2012 41st International Conference on Parallel Processing

Shifted Element Arrangement in Mirror Disk Arrays for High Data Availability during Reconstruction

Xianghong Luo, Jiwu Shu+ and Ying Zhao
Department of Computer Science and Technology
Tsinghua University
Beijing, China
luo-xh09@mails.tsinghua.edu.cn, shujw@tsinghua.edu.cn, yingz@tsinghua.edu.cn
+*Corresponding author: shujw@tsinghua.edu.cn*

***Abstract*—The mirror method and its variants are widely used in storage systems. However, they suffer from low data availability during on-line reconstruction because sequential reads are inevitable under the current element arrangements. In this paper, we propose a shifted element arrangement method in mirror disk arrays to provide high data availability during reconstruction, which can be used in both the mirror method and its variants. We design this novel element arrangement to minimize the maximum number of elements that are read out from one disk, thus minimize the number of read accesses during the reconstruction process. Theoretically, we prove that our shifted element arrangement improves data availability by a factor of n or (2n+1)/4 (n is the number of disks in the data disk array), respectively, in the mirror methods without or with parity, while still enjoying the theoretical optimal write efficiency. Our experimental evaluation with n equal to three to seven shows that the shifted element arrangement achieved an improvement of a factor of 1.54 to 4.55 in data availability during reconstruction, and compatible write efficiency.**

***Keywords*-disk element arrangement; mirror method; RAID reconstruction; data availability; read throughput**

I. Introduction

With the development of massive storage systems, the storage capacity has greatly increased recently. However, this also brings big challenges to reliability and availability in storage systems. The probability of disk failures [1, 2] and latent sector errors [3-6] arises along with storage systems' growth in size and complexity. Consequently, the probability of one or concurrent multiple disk failures is becoming higher and higher. Thus efficient RAID architectures to provide a fault tolerance of one or more disks are needed urgently. For example, the mirror method [7] is the simplest way to provide such data protection. Itself and its variants are widely used in RAID architectures and actual systems [8, 9].

Since data replications and other redundant information are stored in RAID architectures to provide fault tolerance, how to efficiently access them is essential for high performance. For example, write operations involve multiple writes on different locations, updating both the data itself and its related redundant information. The mirror method achieves the theoretical optimal construction and updating efficiency in its write strategy, because the parallel Input/Output (I/O) mechanism (i.e. each disk in RAID can read/write an element synchronously in one read/write access) is fully utilized in its write operations. This also is one of the reasons why the mirror method is often used in practical storage systems.

However, reading out data replications and other redundant information during the reconstruction process becomes a bottleneck in the mirror method and other existing RAID architectures, which hurts both reconstruction efficiency and data availability during reconstruction. The importance of this problem becomes more significant in the on-line reconstruction scenario [10], where the system continues to respond to user applications while recovering failed disks. As a result, the data availability during on-line reconstruction becomes a more and more popular problem in storage systems, and draws recent attentions to optimize RAID reconstruction strategies [10, 11].

We tackle this problem differently from previous works. We observe that the read throughput during the reconstruction process is determined by both the complexity of code computation and the maximum number of read accesses from one disk. In the traditional mirror method, during reconstruction, a large number of elements have to be read out from a single disk, which can only be done sequentially. Hence, we aim to alter element arrangements in RAID architectures to avoid such sequential reads.

In this paper, we propose a novel element arrangement method for mirror disk arrays, which can be used not only in the traditional mirror method, but also in its variants, such as the mirror method with parity. Our proposed arrangement method evenly distributes the replications of data elements in one disk to all the disks in the mirror disk array, such that the number of elements read out from one disk for reconstruction is minimized when disk failures happen. This even distribution allows parallel read accesses for fetching data replications, which leads to high data availability during reconstruction. In addition, our shifted element arrangement is orthogonal to other existing reconstruction optimization technologies [10, 11], such as the I/O Workload Outsourcing method [11], and can be used together with them.

This paper makes the following contributions:

1) We propose a shifted element arrangement method for mirror disk arrays, which maps a column of a stripe in the data disk array to a row in the mirror disk array then loop shifts each row in the mirror disk array with varying shifting distances. We also propose the shifted mirror method and the shifted mirror method with parity to show how to use our arrangement method in the traditional mirror method and its variants.
2) Theoretically, we prove three important properties of our shifted element arrangement method, and prove that our shifted mirror methods (without or with parity) achieve the theoretical optimal write efficiency, and improve data availability by a factor of n or $\frac{2n+1}{4}$, respectively, where n is the number of disks in the data disk array.
3) We implemented our shifted mirror methods (without or with parity) using Jearsure-1.2 [12] and evaluated them against traditional methods from 3 to 7 disks in each disk array. Our experimental results show the shifted mirror methods (without or with parity) achieved an improvement of a factor of 1.54 to 4.55 in data availability during reconstruction, and compatible write efficiency.

The rest of this paper will be organized as follows. The next section will provide some background and Section III will offer the motivation of our work. The new element arrangement for the shifted mirror method and its reconstruction proof are presented in Section IV, and we will see the utilization of the new element arrangement on the mirror method with parity in Section V. Then we will show the features and performance of the new arrangement in Section VI and the corresponding experiment evaluations in Section VII. Finally, the conclusions are shown in the last section.

II. Background

A number of RAID architectures have been proposed in the literature. We introduce the mirror method and two RAID architectures providing a fault tolerance of two in this section. We discuss their advantages and disadvantages in the following aspects: storage efficiency, construction and updating efficiency in write operations, data availability during reconstruction, and some other aspects that are used as the metrics to evaluate RAID architectures [13].

A. Terms and Notations

Before we describe previous RAID architectures, we first summarize a list of terms, definitions, and notations that we use throughout the paper.

Element: A chunk of data or parity information, which is the basic building block in RAID architectures. Each element contains the maximal unit of information that can be modified using the minimal number of disk I/O commands.

Stripe: A maximal set of data and parity elements that are dependent to one another in terms of redundancy relations. The elements between different stripes are independent with one another. "This is synonymous with 'algorithm instance' in that it is a complete instantiation of an erasure algorithm and is independent of any other instantiation [14]." In the research of RAID architectures and algorithms, it is very common to use "stripe" as the basic unit, thus we follow the same way in this paper and only discuss the architectures and their corresponding algorithms in one stripe.

