinds of techniques make different tradeoffs among cost, performance, and effectiveness.

We then begin to discuss how to make the best choice for different requirements on cost, performance, and effectiveness. As we know, the additional storage cost involved by metadata techniques is often negligible. Thus, we will only focus our attention on performance and effectiveness in the following discussion.

In erasure-coded storage systems, validation can be used on every read operation to guarantee the integrity of the data to the host or periodically as an integrity control check. Validation on every read operation provides a very high level of data integrity but can cause a performance penalty on each read operation. To minimize the performance penalty on the read operation, metadata techniques can be used here as a better choice to cope with silent data corruptions for its advantage that it can involve less additional I/O overhead during the validation. Even so, the performance penalty on the read operation is still very high. Thus, validation on every read operation often cannot meet users' requirement on performance. Then, some system designers propose to use periodic validation, which is always integrated into periodic disk scrubbing [50], [51]. Here, since disk scrubbing itself has to read all disk data, validation can involve only a little additional computational overhead, which is negligible compared with I/O overhead. Periodic validation provides a relatively lower level of data integrity. The level of data integrity depends on the frequency of the validation. Luckily, periodic validation often can provide a high enough level of data integrity that is expected by system users. When periodic validation is used in an erasure-coded storage system, the technique based on our data reconstruction method is a better choice to cope with silent data corruptions for its advantage that it does not involve any additional I/O overhead during write operations.

7 Related Work

In recent years, silent data corruptions have attracted much attention. Two studies, one at CERN [21] and one using data from NetApp [22], analyze the occurrence and characteristics of silent data corruptions in state-of-the-art storage systems. Their results show that there exist a significant number of silent data corruptions in storage systems. This reinforces the need for effective techniques to cope with silent data corruptions.

To address the issues caused by silent data corruptions, a significant amount of work has recently been undertaken in

both the academic and industry research. We introduce them as follows:

The paper by Schwarz et al. [50] uses signatures (which have some of the characteristics of hashes, a type of checksum) together with disk scrubbing to detect some forms of silent data corruptions. However, the focus of this paper is on how the scheduling of disk scrubbing affects overall system reliability, and thus, the authors do not directly address the issue of correcting silent data corruptions.

Prabhakaran et al. [19] study how file systems can address disk errors. They develop an IRON taxonomy for detection and recovery techniques, but these techniques are mostly viewed from the perspective of a file system. In comparison, our technique proposed for coping with silent data corruptions lies in the RAID-like layer, which is the layer closest to disk drives in erasure-coded storage systems. According to the design principle of an abstraction layer framework, the RAID-like layer is in the best position to correct silent data corruptions it detects and locates when such corrections are theoretically and practically possible.

Sundaram [20] gives a description of how NetApp protects against various kinds of disk errors. They use colocated checksums per 4-KB block and periodic disk scrubbing to address the problem of silent data corruptions caused by firmware bugs. In addition, since checksums can detect only local silent data corruptions, they adopt a novel technique, which involves storing logical identifiers (e.g., logical address obtained from the upper-level layer) with the data and writing new versions of the data on new physical locations, to solve the problem of lost writes and misdirected writes. Unfortunately, as shown in [23], this combination of protection mechanisms still cannot prevent parity pollution caused due to disk scrubbing and user writes in erasure-coded storage systems because logical identifiers can be verified only by user reads. This again demonstrates that the optimal solution to the problem of silent data corruptions lies in the RAID-like layer closest to disk drives in erasure-coded storage systems.

Existing work closest to our research here is the research by Hafner et al. [24]. They explore a family of four metadata techniques to the problem of Undetected Disk Errors (UDEs), which can be integrated into the RAID-like layer in erasure-coded storage systems. However, these techniques have a remarkable defect: they fail to consider the UDEs that may occur on metadata and parity. Besides, these techniques cannot be used to cope with another category of silent data corruptions mentioned in Section 2.2, i.e., Strip Update Errors (SUEs). In comparison, our technique in this paper can cope with all kinds of silent data corruptions, including both UDEs and SUEs. Furthermore, as mentioned in Section 6.2, our technique is a better choice to cope with silent data corruptions when periodic validation is used in an erasure-coded storage system.

