OI-RAID: A Two-layer RAID Architecture Towards Fast Recovery and High Reliability

Neng Wang[1], Yinlong Xu[1,2], Yongkun Li[1,3], Si Wu[1]
[1]School of Computer Science and Technology, University of Science and Technology of China
[2]Collaborative Innovation Center of High Performance Computing, National University of Defense Technology
[3]AnHui Province Key Laboratory of High Performance Computing
{*campnou, wusi*}@*mail.ustc.edu.cn*, {*ylxu, ykli*}@*ustc.edu.cn*

***Abstract*—A lot of inexpensive disks in modern storage systems induce frequent disk failures. It takes a long time to recover a failed disk due to its large capacity and limited I/O. This paper proposes a hierarchical architecture of erasure code, OI-RAID. OI-RAID consists of two layers of codes, outer layer code and inner layer code. The outer layer code is based on disk grouping and Balanced Incomplete Block Design (BIBD) with skewed data layout to provide efficient parallel I/O of all disks for failure recovery. Inner layer code is deployed within a group of disks. As an example, we deploy RAID5 in both layers and present detailed performance analysis. With RAID5 in both layers, OI-RAID tolerates at least three disk failures meeting practical data availability, and achieves much higher speed up of disk failure recovery than existing approaches, while keeping optimal data update complexity and practically low storage overhead.**

纠删码分层架构

平衡不完全区组设计

Keywords*—*Storage Architecture; Fast Recovery; Data Reliability; RAID

I. Introduction

With the rapid development of data acquisition devices, the volume of digital data increases exponentially [18]. To meet the demand of huge storage capacity and high I/O bandwidth to access data, RAID (*Redundant Arrays of Independent Disks*) [22] or distributed storage systems (e.g., Dynamo [10] and Azure [7]), which aggregate a set of independent disks, are two types of common solutions. Due to the large number of independent devices being deployed, component failure becomes a common event in modern storage systems. So some redundancy must be introduced to modern storage systems to prevent data loss when storage components are failed, where replication and erasure code are two common approaches to provide redundancy.

说一些副本机制和纠删码机制。raid, rdp等。

Replication can be easily deployed in storage systems and provides high I/O bandwidth, but incurs high storage overhead. Erasure code can achieve the same availability of data in a RAID with replication, while only incuring an order of magnitude smaller storage overhead. So it is often deployed in modern storage systems to provide reliability. Typical erasure codes include the RAID5 code tolerating one disk failure, RDP [8], EVENODD [3] and X-code [33] for RAID6 tolerating two concurrent disk failures, STAR code [17] tolerating three concurrent disk failures, and Reed-Solomon codes [19] or CRS(Cauchy Reed-Solomon codes) [4] tolerating an arbitrary number of concurrent disk failures.

Upon disk failures, RAID recovers the failed data to keep the same data availability, and the recovery process should be done as fast as possible to reduce the window size of vulnerability of data permanent loss. Xiang et. al [31, 32] proposed a hybrid recovery scheme to speed up the recovery process, and this scheme reduces about 25% of the data volume for single disk failure recovery of RDP and EVENODD. Xu et. al [34] and Zhu et. al [35] used the similar approach to speed up single disk failure recovery of X-code and STAR code etc. Shen et. al [24] further speeded up the recovery process of single disk failure by reducing the I/O number for the recovery. Tamo et. al [26] proposed a MDS code, Zigzag, which only needs $\frac{1}{r}$ of total data in a RAID to recover a failed disk, if Zigzag is designed to treat r concurrent disk failures. However, if r is large, Zigzag will perform its encoding and decoding in a large Galois field $GF(2^n)$, which will reduce its performance and practicality.

Although all the works above speed up the recovery process, the recovery of a failed disk still takes a very long time. For example, nowadays, terabyte disks are being widely used in large-scale storage systems of various companies. Writing terabytes of data to a commercial disk, e.g., a disk with the volume of four terabytes, will take much more time than traditional disks without responding to any user requests. In practice, most of storage systems should serve user requests during the recovery process, so online recovery becomes necessary as any system shutdown may cause a huge economic loss, and the interference between the recovery process and user requests may degrade both of their performance significantly. Thus, in online recovery scenario, the recovery process can only be executed when the system is idle, and it may last dozens of hours.