Disk Array: A collection of disks in practice on which one or more stripes are instantiated. The disks mapping from logical to physical are rotated from stripe to stripe in order to get load-balance. See also the definition of "stack" below.

Stack: A collection of stripes in a disk array that contains all the different possible disks mappings from logical to physical. As a result, the loss of any two (or one) physical disks in a stack covers all combinations of failure of two (or one) logical disks in one stripe.

Data disk array: A disk array where the original data elements are stored. The disks in a data disk array are called *data disks*.

Mirror disk array: A disk array where a complete replication of all the original data elements is stored. The disks in a mirror disk array are called *mirror disks*.

Parity disk: Other disks besides disk arrays that are used to store the parity elements, which are the XOR sums of specific sets of data elements in the data disk array. With the help of Boolean Algebra, a parity element provides a fault tolerance of one element in its calculation set.

B. The Mirror Method

The mirror method is used in RAID 1, whereby we just copy the whole original data disk array to another disk array-the mirror disk array. As shown in Fig. 1, every data is stored twice in the mirror method. We use numbers to represent elements in Fig. 1 to illustrate the element arrangement, which means the elements labeled with the same numbers store the same data. In addition, a column represents a disk, so a disk failure means a column of elements cannot be accessed. In this example, three disks are located in one stripe of both disk arrays and each disk stores three elements.

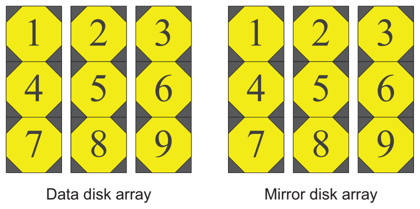

Figure 1. An example of the mirror method.

Following its original idea, other mirror disk arrays can be added into RAID 1 to make it able to provide higher fault tolerance. These methods are used a lot in recent distributed file systems, such as the three-mirror method used in [8, 9], which stores three replicas of data.

The mirror method is the simplest way to provide data protection, where the construction and updating efficiency attains to the optimality. The reconstruction in the mirror method needs to read out the theoretical least number of elements; however, all these elements are stored on the same disk, thus reading them out can only be done sequentially, which significantly impacts the data availability during the reconstruction process.

C. RAID Architectures to Provide a Fault Tolerance of Two

RAID 1 to 5 are the original RAID levels when RAID was proposed [7] and each of them provides a single disk protection. Nearly all the RAID architectures that provide a fault tolerance of two or even more can be viewed as the extensions or combinations from these original RAID architectures.

In particular, RAID 5 is an MDS (Maximum Distance Separable) [15] RAID architecture with optimal storage efficiency. In RAID 5, the XOR sum of data elements in each row is calculated and set as a parity element to be stored in another disk, i.e. the parity disk. In this way, RAID 5 can tolerate any single disk failure and the modification of a single data element in RAID 5 only leads to update one parity element, which attains to the theoretical optimality. Furthermore, the rotation of the parity disk in practice gets rid of the bottleneck in the write strategy of RAID 5. However, all the intact elements need to be read out in any reconstruction processes of RAID 5, leading to very low data availability during reconstruction.

There are two ways to expand RAID 5 into a RAID architecture that provides a fault tolerance of two. The first way is to copy the whole the data disk array to a new mirror disk array, namely, the mirror method with parity. The other way is to add another parity disk, making the architecture RAID 6 [16].

1) The Mirror Method with Parity: The mirror method with parity is a RAID architecture to provide a fault tolerance of two. As shown in Fig. 2, besides data disk array and mirror disk array, a parity disk is added to make up this architecture. It is clear that this architecture keeps the construction and updating efficiency optimal, but its storage efficiency is always less than 50 percent.

In most failure situations, when the two failed disks are not the replications of each other, only the elements on their own corresponding replication disks need to be read out for data recovery. However, this read out phase can only be done sequentially because all the elements we need are on the same disk, which is harmful for the data availability during the reconstruction process.

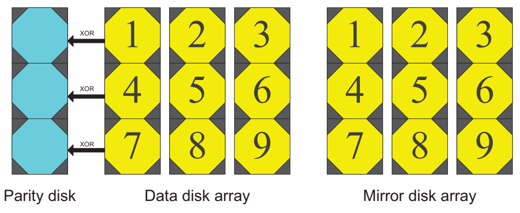

Figure 2. An example of the mirror method with parity.

2) RAID 6: RAID 6 is the first level of RAID architecture that provides a fault tolerance of two [16]. Many erasure code instances for RAID 6 are proposed in the literature, examples include EVENODD code [17] and RDP code [18].

There are two parity disks in most of the RAID 6 codes, so they are always MDS codes with the optimal storage efficiency. In most RAID 6 codes, the calculation algorithm of the first parity disk is the same as that of the parity disk in RAID 5, but that of the second parity disk is much more complicated. As proved by [19, 20], horizontal RAID 6 code cannot attain the theoretically optimal construction and updating efficiency, which means the modification for a single data element in RAID 6 may lead to update more than two parity elements.

There are two factors behind the low data availability during the reconstruction process of RAID 6. The main reason is that all the intact elements in RAID 6 have to be read out to recover the failed elements in nearly all failure situations. The second factor is that the complicated calculation algorithm for the second parity disk makes the reconstruction algorithm cumbersome; a lot of calculations have to be done before reconstructing the failed elements.

III. Motivation & Problem Statement

During the on-line reconstruction process [10], the storage system keeps on serving user applications. When a user requires to read data on the disk under reconstruction, the failed data is recovered and responded to user with a higher priority than other reconstruction I/Os. Hence, it is very important to quickly make failed data available during the reconstruction process. Formally, we define the data availability during reconstruction at the disk array level as the amount of recovered data read out from disk arrays in unit time [21], which is regarded as *read throughput*. Regardless of the actual strategy employed in the underlining Operation System and File System, higher read throughput during the reconstruction process leads to more failed data available at the disk array level, hence the storage system could respond to user read requests faster. In this paper, we focus on improving current RAID architectures to achieve high data availability during reconstruction.