The work of Krioukov et al. [23] makes a detailed analysis of the exact protection offered by each type of metadata like checksums, physical and logical identity, version numbers, etc. Krioukov et al. use model checking to evaluate whether common protection techniques used in RAID subsystems are sufficient in light of the increasingly complex failure modes of modern disk drives. They also identify a scheme, including version mirroring, physical and logical identity, block checksums, and RAID, to eliminate data loss or corruption for a given realistic range of disk errors. However, this scheme requires logical identity that should be obtained from the upper-level layer and thus conflicts with the design principle of an abstraction layer framework. In comparison, our main effort in this paper is a data reconstruction method that can effectively prevent silent data corruptions from propagating. Moreover, our data reconstruction method here can be used to cope with all kinds of silent data corruptions, including both UDEs and SUEs.

Finally, in the literature of distributed erasure-coded storage systems, there is also some related work that deserves mention. To verify whether a given strip indeed corresponds to a specific original block, *cross-checksums* [52], [53] (or *fingerprinted cross-checksums* [54]) are often used in these systems. However, their main purpose is to achieve a high level of security in an untrusty environment in which Byzantine clients [12], [13] may exist. Thus, it would involve very extravagant overhead if we use cross-checksums (or fingerprinted cross-checksums) to cope with silent data corruptions in the disk array subsystems of erasure-coded storage systems.

8 Conclusions

In this paper, we have carried out a thorough study on how to prevent silent data corruptions from propagating during data reconstruction in the context of erasure-coded storage systems. We first showed how silent data corruptions can propagate during data reconstruction and then proposed to prevent the propagation of silent data corruptions during data reconstruction by exploiting the error-correcting ability of erasure codes. To develop a data reconstruction method that can prevent silent data corruptions from propagating, we introduced a concept of the *consistency* of a strip group and then studied the impact of silent data corruptions on the consistency of strip groups. Based on the conclusions obtained from the study, we developed an efficient adaptive data reconstruction method that can cope with both lost strips and silently corrupted strips. We then discussed when we can correctly reconstruct lost data using our new data reconstruction method. Our discussion shows that when using our data reconstruction method, the actual ability of an erasure code to tolerate errors is often much beyond the well-known upper bound that $f + 2r \leq m$ in coding theory [27]. We also studied the overall performance impact of our data reconstruction method in practical systems using a probabilistic method. Our results show that the overall performance impact of our data reconstruction method is negligible in practical systems. Finally, we made a comparison of techniques for coping with silent data corruptions in erasure-coded storage systems. The comparison shows that the technique based on our data reconstruction method is a better choice to cope with silent data corruptions when periodic validation is used in an erasure-coded storage system.

Acknowledgments

The authors would like to thank Professor Qing (Ken) Yang in the Department of Electrical, Computer, and Biomedical Engineering at the University of Rhode Island for his great

help in preparing the manuscript of this paper. The authors are also grateful to the anonymous reviewers and the associate editor Professor Cristiana Bolchini for their constructive and valuable comments that helped in improving this paper. This work was supported by the National Science Fund for Distinguished Young Scholars of China (Grant No. 60925006) and the National High Technology Joint Research Program of China (Grant No. 2009AA01A403).

REFERENCES

[1] E. Pinheiro, W.D. Weber, and L.A. Barroso, "Failure Trends in a Large Disk Drive Population," *Proc. File and Storage Technologies (FAST '07),* Feb. 2007.

[2] B. Schroeder and G.A. Gibson, "Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?" *Proc. File and Storage Technologies (FAST '07),* Feb. 2007.