把TB级别的数据写入磁盘需要花费很长时间（除去响应用户）。但在实际中，在恢复数据时仍需响应用户请求。

To substantially speed up the recovery process, Wan et. al [29] proposed S^2-RAID, which divides all disks into groups and each disk into logical storage units. To construct a RAID, a fixed number of storage units with the same logical number, one from each group, are grouped together to form a RAID5. To speed up recovery, S^2-RAID uses a skewed layout of the storage units such that a failed disk can be completely recovered by the storage units in different groups in parallel. In particular, if all disks are divided into r groups, S^2-RAID can recover a failed disk by taking only about $1/r$ of the recovery time of conventional RAID5. However, S^2-RAID introduces high storage overhead, and it can only tolerate a single disk failure. Besides, since S^2-RAID divides all disks into groups, and each of which further contains multiple disks, there are usually a large number of disks in the storage system. Thus, S^2-RAID may not be able to provide adequate reliability levels as required by applications due to its limited fault tolerance.

大量磁盘很难保证只坏一个。

In this paper, we propose a new RAID architecture, OI-RAID, which consists of two layers of codes: *outer layer*

978-1-4673-8891-7/16 $31.00 © 2016 IEEE
DOI 10.1109/DSN.2016.15

IEEE computer society

分组然后把磁盘分块

分块后的存储单元组成一个区域

code and *inner layer code*. The main idea is to divide disks in a RAID system into groups and also divide each disk into equal-sized storage units. Some certain storage units in a group further form a *region* so that the RAID system actually consists of a region array. Specially selected regions from different groups are further combined to form a *region set*, and the selection of regions for forming a region set is based on balanced incomplete block designs (abbr. as BIBD) [13]. The term region set is synonymous with the term tuple expressed in the BIBD in the construction of OI-RAID in Section III and IV. Based on the concepts of region and region set, OI-RAID first defines an outer layer code to construct RAID5 among all region sets with skewed data layout, and then defines an inner layer code within each region to further construct RAID5 along the same diagonal. The main benefit of the two-layer coding structure is that the rebuilding process of a failed disk can be processed in parallel. The main advantages of OI-RAID which are also our main contributions in this paper can be summarized as follows.

- We propose a new RAID architecture, OI-RAID, which accesses much less data from each of the surviving disks for failure recovery and provides parallel and conflict-free failure recovery among all surviving disks. So it could achieve dozens of times of speed-up for single disk failure recovery compared to conventional RAID systems.
- OI-RAID tolerates three arbitrary disk failures and some patterns of more than three disk failures, achieving higher reliability than 3-replication which is an accepted industry standard.
- The evaluations show that we just need nearly the theoretically fewest disks to achieve the high speed of data recovery. Meanwhile, compared with conventional RAID systems tolerating multiple disk failures, our scheme reads much less data from surviving disks for failure recovery.

The rest of this paper is organized as follows. Section II reviews the concept of erasure code and two kinds of typical RAID architectures, then motivates the demand of accelerating data recovery. In Section III, we illustrate the construction process of OI-RAID based on an example. In Section IV, we introduce the general design and implementation of OI-RAID. Section V presents the reliability analysis of OI-RAID and its recovery algorithm. We analyze the recovery performance of OI-RAID in Section VI. Section VII reviews related works. Finally, Section VIII concludes the paper.

II. Background and Motivation

In this section, we introduce the concepts of erasure code and review some closely related works which speed up disk recovery by optimizing data layout. We discuss the merits and drawbacks of these methods and then motivate our design.

A. *Erasure codes and RAID5 code*

With an erasure code, k data blocks are encoded into m blocks. The original k data blocks can be decoded from any l $(k \le l \le m)$ out of the m encoded blocks. To better serve user requests, most of existing erasure codes are so-called

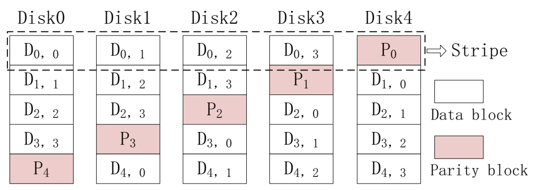

Fig. 1: Layout of the left-symmetric RAID5

systematic codes, i.e., the k original data blocks are included in the m encoded blocks forming a *stripe*. To minimize storage overhead and increase data availability, many erasure codes are designed such that the k original data blocks can be decoded from any k out of the m encoded blocks, which are so-called Maximum Distance Separable (MDS) codes. With a (k, m) systematic MDS code, there are m disks, where k disks store the original data blocks, and $m-k$ disks store the encoded blocks (usually be called parity blocks). A (k, m) MDS code can tolerate any $m-k$ concurrent disk failures.