One major limitation of current RAID architectures is the low data availability during reconstruction. For example, either in the mirror method or the mirror method with parity, when one disk from the data disk array fails, all the replications of the failed elements can be read out only from its corresponding replication disk in the mirror disk array. Even worse, all the intact elements need to be read out to reconstruct the failed elements in RAID 5 and RAID 6. The reason why we have to read elements sequentially and suffer from low read throughput in these traditional RAID architectures is because the elements we need are on one single disk, which cannot bring parallel I/O into play.

On the other hand, we know that elements from different disks can be read out simultaneously thanks to the parallel I/O mechanism in RAID architectures. For example, for n elements, if they are on the same disk, we need n read accesses in completion their sequential read, whereas, only one read access is needed if they are on n different disks. In other words, the number of read accesses in a RAID architecture is determined by the maximum number of elements that we read out from one disk. If we minimize the maximum number of elements we read out from one disk, we can recover the same amount of data with least number of read accesses, thus improving the data availability during the reconstruction process. This observation directly motives us to distribute the replications of the elements that on a disk evenly to all the disks in the mirror disk array, leading to a novel element arrangement in mirror disk arrays to provide high data availability during reconstruction.

We also consider write efficiency in designing our new element arrangement method. The efficiency of write operations is important to provide fast response to user write requests, and varies a lot in different RAID architectures. Both the traditional mirror method and mirror method with parity provide the optimal write efficiency, but RAID 6 cannot [19, 20]. To keep the theoretical optimal write efficiency is also a mission in designing the new element arrangement in this paper.

IV. Shifted Mirror Method

In our new mirror method, there are two disk arrays (i.e. the data disk array and the mirror disk array). Data are stored in both disk arrays, i.e. the original data elements are stored in the data disk array and their replication elements are stored in the mirror disk array. However, different from the traditional mirror method, the mirror disk array is no longer just a simple replication of the data disk array; we rearrange the elements in the mirror disk array by some shifting operations to provide high data availability by full utilization of parallel I/O during the reconstruction process. Under the novel element arrangement, our new mirror method is regarded as the *shifted mirror method*.

We first illustrate the basic idea of the shifted mirror method with an example in Fig. 3. Without loss of generality, we can focus on the architecture and its corresponding algorithm in one stripe [14], so only one stripe is shown in the figure. Later on, we will describe the systematic

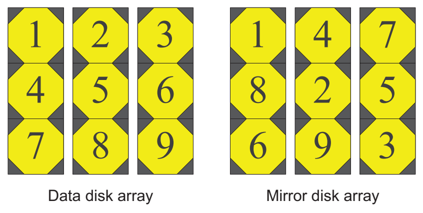

Figure 3. An example of the shifted mirror method.

element arrangement method in mirror disk arrays formally and verify features and performance that the shifted mirror method provides.

Similarly, we use numbers to represent elements in Fig. 3, which means the elements labeled with the same numbers store the same data. In addition, a column represents a disk in the figure, so a disk failure means a column of elements cannot be accessed. In this example, three disks are considered in one stripe of both disk arrays and each disk stores three elements. For example, elements 1, 4 and 7 are stored on the same data disk, but their replications are stored on different mirror disks. This is the key insight of our paper that the mirror disk array contains the same data as in the data disk array, but with different element arrangement.

A. Element Arrangement

In Fig. 4, we illustrate the systematic element arrangement method in mirror disk arrays. The elements in Fig. 4 are labeled with two numbers separated by a comma, with the first number being the column index in the data disk array and the second number being the row index in the data disk array. In addition, the two elements marked with the same pairs of numbers store the same data.

We assume both the data disk array and the mirror disk array have n disks. Our arrangement method would like to distribute elements from one data disk to all the n mirror disks. Hence, we consider n elements for each stripe to maximize storage efficiency. As a result, there are $n \times n$ elements in a stripe for each disk array and no additional limitations on n. Actually, the number of rows in a stripe can be chosen to adapt to the number of data disks because there

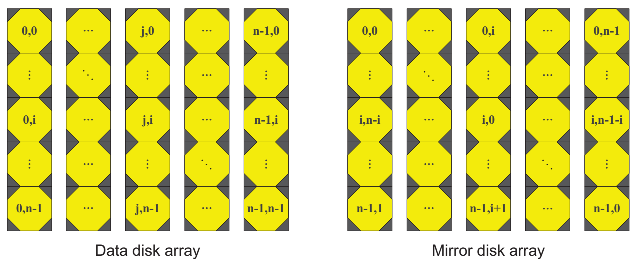

Figure 4. Systematic element arrangement in the shifted mirror method.

are many stripes in each practice disk array. Furthermore, the increase in the capacity of disks will lead to more stripes in the whole disk array but will not change the shape of each stripe.

In the following description, we use the standard mathematical notation $\langle x \rangle_y$ to represent the non-negative remainder when x is divided by y. That is to say, $\langle x \rangle_y = z$ if and only if $x \equiv z \pmod{y}$, $0 \leq z < y$. For example, $\langle 5 \rangle_3 = 2$ and $\langle -1 \rangle_5 = 4$.

The disks are numbered 0 to $n-1$ from left to right while the elements in each disk are numbered 0 to $n-1$ from top to bottom in both disk arrays. Now we can denote the element $a_{i,j}$ ($0 \leq i < n$, $0 \leq j < n$) as the j^{th} element on the i^{th} data disk and the element $b_{i,j}$ ($0 \leq i < n$, $0 \leq j < n$) as the j^{th} element on the i^{th} mirror disk.

In our novel element arrangement, we take the elements on each data disk onto different mirror disks, so that the read throughput during the reconstruction processes of these disks could be accelerated by parallel I/O. Exchanging the column index-i and the row index-j is one of the best choices to achieve this. However, under this exchange, the elements on the same row in the data disk array are allocated on the same mirror disk, which is badly harmful for the write efficiency. Thus, loop shifts are taken in each row of the mirror disk array after the exchanging. The shifting distance of each row is determined by its row index, i.e. the final column location is no longer j, but $\langle i+j \rangle_n$. Above all, the element arrangement in shifted mirror method can be expressed by:

$$a_{i,j} = b_{\langle i+j \rangle_n, i}, 0 \leq i < n, 0 \leq j < n.$$

This formula can be translated so as to easily represent the elements in the mirror disk array, i.e.:

$$b_{i,j} = a_{j, \langle i-j \rangle_n}, 0 \leq i < n, 0 \leq j < n.$$

Actually, we can explain the element arrangement method in mirror disk arrays in a visualized fashion: We just take the elements on each data disk onto each row of the mirror disk array then loop shift them with a distance determined by the column index in the data disk array.