[3] L.N. Bairavasundaram, G.R. Goodson, S. Pasupathy, and J. Schindler, "An Analysis of Latent Sector Errors in Disk Drives," *Proc. SIGMETRICS '07,* Jun. 2007.

[4] A. Dholakia, E. Eleftheriou, X.-Y. Hu, I. Iliadis, J. Menon, and K. Rao, "A New Intra-Disk Redundancy Scheme for High-Reliability RAID Storage Systems in the Presence of Unrecoverable Errors," *ACM Trans. Storage,* vol. 4, no. 1, pp. 1-42, May 2008.

[5] I. Iliadis, R. Haas, X.-Y. Hu, and E. Eleftheriou, "Disk Scrubbing Versus Intra-Disk Redundancy for High-Reliability RAID Storage Systems," *Proc. SIGMETRICS '08,* Jun. 2008.

[6] J.S. Plank, "Erasure Codes for Storage Applications," Tutorial Slides, *Fourth USENIX Conf. File and Storage Technologies (FAST'05),* Dec. 2005.

[7] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, and D.A. Patterson, "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys,* vol. 26, no. 2, pp. 145-185, Jun. 1994.

[8] C. Carlane and A. Osuna, "IBM System Storage *N* Series Implementation of RAID Double Parity for Data Protection," IBM Redpaper REDP-4169-00, http://www.redbooks.ibm.com/redpapers/pdfs/redp4169.pdf, Apr. 2006.

[9] 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," *Proc. Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '00),* Nov. 2000.

[10] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: Highly Durable, Decentralized Storage Despite Massive Correlated Failures," *Proc. Second Symp. Networked Systems Design and Implementation (NSDI '05),* May 2005.

[11] S. Frolund, A. Merchant, Y. Saito, S. Spence, and A. Veitch, "A Decentralized Algorithm for Erasure-Coded Virtual Disks," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '04),* Jun. 2004.

[12] G.R. Goodson, J.J. Wylie, G.R. Ganger, and M.K. Reiter, "Efficient Byzantine-Tolerant Erasure-Coded Storage," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '04),* Jun. 2004.

[13] J. Hendricks, G.R. Ganger, and M.K. Reiter, "Low-Overhead Byzantine Fault-Tolerant Storage," *Proc. 21st ACM Symp. Operating Systems Principles (SOSP '07),* Oct. 2007.

[14] H. Xia and A.A. Chien, "RobuSTore: A Distributed Storage Architecture with Robust and High Performance," *Proc. 2007 ACM/IEEE Conf. Supercomputing (SC '07),* Nov. 2007.

[15] M.W. Storer, K.M. Greenan, E.L. Miller, and K. Voruganti, "Pergamum: Replacing Tape with Energy Efficient, Reliable, Disk-Based Archival Storage," *Proc. File and Storage Technologies (FAST '08),* Feb. 2008.

[16] Cleversafe, Inc. Cleversafe Dispersed Storage. Open source code distribution at http://www.cleversafe.org/downloads, 2010.

[17] Allmydata, Inc. Unlimited Online Backup, Storage, and Sharing, http://www.allmydata.com/, 2010.

[18] Permabit Technology Corporation. Disk Based Enterprise Archive, Data Archiving Solutions, http://www.permabit.com/, 2010.

[19] V. Prabhakaran, L.N. Bairavasundaram, N. Agrawal, H.S. Gunawi, A.C. Arpaci-Dusseau, and R.H. Arpaci-Dusseau, "IRON File Systems," *Proc. 21st ACM Symp. Operating Systems Principles (SOSP '05),* Oct. 2005.

[20] R. Sundaram, "The Private Lives of Disk Drives," Tech OnTap, NetApp, Inc., http://www.netapp.com/go/techontap/matl/sample/0206tot_resiliency.html, Feb. 2006.

[21] K. Péter, "Silent Corruptions," CERN, http://fuji.web.cern.ch/fuji/talk/2007/kelemen-2007-C5-Silent_Corruptions.pdf, Jun. 2007.