RAID5 code is a systematical MDS code which tolerates one disk failure. Assume that there are k data blocks $\mathcal{B}_1, \mathcal{B}_2, ..., \mathcal{B}_k$, RAID5 encodes them into a parity block $\mathcal{P}$ simply XOR-summing them, i.e., $\mathcal{P} = \mathcal{B}_1 \oplus \mathcal{B}_2 \oplus ... \oplus \mathcal{B}_k$. Figure 1 shows the layout of a left-symmetric RAID5 consisting of five disks, four data blocks and one parity block in the same row form a stripe. The parity block is encoded from (i.e., the XOR-sum of) the four data blocks in the same stripe. Moreover, all the data blocks and parity blocks are rotationally stored among the five disks for load balancing.

If a disk in Figure 1 fails, we can read all the data/parity blocks in surviving disks, perform the same Exclusive-OR computation in each stripe, and then rebuild the failed disk. However, we should read all the data in the system to reconstruct the data in the failed disk. The reconstruction process will last very long with current commercial disks which store terabytes of data. To speed up the reconstruction process, some data layouts of erasure code, with which the reconstruction process can be performed in parallel, have been proposed. We will introduce two typical ones in the following two subsections.

B. *S^2-RAID*

S^2-RAID [29] is a skewed sub-array RAID architecture in which reconstruction can be done in parallel. Figure 2 shows an example of S^2-RAID5(3,3) which consists of 9 disks $\mathcal{D}_0, \mathcal{D}_1, ..., \mathcal{D}_8$. In this example, disks are divided into three groups and the entire storage space of each disk is further divided into three storage units. All the storage units are labeled in a skewed way and the ones with the same label form a subRAID. There are nine subRAIDs in Figure 2 and each subRAID is deployed with a RAID5 code.

Now suppose that disk $\mathcal{D}_4$ is failed. To rebuild the storage unit labeled 1 in disk $\mathcal{D}_4$, we first read storage units labeled 1 in $\mathcal{D}_1$ and $\mathcal{D}_7$, perform an Exclusive-OR computation, and then write the reconstructed data into multiple spare disks or available space on surviving disks temporarily. In the same way we can rebuild storage units labeled 3 and 8 in $\mathcal{D}_4$. The shaded areas in Figure 2 show that we only read at most one

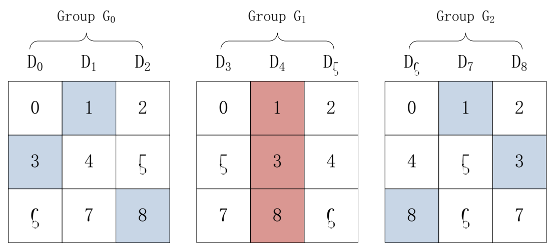

Fig. 2: Layout of S^2-RAID5(3,3)

storage unit from each surviving disk to rebuild the three failed storage units in parallel. Hence we expect that this data layout could achieve three times of speed-up in single disk failure recovery compared to traditional RAID5 data layout.

C. Parity Declustering

校验散布

Before discussing parity declustering [14, 21], we first review some mathematical theories of balanced incomplete block design (abbr. as BIBD) [13], which is the foundation of parity declustering and will also be used in the construction of our OI-RAID.

A (b, v, r, k, λ)-BIBD arranges v distinct objects into b tuples[1] satisfying the following properties: (1) Each tuple contains k objects and each object appears in r tuples, and (2) Each pair of objects are contained in exact λ tuples. The five parameters b, v, r, k, λ satisfy the following two equations:

$$bk = vr \tag{1}$$
$$\lambda(v-1) = r(k-1) \tag{2}$$

Figure 3 shows an example of $(7, 7, 3, 3, 1)$-BIBD, where there are 7 distinct objects and 7 tuples. Each tuple contains 3 objects and each object appears in 3 tuples. Any pair of objects are contained in exactly 1 tuple.