Alternatively, we could perform the arrangement in two steps. Firstly, the first element of each data disk (the elements on the first row) are exactly allocated one by one on the left-most main diagonal of slope 1 (the diagonal contains the top-left element) in the mirror disk array. This allocation is shown in Fig. 5.

In the second step, once the first element of each data disk is placed in the mirror disk array, the rest elements on each data disk can be placed easily. In particular, the elements on a data disk are taken out from top to bottom, and their replications are placed from left to right after the replication of first element in the mirror disk array. If the destination disk goes beyond disk $n-1$ in the mirror disk array, we go back and start from mirror disk 0 again.

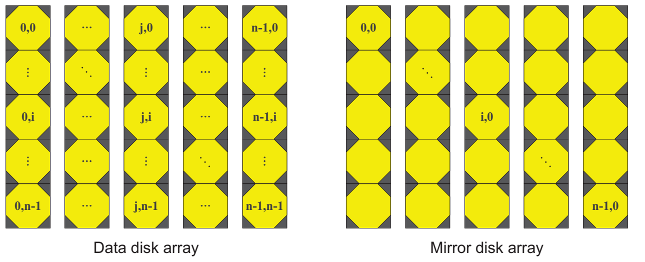

Figure 5. Main diagonal elements in the mirror disk array of the shifted mirror method.

B. Reconstruction Analysis

Under our novel element arrangement in the mirror disk array, the shifted mirror method still protect data from any single disk failure. Because each element and its replication are stored in different disk arrays, when any single disk fails, the replications of the failed elements on this disk are intact. So no data has been lost.

We provide two properties of the shifted mirror method before analyzing the read throughput during reconstruction.

From the arrangement formulas, any two different elements on the same data disk, $a_{i,p}$ and $a_{i,q}$, are allocated on mirror disks $\langle p+i \rangle_n$ and $\langle q+i \rangle_n$, respectively. Since $a_{i,p}$ and $a_{i,q}$ are different elements, $p \neq q$, we also have $0 \leq p, q < n$, so that $\langle p+i \rangle_n \neq \langle q+i \rangle_n$, which implies that they are allocated on different mirror disks. Therefore, we can conclude that the replications of different elements on the same data disk are allocated on different mirror disks. In addition, there are n elements in each data disk and exactly n mirror disks. Above all, we can get the first important property of the shifted mirror method for each stripe:

Property 1. *The replications of the elements on the same data disk are allocated on all the mirror disks, each on exactly one different mirror disk.*

Furthermore, by definition, any two different elements on the same mirror disk, $b_{i,p}$ and $b_{i,q}$, are replicated from data disk p and q, respectively. Since $p \neq q$, they are replicated from different data disks. Thus different elements on the same mirror disk must be replicated from different data disks. So we can get a similar conjecture with Property 1, which is the second important property of the shifted mirror method for each stripe:

Property 2. *The elements on the same mirror disk are replicated from all the data disks, each from exactly one different data disk.*

From Property 1 and Property 2, in the shifted mirror method, the corresponding replications of the elements on any disk are allocated on all the disks in the other disk array, each on exactly one different disk. As a result, in any single disk failure situation, we can read out the replications of

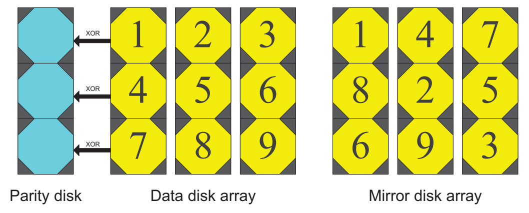

Figure 6. An example of the shifted mirror method with parity.

failed elements across all the disks in the other disk array. Thanks to parallel I/O in RAID architectures, each disk can read out an element synchronously in one read access. Above all, we can read out the replications of all the failed elements in one read access, which significantly improves the read throughput during the reconstruction process. To make a conclusion, in the mirror method with n disks in both disk array, the read throughput during the reconstruction process of the shifted mirror method increases n times in a stripe to that of the traditional one.

V. Shifted Mirror Method with Parity

A. Element Arrangement

The above novel element arrangement in mirror disk arrays can also be implemented to the mirror method with parity. As shown in Fig. 6, besides the data disk array and the mirror disk array under our novel element arrangement, a parity disk is added to provide a fault tolerance of two. So we name it *shifted mirror method with parity*.

The elements in the parity disk are numbered 0 to $n-1$ from top to bottom, we denote the element c_j $(0 \leq j < n)$ as the j^{th} element on the parity disk. Beside the notations defined in Subsection IV-A, we use the mathematical notation $\oplus$ to represent the XOR operation.

The parity disk is used in the same way as in the original mirror method with parity. So we can calculate the parity elements by:

$$c_j = \bigoplus_{i=0}^{n-1} a_{i,j}, 0 \leq j < n.$$

The relationship between the element arrangement in the data disk array and that in the mirror disk array obey the same rules as discussed in Subsection IV-A. So the shifted mirror method with parity still keeps Property 1 and Property 2.

B. Reconstruction Analysis

Under our novel element arrangement, the shifted mirror method with parity has the ability to provide a fault tolerance of two. We enumerate all possible failure situations to verify this and provide the number of read accesses that are needed to recover the failed elements and respond to user read requests during the reconstruction process of each failure situation below.

1) A Single Disk Failed: When one of the disks in either disk array failed in the shifted mirror method with parity, the replications of all its elements are stored on the other disk array and all the failed elements can be recovered without any data loss. When the parity disk failed, there is no data in failure, so it does not affect the responding to user read requests. However, we have to read out all the data to recalculate the parity elements.

As mentioned in Subsection IV-B, when the failed disk is one of the disks in either disk array, from Property 1 and Property 2, we can read out all the replications of the failed elements of a single stripe in one read access with the utilization of parallel I/O in RAID architectures.