[22] L.N. Bairavasundaram, G.R. Goodson, B. Schroeder, A.C. Arpaci-Dusseau, and R.H. Arpaci-Dusseau, "An Analysis of Data Corruption in the Storage Stack," *Proc. File and Storage Technologies (FAST '08),* Feb. 2008.

[23] A. Krioukov, L.N. Bairavasundaram, G.R. Goodson, K. Srinivasan, R. Thelen, A.C. Arpaci-Dusseau, and R.H. Arpaci-Dusseau, "Parity Lost and Parity Regained," *Proc. File and Storage Technologies (FAST '08),* Feb. 2008.

[24] J.L. Hafner, V. Deenadhayalan, W. Belluomini, and K. Rao, "Undetected Disk Errors in RAID Arrays," *IBM J. Research and Development,* vol. 52, nos. 4/5, pp. 413-425, Jul./Sep. 2008.

[25] E.W.D. Rozier, W. Belluomini, V. Deenadhayalan, J. Hafner, K. Rao, and P. Zhou, "Evaluating the Impact of Undetected Disk Errors in RAID Systems," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '09),* Jun. 2009.

[26] J.L. Hafner, V. Deenadhayalan, K. Rao, and J.A. Tomlin, "Matrix Methods for Lost Data Reconstruction in Erasure Codes," *Proc. File and Storage Technologies (FAST '05),* Dec. 2005.

[27] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error-Correcting Codes.* Elsevier, 1977.

[28] I.S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *J. Soc. for Industrial and Applied Math.,* vol. 8, no. 2, pp. 300-304, Jun. 1960.

[29] R.R. Roth and A. Lempel, "On MDS Codes via Cauchy Matrices," *IEEE Trans. Information Theory,* vol. 35, no. 6, pp. 1314-1319, Nov. 1989.

[30] J. Blomer, M. Kalfane, M. Karpinski, R. Karp, M. Luby, and D. Zuckerman, "An XOR-Based Erasure-Resilient Coding Scheme," Technical Report TR-95-048, Int'l Computer Science Inst., Aug. 1995.

[31] J.S. Plank and L. Xu, "Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Network Storage Applications," *Proc. Fifth IEEE Int'l Symp. Network Computing and Applications (NCA '06),* Jul. 2006.

[32] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures," *IEEE Trans. Computers,* vol. 44, no. 2, pp. 192-202, Feb. 1995.

[33] M. Blaum, J. Bruck, and A. Vardy, "MDS Array Codes with Independent Parity Symbols," *IEEE Trans. Information Theory,* vol. 42, no. 2, pp. 529-542, Mar. 1996.

[34] M. Blaum, J. Brady, J. Bruck, J. Menon, and A. Vardy, "The EVENODD Code and its Generalization," *High Performance Mass Storage and Parallel I/O: Technologies and Applications,* J. Jin, T. Cortest, and R. Buyya, eds., chapter 14, pp. 187-208, IEEE and Wiley Press, 2001.

[35] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-Diagonal Parity for Double Disk Failure," *Proc. File and Storage Technologies (FAST '04),* Apr. 2004.

[36] C. Huang and L. Xu, "STAR: An Efficient Coding Scheme for Correcting Triple Storage Node Failures," *Proc. File and Storage Technologies (FAST '05),* Dec. 2005.

[37] G. Feng, R. Deng, F. Bao, and J. Shen, "New Efficient MDS Array Codes for RAID Part I: Reed-Solomon-Like Codes for Tolerating Three Disk Failures," *IEEE Trans. Computers,* vol. 54, no. 9, pp. 1071-1080, Sept. 2005.

[38] G. Feng, R. Deng, F. Bao, and J. Shen, "New Efficient MDS Array Codes for RAID Part II: Rabin-Like Codes for Tolerating Multiple (≥ 4) Disk Failures," *IEEE Trans. Computers,* vol. 54, no. 12, pp. 1473-1483, Dec. 2005.