When implementing an erasure code into an RAID system, each disk is divided into many blocks. The erasure code is independently performed in each stripe, where each stripe consists of some blocks with exactly one block out of a disk. In conventional RAID systems, the term *stripe size* is the number of blocks in a stripe. The parity declustering is actually a mapping that allows stripes with stripe size G to be distributed over C disks (C is larger than G). Figure 4 is an example of parity declustering constructed with the BIBD shown in Figure 3. In this example, there are 7 disks $\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_6$, each of which is divided into blocks (we just show the first three blocks in each disk). The blocks with the same number form a stripe and the stripe size is 3. The mapping associates disks with objects, and associates stripes with tuples. For example, Tuple $\mathcal{T}_0$ in Figure 3 is used to lay out the stripe 0 in Figure 4, which consists of the three blocks on $\mathcal{D}_0$, $\mathcal{D}_2$ and $\mathcal{D}_6$, respectively. The other stripes can be constructed similarly.

Now assume that disk $\mathcal{D}_3$ in Figure 4 is failed. This disk contains three blocks which are parts of stripe 1, 3 and 4, respectively. The other six surviving blocks belong to these

[1]The term *tuple* is called *block* in the literatures about block design. But it is easily confused with the common used definition of a block as a contiguous chunk of data in storage. So we use tuple following the work in [14]

```
Tuple T0: 0,2,6  Tuple T1: 0,1,3  Tuple T2: 1,2,4
Tuple T3: 2,3,5  Tuple T4: 3,4,6  Tuple T5: 0,4,5
Tuple T6: 1,5,6
```

Fig. 3: A sample BIBD of $(7, 7, 3, 3, 1)$

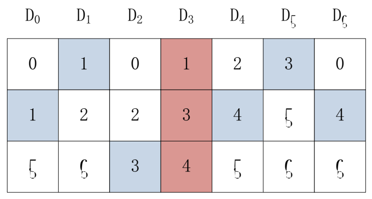

Fig. 4: Layout of parity declustering with G=3,C=7

three stripes are stored on the other six surviving disks with one on each disk. So we can rebuild $\mathcal{D}_3$ in the same way as in S^2-RAID. The shaded areas in Figure 4 show that we only need to read exact one block, which is only one-third of data compared to conventional RAID layout, from each of *all* the surviving disks to reconstruct the three failed blocks in parallel. Theoretically we expected that the recovery speed with parity declustering in Figure 4 is three times as high as conventional layout. In real RAID systems, it does approach but not reach three times because of non successive disk reads which cost more disk seek time.

D. Motivation

As mentioned above, both S^2-RAID and parity declustering substantially speed up the recovery process of single disk failure. S^2-RAID even achieves a higher speed-up with larger group size. But on one hand, the storage system will have more disks with larger size of groups, which leads to more frequent disk failures. So a S^2-RAID of large scale may not be sufficiently reliable because it tolerates only one arbitrary disk failure. On the other hand, in S^2-RAID, all the disks in the same group with the failed disk do not participate in the recovery. So S^2-RAID can not exploit the parallel recovery of all disks in the system. As parity declustering, it needs BIBDs of large scale to achieve high rate speed-up of the recovery process. On one hand, it is difficult to find a BIBD of large scale. On the other hand, parity declustering with BIBDs of large scale will need many disks. It also makes a large scale storage system be not sufficiently reliable because parity declustering also tolerates only one arbitrary disk failure.

Motivated by the strong demand of high speed recovery of a terabyte disk and high reliability of large scale storage system, we aim to use the theoretically fewest disks to achieve fast recovery so as to maintain storage systems of small scale. Meanwhile, we aim to deploy an erasure code which tolerates three concurrent disk failures with small stripe size and low storage overhead so as to guarantee high data reliability. In the following sections, we will present the design of our OI-RAID to achieve these goals.