2) The Two Failed Disks Include the Parity Disk: In this case, one of the two disk arrays is intact, thus no data is lost. Since the parity disk failed, we have to read out all the data to recalculate the parity elements. Similarly, the replications of all the elements on the other failed disk of a single stripe can be read out through one read access from the intact disk array by Property 1 and Property 2, which significantly improves the read throughput during the reconstruction process.

3) The Two Failed Disks are in the Same Disk Array: In this case, the other disk array is also intact, so the recoverability of this situation is obvious. In this case, we need two read accesses to read out all the replications of the failed elements in a single stripe, one read access per failed disk.

4) Each Disk Array Contains One Failed Disk: To make it more convenient in the following description, without loss of generality, we assume that the failed disks are the x^{th} $(0 \leq x < n)$ data disk and the y^{th} $(0 \leq y < n)$ mirror disk.

By definition, for each stripe, only one element of data disk x has its replications on mirror disk y (i.e. $a_{x,\langle y-x\rangle_n} = b_{y,x}$), and vice versa. Thus, the rest of the elements on data disk x have their replications stored intact in the mirror disk array and can be recovered from these replications. Similarly, we can verify the recoverability of the elements except $b_{y,x}$ on mirror disk y as well. After this step, we can recover $a_{x,\langle y-x\rangle_n}$ $(b_{y,x})$ by using the XOR sum of other elements on the $\langle y-x\rangle_n{}^{th}$ row of data disk array and the parity element $c_{\langle y-x\rangle_n}$ in the parity disk. Hence, the recoverability of this failure situation has been verified.

In terms of the number of read accesses to read out all the data needed for recovering failed elements in one stripe, our shifted mirror method with parity improves the traditional method greatly as well. For each stripe, the replications of the elements, except $a_{x,\langle y-x\rangle_n}$, on data disk x can be read out through one synchronous read access from the mirror disk array, concurrently one synchronous read access from the data disk array can bring the replications of the elements, except $b_{y,x}$, on mirror disk y.

To recover $a_{x,\langle y-x\rangle_n}$, we need to read out all the elements from the $\langle y-x\rangle_n{}^{th}$ row of the data disk array and the

$\langle y-x \rangle_n{}^{th}$ element on the parity disk-$c_{\langle y-x \rangle_n}$, which can be done by one read access as well.

In summary, only two read accesses are needed for recovering failed elements for one stripe in our shifted mirror method with parity.

We have enumerated all the failure situations above and proved that we can always find a reconstruction strategy for each one of them. Thus, we have proved that our shifted mirror method with parity has the ability to provide a fault tolerance of two.

VI. FEATURES

In this section, we discuss features and performance achieved by our shifted mirror methods (without or with parity): the high data availability during reconstruction, the reconstruction efficiency, the theoretical optimal write performance, the storage efficiency, and other element arrangement methods to achieve these features. We focus on theoretical analysis in this section and provide the corresponding experimental results in the next section.

As illustrated in [14], various performance metrics can be measured by rigorous counting and averaging with an assumption of equal failure probability for all disks. We follow the same methodology in measuring performance of our shifted mirror methods (without or with parity). In particular, the stack notion allows us to do the measurements in the following evaluations by rigorous counting and averaging on a simple stripe [14]. To see that, in the shifted mirror method (without or with parity), the placement of disks is rotated from stripe to stripe in practice. From the definition of "stack" in Subsection II-A, we can logically regard all the disks in a stripe as having the same possibility of getting into failure. In addition, the rotation of the parity disks in practice in the shifted mirror method with parity avoids bottleneck effects when repeated write operations are performed.

A. Data Availability During Reconstruction

In this subsection, we use the number of read accesses to measure data availability during reconstruction. The less number of read accesses is used to read out unit data, the higher data availability is provided.

In either the traditional or our shifted mirror method, each disk from the two disk arrays is equally likely to fail, hence the average number of read accesses for recovering one stripe is the same as the one when any disk fails. From our discussions in Subsection II-B and Subsection IV-B, we conclude that the read throughput during the reconstruction process of our shifted mirror method increases n times in a stripe to that of traditional mirror method when there are n data disks.

For our shifted mirror method with parity, we have enumerated all the failure situations in Subsection V-B and provided the number of read accesses that are needed to recover the failed elements and respond to user read requests during the reconstruction process of each failure situation. Now, we count the number of cases of each failure situation by combinatorial algorithm, and summarize all the double-disk failure situations in Table I to provide a clearer analysis on the read throughput during reconstruction in terms of the number of read accesses.

Table I
READ THROUGHPUT DURING THE RECONSTRUCTION PROCESS OF OUR SHIFTED MIRROR METHOD WITH PARITY.

Failure situation F_i	Number of cases $Num\_Case(F_i)$	Number of read accesses $Num\_Read(F_i)$
F_1: The two failed disks include the parity disk	$2n$	1
F_2: The two failed disks are in the same disk array	$n(n-1)$	2
F_3: Each disk array contains one failed disk	n^2	2

The overall average number of read accesses can be generated by calculating the expectation value of Table I. We use F_i to represent each failure situation, thus $Num\_Case(F_i)$ and $Num\_Read(F_i)$ are the number of cases and the number of read accesses, respectively, in F_i. So the average number of read accesses during the reconstruction process of the shifted mirror method with parity can be calculated by:

$$\begin{aligned} Avg\_Read &= \frac{\sum(Num\_Case(F_i) \times Num\_Read(F_i))}{\sum Num\_Case(F_i)} \\ &= \frac{2n \times 1 + n(n-1) \times 2 + n^2 \times 2}{2n + n(n-1) + n^2} \\ &= \frac{4n}{2n+1}. \end{aligned}$$

To ease the comparison of the read throughput during the reconstruction processes among various RAID architectures to provide a fault tolerance of two, we also plot the relative theoretical read throughput in a stripe of our shifted mirror method with parity against the traditional mirror method with parity and RAID 6 as the number of data disks increases. In particular, we measure the read throughput as the average number of read accesses during the reconstruction process. The ratios of the number of read accesses of our method over traditional RAID architectures are presented in Fig. 7.