[39] J.S. Plank, "The RAID-6 Liberation Codes," *Proc. File and Storage Technologies (FAST '08),* Feb. 2008.

[40] L. Xu and J. Bruck, "X-Code: MDS Array Codes with Optimal Encoding," *IEEE Trans. Information Theory,* vol. 45, no. 1, pp. 272-276, Jan. 1999.

[41] J.L. Hafner, "WEAVER Codes: High Fault Tolerant Erasure Codes for Storage Systems," *Proc. File and Storage Technologies (FAST '05),* Dec. 2005.

[42] J.L. Hafner, "HoVer Erasure Codes for Disk Arrays," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '06),* Jun. 2006.

[43] R.G. Gallager, *Low-Density Parity-Check Codes.* MIT Press, 1963.

[44] M.G. Luby, M. Mitzenmacher, A. Shokrollahi, and D.A. Spielman, "Efficient Erasure Correcting Codes," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 569-584, Feb. 2001.
[45] R.M. Tanner, "A Recursive Approach to Low-Complexity Codes," *IEEE Trans. Information Theory*, vol. 27, no. 5, pp. 533-547, Sept. 1981.
[46] J.S. Plank and M.G. Thomason, "A Practical Analysis of Low-Density Parity-Check Erasure Codes for Wide Area Storage Applications," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '04)*, Jun. 2004.
[47] J.S. Plank, R.L. Collins, A.L. Buchsbaum, and M.G. Thomason, "Small Parity-Check Erasure Codes—Exploration and Observations," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '05)*, Jun. 2005.
[48] M. Li, J. Shu, and W. Zheng, "GRID Codes: Strip-Based Erasure Codes with High Fault Tolerance for Storage Systems," *ACM Trans. Storage*, vol. 4, no. 4, Article 15, Jan. 2009.
[49] A. Vardy, "Algorithmic Complexity in Coding Theory and the Minimum Distance Problem," *Proc. 29th ACM Symp. Theory of Computing (STOC '97)*, May 1997.
[50] T.J.E. Schwarz, Q. Xin, E.L. Miller, and D.D.E. Long, "Disk Scrubbing in Large Archival Storage Systems," *Proc. 12th IEEE/ACM Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecomm. Systems (MASCOTS '04)*, Oct. 2004.
[51] A. Oprea and A. Juels, "A Clean-Slate Look at Disk Scrubbing," *Proc. File and Storage Technologies (FAST '10)*, Feb. 2010.
[52] L. Gong, "Securely Replicating Authentication Services," *Proc. Ninth Int'l Conf. Distributed Computing Systems (ICDCS '89)*, Jun. 1989.
[53] H. Krawczyk, "Distributed Fingerprints and Secure Information Dispersal," *Proc. 12th Ann. ACM Symp. Principles of Distributed Computing (PODC '93)*, Aug. 1993.
[54] J. Hendricks, G.R. Ganger, and M.K. Reiter, "Verifying Distributed Erasure-Coded Data," *Proc. 12th Ann. ACM Symp. Principles of Distributed Computing (PODC '07)*, Aug. 2007.

Mingqiang Li received the BSc degree in mathematics from the University of Electronic Science and Technology of China in 2006. He is now working toward the PhD degree in the Department of Computer Science and Technology at Tsinghua University. His current research interests include storage systems, coding theory, and distributed computing. He is a student member of the IEEE and the IEEE Computer Society.

Jiwu Shu received the PhD degree in computer science from Nanjing University in 1998, and finished the postdoctoral research at Tsinghua University in 2000. Since then, he has been working as a teacher at Tsinghua University. He is now a professor in the Department of Computer Science and Technology at Tsinghua University. His current research interests include storage systems and parallel computing. He is a member of the IEEE and the IEEE Computer Society.