III. Construction of OI-RAID-An Example

OI-RAID is designed as a hierarchical architecture of two layers. We divide all disks into groups and divide each group of

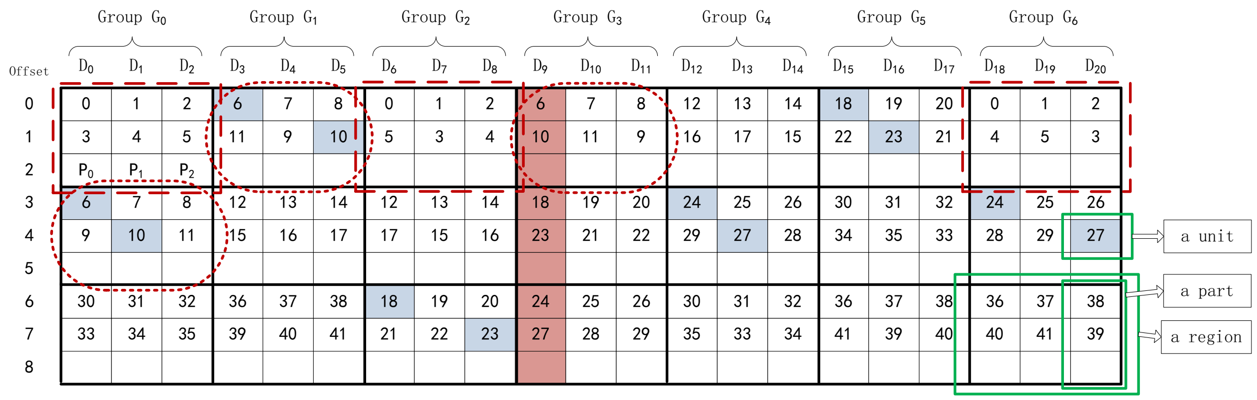

Fig. 5: An example of OI-RAID layout

disks into regions. Then we divide all regions into tuples based on BIBD. The first layer, called outer layer, is a RAID5 based on all regions within a tuple with skewed data layout, and the second layer, called inner layer, is a RAID5 within a region, where a parity block is encoded from all data blocks along the same diagonal. OI-RAID achieves high recovery speed with the outer layer code, and achieves high data reliability with both layers. Before presenting the general design of OI-RAID, we use an example in Figure 5 to explain the main idea of OI-RAID.

用梨子说明两层架构

Figure 5 is based on the $(7,7,3,3,1)$-BIBD showed in Figure 3. In Figure 5, 21 disks are divided into 7 groups $\mathcal{G}_0, \mathcal{G}_1, ..., \mathcal{G}_6$, with 3 disks each. We further divide each of all disks into 3 parts of equal size. The same parts of all disks within a group form a *region*. We define a group of disks as an object in the BIBD. So the $(7,7,3,3,1)$-BIBD is based on set $\mathbf{G} = \{\mathcal{G}_0, \mathcal{G}_1, ..., \mathcal{G}_6\}$. A tuple consists of 3 groups, e.g., tuple $\mathcal{T}_0 = \{\mathcal{G}_0, \mathcal{G}_2, \mathcal{G}_6\}$, which corresponds to three regions, each being the first region from $\mathcal{G}_0, \mathcal{G}_2, \mathcal{G}_6$ respectively, as shown in Figure 5 in the dashed boxes. Also for tuple $\mathcal{T}_1 = \{\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_3\}$, it corresponds to the second region from group $\mathcal{G}_0$ and the first regions from group $\mathcal{G}_1$ and group $\mathcal{G}_3$. There are 7 tuples. Each group is included in 3 tuples, which corresponds to its 3 regions being distributed into 3 different tuples. In Figure 5, each pair of groups are just included in one tuple, which means that for each pair of groups, there are one region out of each group, the two regions are included in just one tuple.

Given the above definition, we explain the designs of outer layer code and inner layer code in the following two subsections.