The read throughput during the reconstruction process of RAID 6 in Fig. 7 is a little lower than that of traditional mirror method with parity because of the use of the "shorten" method [22]. As shown in Fig. 7, the ratios decrease very fast as the number of data disks increases in the storage system, achieving as low as 5 percent. Under

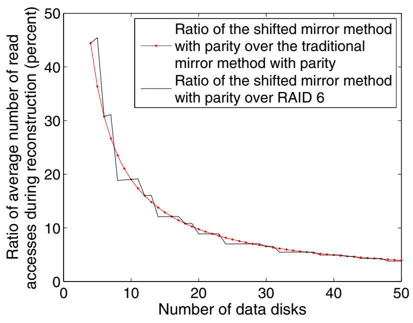

Figure 7. Theoretical read throughput during reconstruction comparisons between our shifted mirror method with parity and traditional RAID architectures.

our novel element arrangement, all the disks are under load balance (i.e. the disks in a disk array are equally burdened the read load, thus minimize the maximum number of read accesses from a single disk) during the reconstruction process. We only need to read out at most one (or two) element from one disk in a stripe in any failure situation of the mirror method without (or with) parity to recover the failed elements and respond to user read requests. Thanks to parallel I/O, we can significantly improve data availability during the reconstruction process by read all these elements in one (or two) read access.

In addition, nearly all the failed elements can be recovered by just copying from their replications in either the shifted mirror method or the shifted mirror method with parity. Only one element needs to be calculated by a quite simple algorithm in the failure situation that each disk array contains one failed disk in our shifted mirror method with parity. Hence, the reconstruction algorithm is very simple, which leads to a higher data availability during the reconstruction process, comparing with RAID 6.

B. Reconstruction Efficiency

During the reconstruction process of our shifted mirror methods (without or with parity), we can rapidly read out the needed data (either the replications of failed elements or the elements to help calculating failed elements), which may potentially improve reconstruction efficiency, especially for disk arrays where write speed is faster than read speed (for example, in our experiment environment in the next section). Different reconstruction strategies and optimizations [10, 11] may be used to provide different reconstruction efficiencies and trade off between data availability and reconstruction efficiency; our shifted element arrangement can be implemented orthogonally with them to provide higher data availability and reconstruction efficiency.

C. Write Strategy

The term *small write* is defined as the modification of a single data element in RAID architectures. However, since redundant information is provided in RAID, the modification for a single data element needs to update all the redundant information related to it. In our shifted mirror method, the modification of a single data element needs to write down to the two disk arrays, updating both the original element and its replication, which is the theoretical optimality for the small write strategy in RAID architectures to provide a single disk protection.

For our shifted mirror method with parity, besides the original element and its replication, we have to update the corresponding parity element at the same time, which also attains to the theoretical optimality in RAID architectures to provide a fault tolerance of two. In contrast, RAID 6 code cannot attain the theoretically optimal updating efficiency [19, 20], which is a bottleneck for the small write efficiency of RAID 6.

Besides small write, one of the important goals of designing RAID architectures is to achieve high write efficiency for *large write*, which means writing data elements row by row in the data disk array [7, 16]. The large write strategy employed in traditional RAID architectures writes an element down to each disk synchronously, hence a row of data elements can be written down in one write access.

Our shifted mirror methods (without or with parity) attain to optimal write efficiency for large write as well. To see this, we need to derive another important property of our novel element arrangement. Assume $a_{p,i}$ and $a_{q,i}$ are two different elements allocated in row i in the data disk array. By definition, their replications are allocated in disk $\langle i+p\rangle_n$ and disk $\langle i+q\rangle_n$, respectively, in the mirror disk array. Since $0 \leq i, p, q < n$, $p \neq q$, we have $\langle i+p\rangle_n \neq \langle i+q\rangle_n$, which means their replications are allocated in different mirror disks. Hence, we have the third property of our shifted element arrangement:

Property 3. *The elements that come from the same row of the data disk array are allocated on different mirror disks, each element on exactly one different mirror disk.*

From Property 3, when we modify data elements on a specific row in the data disk array, we can update their corresponding replications in the mirror disk array with one write access thanks to parallel I/O. Furthermore, the write operation for a row of elements in the data disk array and the updating for their corresponding parity element can be accomplished at the same time. As a result, under our novel element arrangement, we can accomplish the large write for a row of n data elements with one write access, which attains to the theoretical optimality in RAID architectures.

D. Storage Efficiency

RAID 6 is an MDS RAID architecture with optimal storage efficiency, which equals $\frac{n}{n+2}$, where n is the number of disks that contain data elements. In either the traditional or our shifted mirror methods (without or with parity), the

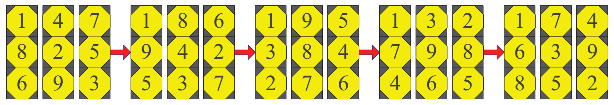

Figure 8. The resultant element arrangements after each iteration step.

two disk arrays have the same number of disks, but one stores data elements while the other stores their replications. In addition, there is a parity disk in the mirror method with parity. Assuming we have n data disks, the storage efficiency equals $\frac{n}{2n}$ in the mirror method and $\frac{n}{2n+1}$ in the mirror method with parity. With the increasing number of disks, the storage efficiency arises and approaches to 50 percent eventually.

In contrast to RAID 6, the mirror method with parity does have around 50 percent sacrifice in storage efficiency, but under our novel element arrangement, it needs as low as 5 percent number of read accesses of RAID 6 during the reconstruction process as shown in Fig. 7, which significantly improves the data availability during reconstruction. In summary, our shifted mirror methods (without or with parity) are one of the results in balancing storage for data availability, reconstruction efficiency, write efficiency, and other positive features.

E. Other Element Arrangements in Mirror Disk Arrays

The arrangement in this paper is not the only approach for the element arrangement in mirror disk arrays that provides high data availability during reconstruction. Other arrangements that satisfy the three properties could also be used in mirror disk arrays to provide the same features. In this subsection, we provide a systematic method to generate other equally powerful element arrangements.

Our novel element arrangement in mirror disk arrays can be regarded as a transformation function from the arrangement of $n \times n$ elements in the data disk array to an arrangement of $n \times n$ elements in the mirror disk array. We can apply this transformation function to the basic element arrangement iteratively. The resultant element arrangements after each iteration step are illustrated in Fig. 8.