A. The Design of Outer Layer

To maximize the parallel recovery I/Os of all disks in the system, we divide each part of a disk into three storage units, each unit storing data or parity blocks. As shown in Figure 5, each region is a 3×3 matrix of storage units. The outer layer of OI-RAID is a RAID5 among all regions in the same tuple with skewed data layout. Take tuple $\mathcal{T}_0$ as an example, we label the first six units in the first region of group $\mathcal{G}_0$ as 0, 1, ..., 5, just in row-major order. But for the region of tuple $\mathcal{T}_0$ from group $\mathcal{G}_2$, we circularly shift all labels in the second row to the right on one position. The data layout of the region from group $\mathcal{G}_6$ is accordingly shifted from the region in group $\mathcal{G}_2$.

Now we take the three storage units, one from each group with the same label, as an outer layer code vector with RAID5, i.e, two units store data blocks, and the other one stores the parity block which is the XOR-sum of the two data blocks. Take the three units labeled 0 as an example, we can store two data blocks, say $\mathcal{B}_1$ and $\mathcal{B}_2$, in $\mathcal{D}_0$ and $\mathcal{D}_6$ respectively, and store the parity block $\mathcal{B}_1 \oplus \mathcal{B}_2$ in $\mathcal{D}_{18}$. The outer layer code within other tuples are the same, as shown in Figure 5.

B. The Design of Inner Layer

The code of inner code is within a region as a RAID5. In Figure 5, each region is a 3×3 matrix of units. We store the data blocks in the first two rows and the parity blocks in the third row. Each parity blocks is the XOR-sum of the two data blocks along the same diagonal. Take the first region in Figure 5 as an example, each of the units labeled 0, 1, 2, 3, 4, 5 stores a data block, while each of the units labeled $\mathcal{P}_0$, $\mathcal{P}_1$, $\mathcal{P}_2$ stores a parity block. The parity block in unit $\mathcal{P}_0$ is the XOR-sum of data blocks in unit 1 and 5.

From Figure 5, we can see that if a disk fails, say the shadowed $\mathcal{D}_9$, OI-RAID only reads up to one storage unit from each of the surviving disks in other groups for the recovery (the storage units hold inner parity blocks will be recovered when the storage system is idle). So the recovery process is greatly speeded up.

C. Summary of this Section

In this section, we showed the main idea of OI-RAID with an example. The following is a summary of the advantages of OI-RAID:

- With BIBD, we can design the outer layer code which efficiently exploits the parallel I/Os of all disks for failure recovery.
- Though we design both of the outer layer code and the inner layer code with RAID5 which tolerates only one disk failure, OI-RAID tolerates three arbitrary disk failures. We will formally prove this claim in Section V-A. So OI-RAID achieves high reliability with fast recovery.

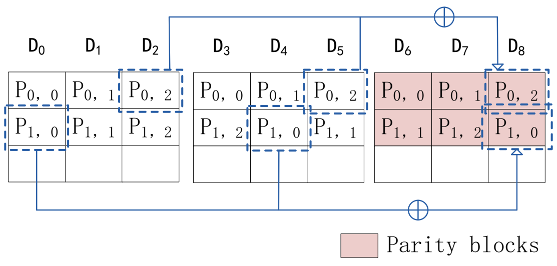
D0 D1 D2 D3 D4 D5 D6 D7 D8
P0, 0 P0, 1 P0, 2 P0, 0 P0, 1 P0, 2 P0, 0 P0, 1 P0, 2
P1, 0 P1, 1 P1, 2 P1, 2 P1, 0 P1, 1 P1, 1 P1, 2 P1, 0
Parity blocks

Fig. 6: An example of the layout of outer layer

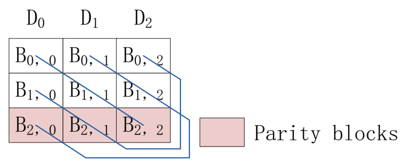

Fig. 7: An example of the layout of inner layer

IV. The Design of General OI-RAID

In this section, we will present the general design of OI-RAID. Its outer layer is based on BIBD and its inner layer is a special RAID5. Suppose that there is a (b, v, r, k, λ)-BIBD, based on which we will present the outer layer design of OI-RAID.

A. The Code Design

To realize an OI-RAID based on a (b, v, r, k, λ)-BIBD, there should be $v \times g$ disks in the RAID, where $g \geq k$ is a prime. We first divide all disks in a RAID into a set of v Groups, each group of g disks. Suppose that the BIBD is built on the set $\boldsymbol{G} = \{\mathcal{G}_0, \mathcal{G}_1, ..., \mathcal{G}_{v-1}\}$ of disk groups. We further divide each disk into r parts of equal size, and divide each part into g storage units. The g parts, one from each disk of a group, form a region. So a region is a $g \times g$ matrix of storage units.

1) The Code Design of Outer Layer: According to the BIBD, all regions of all disks can be divided into b tuples, each tuple of k regions. We design the outer layer code within a tuple. The code design of different tuples are the same. Suppose that we label the $(g-1) \times g$ storage units (the last g storage units which are in last row will be used to store parity blocks of inner layer) of the first region in a tuple as a matrix $\boldsymbol{L}_0 = (\mathcal{P}_{i,j})_{(g-1)\times g}$, then the $(g-1) \times g$ storage units of the l-th region in the same tuple will be labeled as $\boldsymbol{L}_l = (\mathcal{P}_{i,(j-i\times l) modg})_{(g-1)\times g}$, where $1 \leq l \leq k-1$.

We take the k storage units, one from each region, with the same label as a code group, and deploy RAID5 within each code group. Suppose that the k data/parity blocks stored in the k storage units labeled $\mathcal{P}_{i,j}$ are $\mathcal{B}^0_{i,j}, \mathcal{B}^1_{i,j}, ..., \mathcal{B}^{k-1}_{i,j}$ respectively, we can just take $\mathcal{B}^{k-1}_{i,j}$ as the parity block and simply deploy RAID5 code as $\mathcal{B}^{k-1}_{i,j} = \mathcal{B}^0_{i,j} \oplus \mathcal{B}^1_{i,j} \oplus ... \oplus \mathcal{B}^{k-2}_{i,j}$, where Figure 6 is an example.

2) The Code Design of Inner Layer: The inner code is deployed within a region as a RAID5, but just encoded the data blocks along the same diagonal into a parity block. Suppose that the data/parity blocks in a region is a matrix of $(\mathcal{B}_{i,j})_{g\times g}$, we just take the g blocks $\mathcal{B}_{g-1,0}, \mathcal{B}_{g-1,1}, ..., \mathcal{B}_{g-1,g-1}$ in the $(g-1)$-th row as parity blocks and simply deploy RAID5 code as

$$\mathcal{B}_{g-1,g-1-j} = \bigoplus_{i=0}^{g-2} \mathcal{B}_{i,<i-j>_g}, \tag{3}$$

where $<i-j>_g = (i-j) \bmod g$. Please refer to Figure 7 as an example.

B. Implementation of OI-RAID

There are some techniques to construct BIBDs. For example, we can construct BIBDs with perfect difference sets [9] (Singer et al. [25] gave all these perfect difference sets). From each of the perfect difference sets, we can construct a Youden square, which corresponds to a BIBD. Table I shows some examples of BIBDs with the corresponding perfect difference sets.

b	v	r	k	λ	perfect difference set
7	7	3	3	1	0,1,3
13	13	4	4	1	0,1,3,9
21	21	5	5	1	0,1,4,14,16
31	31	6	6	1	0,1,3,8,12,18
51	51	8	8	1	0,1,3,13,32,36,43,52
73	73	9	9	1	0,1,3,7,15,31,36,54,63
91	91	10	10	1	0,1,3,9,27,49,56,61,77,81

TABLE I: A partial list of BIBDs with their associated perfect difference sets

OI-RAID depends on BIBD. For a (b, v, r, k, λ)-BIBD, OI-RAID principally assumes that the storage system has $v \times g$ $(g \geq k)$ disks, which makes OI-RAID can not be implemented on RAIDs with arbitrary number of disks. But in the implementation of OI-RAID, if there are less than $v \times g$ disks, we can add some virtual disks to the system. The virtual disks only store data blocks with value of 0.

V. Reliability and Failure Recovery

In this section, we analyze the data reliability of OI-RAID and present its recovery scheme for failed disks.

A. Reliability

Theorem: OI-RAID tolerates three concurrent disk failures.