The transformation function actually takes a column (disk) of elements in the previous arrangement onto a row in the next arrangement and vice versa. Therefore, from the original data disk array, the arrangements obtained by performing odd number of such transformations satisfy Property 1 and Property 2. For example, in Fig. 8, the leftmost arrangement is the one used in the shifted mirror method, and the third and fifth arrangements also satisfy Property 1 and Property 2. However, not all these arrangements comply with Property 3. For example, the fifth arrangement in Fig. 8 satisfies Property 3 while the third one does not.

All in all, we can apply the transformation function iteratively to find other element arrangements in mirror disk arrays, but we have to check the arrangements carefully to make sure that they satisfy all three properties.

VII. Experimental Evaluations

We implemented both the traditional and shifted mirror methods (without or with parity) and evaluated them in terms of read throughput during reconstruction and write efficiency on real disk arrays. The implementation was based on an open source library, Jerasure-1.2 [12], which is widely used by the erasure code community [23].

All our experiments were performed on a machine with Intel Xeon processor (4 cores) that running at 2.40 GHz and 10 G physical memory, and the Operating System is SUSE Enterprise Server 11 with a kernel of V2.6.33. All the disks in our experiments are sitting in an individual array with 16 SAS disks. The disk type is Seagate/Savvio 10K.3, and the disk model number is ST9300603SS. Every disk is 300 GB and 10000 rpm with a cache of 16 MB, and each provides a peak read speed of 54.8 MB/Sec and a peak write speed of 130 MB/Sec.

In the following experiments, we set each element with a size of 4 MB, which is a typical choice in storage systems [24]. Because we aim to exam the read throughput during reconstruction, the actual size of data under reconstruction does not matter, and we encoded a film file and stored 17 GB data on each data disk. The number of data disks, n, varied from 3 to 7 in our experiments. Multiple stacks were considered in our experiment, according to its definition in Subsection II-A and property [14], each stack contains all the possible mappings from logical disks to physical disks, so the experimental result is a true reflection of read and write efficiency.

A. Read Throughput During Reconstruction

We enumerated all the disks, either the data disk or the mirror disk, to be the virtual failed disk in the mirror method. Either in the traditional or shifted mirror method, we tried to reconstruct the failed disk and recorded the read throughput during this reconstruction process. Finally, we averaged these values, which are shown in Fig. 9(a).

Similarly, we enumerated all the combinations of two disk failures (can be as many as 105 cases for 7 data disks, 7 mirror disks, and 1 parity disk) and averaged their read throughput values during the reconstruction process as the results to compare traditional and shifted mirror method with parity, which are shown in Fig. 9(b).

Note that after each reconstruction process, we also compared the original data on the virtual failed disk and the recovered data to verify the correctness of the reconstruction process.

Since the reconstruction processes of both the traditional mirror method with parity and RAID 6 need to read out all

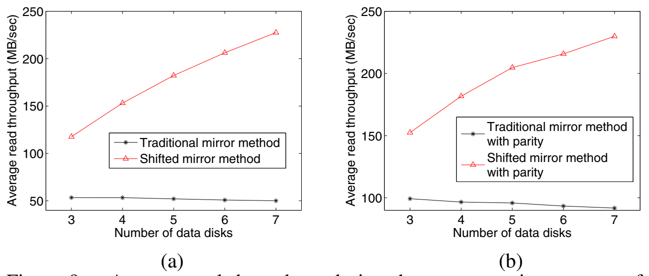

Figure 9. Average read throughput during the reconstruction process of the traditional and shifted (a) mirror method and (b) mirror method with parity.

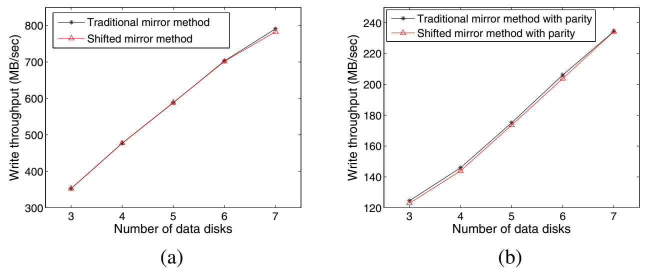

Figure 10. Write performance of the traditional and shifted (a) mirror method and (b) mirror method with parity.

the elements from specific disks (or even worse for RAID 6 with the "shorten" method [22]), we only implemented the traditional mirror method with parity, and the comparison between our method and RAID 6 is similar.

As shown in Fig. 9, the average read throughput during the reconstruction process of shifted mirror methods (without or with parity) is 1.54 to 4.55 times higher than that of traditional ones, which is in agreement with the observations from Fig. 7 to a large extent. We also observe the average read throughput of our shifted mirror methods (without or with parity) increase as n increases because of the higher I/O parallelism, whereas that of the traditional ones stay stable as n increases. Thus the read throughput gap between these two arrangements' reconstruction process becomes higher as the number of data disks increases.

Note that the read access operations during the reconstruction process of our shifted mirror methods (without or with parity) are "random reads," while that of the traditional ones are "sequential reads," which eliminates the seek time. In addition, there may be some I/O merge optimizations for sequential reads in OS. Therefore, the empirical improvement of data availability is not as high as the theoretical analysis, but is still substantial to indicate that the data availability is improved significantly by using our novel element arrangement method.

B. Write Performance

We created a workload of one thousand random large write operations of the size varying from one element to as large as a whole stripe. The traditional and shifted mirror methods (without or with parity) were tested under the same workload to ensure the fairness of our experiments. In our experiments, we chose to directly write to disks by setting a parameter in Jerasure-1.2 to avoid cache effects. The comparison results in terms of write throughput are shown in Fig. 10.

The write performance of the mirror method is better than that of the mirror method with parity, because we have to update the corresponding parity elements in the latter's write strategy. According to the scale and positional relationships of the modified data elements, we chose "read-modify-write" or "reconstruct-write" strategy on the updating of parity elements. In either strategy, we have to read out some data elements and the original parity elements, then use them to calculate the new values of the parity elements.

As shown in Fig. 10, the write performance of the traditional and our shifted mirror methods is about the same to a large extent. Since the traditional mirror methods (without or with parity) attain to the theoretical optimal write performance, besides the theoretical analysis in Subsection VI-C, we have shown by experimental results that our shifted mirror methods (without or with parity) provide the optimal write performance.

VIII. Conclusions

We have proposed a novel shifted element arrangement method for the mirror method and its variants in this paper. In particular, we have shown how to adapt this arrangement method in the mirror method and the mirror method with parity. Using this shifted element arrangement, we evenly distribute the replications of data elements in one disk to all the disks in the mirror disk array, leading to higher data availability during reconstruction. In addition, the shifted mirror methods (without or with parity) achieve the theoretical optimal write efficiency. We have implemented our shifted mirror methods (without or with parity) and evaluated them against the traditional methods. Our experimental results show that our methods achieve an improvement of a factor of 1.54 to 4.55 in data availability during reconstruction, and compatible write efficiency. In the future, we intend to extend our current shifted element arrangement to cope with other existing RAID architectures, such as the three-mirror method used in [8, 9].

Acknowledgment

The authors would like to thank the reviewers for their constructive comments. We also thank MingQiang Li and Letian Yi for helpful discussion, Jianping Qiu and Miao Duan for the help of experiments. This work is funded by the National Natural Science Foundation of China (Grant No.60925006), the Scientific and Technical

Supporting Programs funded by Ministry of Science & Technology of China (Project No. 2009BAH42B0202), and Intel International Cooperation Program (Intel-CRC-2010-06).

REFERENCES

[1] E. Pinheiro, W.-D. Weber and L. A. Barroso, "Failure trends in a large disk drive population," in *Proceedings of 5th USENIX Conference on File and Storage Technologies (FAST'07)*, San Jose, CA, February. 2007, pp. 17-28.

[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?" in *Proceedings of 5th USENIX Conference on File and Storage Technologies (FAST'07)*, San Jose, CA, February. 2007, pp. 1-16.

[3] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy and J. Schindler, "An analysis of latent sector errors in disk drives," in *Proceedings of 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'07)*, San Diego, CA, June. 2007, pp. 289-300.

[4] J. L. Hafner, V. Deenadhayalan, W. Belluomini, and K. K. Rao, "Undetected disk errors in RAID arrays," *IBM Journal of Research and Development*, Vol. 52 No. 4/5, July/September 2008, pp. 413-425.

[5] 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," in *Proceedings of 6th USENIX Conference on File and Storage Technologies (FAST'08)*, San Jose, CA, February 2008, pp.127-141.

[6] B. Schroeder, S. Damouras, and P. Gill, "Understanding Latent Sector Errors and How to Protect Against Them," in *Proceedings of 8th USENIX Conference on File and Storage Technologies (FAST'10)*, San Jose, CA, February 2010, pp. 71-84.

[7] D. A. Patterson, G. Gibson and R. H. Katz, "A case for redundant arrays of inexpensive disks," in *Proceedings of the ACM SIGMOD*, pp. 109- 116, June. 1988.

[8] S. Ghemawat, H. Gobioff and S.-T. Leung. "The Google File System," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, 2003, 29-43.

[9] Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E. and Maltzahn, C. "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI'06)*, 2006, 307-320.

[10] M. Holland. "On-Line Data Reconstruction in Redundant Disk Arrays." PhD thesis, Carnegie Mellon University, Apr. 1994.

[11] S. Wu, H. Jiang, D. Feng, L. Tian and B. Mao. "WorkOut: I/O Workload Outsourcing for Boosting RAID Reconstruction Performance." in *Proceedings of 7th USENIX Conference on File and Storage Technologies (FAST'09)*, San Francisco, CA, USA, February, 2009.

[12] J. Plank, S. Simmerman and C. Schuman. "Jerasure: A library in C/C++ facilitating erasure coding for storage applications-Version 1.2." *Technical Report CS-08-627*, University of Tennessee, August 2008.

[13] M. Li, J. Shu and W. Zheng, "GRID Codes: Strip-Based Erasure Codes with High Fault Tolerance for Storage Systems." *ACM Transactions on Storage*, Vol.4, No.4, Article 15, January 2009.

[14] J. M. Hafner, V. Deenadhayalan, T. Kanungo and KK Rao, "Performance Metrics for Erasure Codes in Storage Systems," *Technical Report*, RJ 10321, IBM Research, San Jose, CA, 2004.

[15] F. J. MacWilliams and N. J. A. Sloane. "The Theory of Error-Correcting Codes." *New York: North-Holland*, 1977.

[16] 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, June 1994.

[17] M. Blaum, J. Brady, J. Bruck and J. Menon, "EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures," *IEEE Transactions on Computer*, Vol. 44, No. 2, February 1995.

[18] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong and S. Sankar, "Row-Diagonal Redundant for Double Disk Failure Correction," in *Proceedings of 3rd USENIX Conference on File and Storage Technologies (FAST'04)*, San Francisco, CA, USA, March 31-April 2, 2004.

[19] M. Blaum and R.M. Roth, "On Lowest Density MDS Codes," *IEEE Transactions on Data Theory*, Vol. 45, No. 1, January 1999.

[20] M. Blaum, J. Bruck, and A. Vardy, "MDS array codes with independent parity elements," *IEEE Transactions on Data Theory*, Vol. 42, No. 2, pp. 529-542, March. 1996.

[21] A. Brown and D. A. Patterson. "Towards availability benchmarks: A case study of software RAID systems." In *Proceedings of 2000 USENIX Annual Technical Conference*, San Diego, California, USA, June 18-23, 2000.

[22] C. Jin, H. Jiang, D. Feng and L. Tian, "P-code: A new RAID-6 code with optimal properties," in *Proceedings of the 23rd International Conference on Supercomputing (ICS'09)*, Yorktown Heights, NY, June 2009, pp. 360-369.

[23] J. Plank, J. Luo, C. D. Schuman, L. Xu and Z. W. O' Hearn, "A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries For Storage," in *Proceedings of 7th USENIX Conference on File and Storage Technologies (FAST'09)*, San Francisco, CA, USA, February, 2009.

[24] J. Schindler, S. W. Schlosser, M. Shao, A. Ailamaki, G. R. Ganger, "Atropos: A Disk Array Volume Manager for Orchestrated Use of Disks," in *Proceedings of 3rd USENIX Conference on File and Storage Technologies (FAST'04)*, San Francisco, CA, USA, March 31-April 2, 2004.