Proof: In the following proof, we define that a parity set of a parity block as the set consists of the parity block and all data blocks which are encoded into this parity block. For example, in Figure 8, the parity sets of parity blocks $\mathcal{B}_{0,7}$ and $\mathcal{B}_{2,5}$ are $PS(\mathcal{B}_{0,7}) = \{\mathcal{B}_{0,2}, \mathcal{B}_{0,3}, \mathcal{B}_{0,7}\}$ and $PS(\mathcal{B}_{2,5}) = \{\mathcal{B}_{0,5}, \mathcal{B}_{1,5}, \mathcal{B}_{2,5}\}$, respectively.

There are three cases of three concurrent disk failures in OI-RAID:

1) All the three disks are in the same group.
2) One failed disk is in a group and the other two are in another group.
3) The three failed disks are in three different groups.

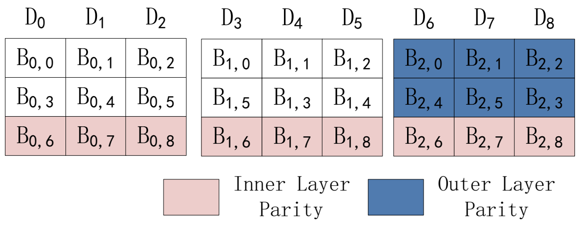

Fig. 8: An example of the layout of a tuple (region set)

We now show that OI-RAID can recover all failed data in any of the above three cases.

Case 1: In the design of outer layer code, each parity block is encoded from data blocks from different regions, where all regions come from different groups of disks. So if all failed disks are in the same group, only one data/parity block in a parity set of the outer layer code is failed. So with RAID5 of outer layer code, all failed data/parity blocks can be reconstructed.

Case 2: In the second case, the only one disk which is failed in a group can be recovered with inner layer code. After the only failed disk in a group being recovered, the other two failed disks can be recovered with outer layer code, just similar to Case 1.

Case 3: In the third case where all failed disks are in different groups. Each of the failed disks can be recovered within its group with inner layer code respectively.

Summarizing the above cases, we conclude that three arbitrary failed disks can be recovered by OI-RAID. ■

We point out that even if all the disks in a group are failed, they can still be recovered with outer layer code. If a single disk in a group is failed, it can be recovered with inner layer code. Hence the region set in Figure 8 can also tolerate some patterns of four or even more disk failures. So OI-RAID achieves higher availability than 3-replication which is an accepted industry standard. Moreover, we can deploy various erasure codes in both layers of OI-RAID to realize different levels of fault tolerance. For example, if we deploy RAID6 code in the inner layer and use RAID5 in the outer layer, then OI-RAID can tolerate all cases of five disk failures and some cases of more than five disk failures.

S^2-RAID and parity declustering are designed along the same lines as OI-RAID. Both of them divide all disks in a RAID into some groups and then perform an erasure code on top of different groups. If we deploy RAID5 in S^2-RAID and parity declustering, both of them tolerate one disk failure. But with another layer of RIAD5 in each group, OI-RAID tolerates three disk failures. So the system reliability of OI-RAID is much higher than both of S^2-RAID and parity declustering.

B. Failure Recovery

In this subsection, we firstly explain how to speed up the recovery process of OI-RAID, and then give its recovery scheme.

If we rebuild a failed disk with outer layer code, we conclude from its design that we only read up to one storage unit from each disk in other groups to reconstruct all units in outer layer parity sets in this disk. For example in Figure 8, if disk $\mathcal{D}_4$ fails, we can reconstruct the unit $\mathcal{B}_{1,1}$ and $\mathcal{B}_{1,3}$ in $\mathcal{D}_4$ with outer layer code as

$$\mathcal{B}_{1,1} = \mathcal{B}_{0,1} \oplus \mathcal{B}_{2,1}$$

$$\mathcal{B}_{1,3} = \mathcal{B}_{0,3} \oplus \mathcal{B}_{2,3}$$

It only reads one storage unit from each of disks $\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_7, \mathcal{D}_8$. The unit $\mathcal{B}_{1,7}$ will be reconstructed when the storage system is idle, because it always store inner layer parities so that it will not be accessed by users.

But if we rebuild a failed disk with inner layer code, we should read all data blocks from the surviving disks within this group. For example in Figure 8, if disk $\mathcal{D}_4$ fails, we can rebuild $\mathcal{D}_4$ with inner layer code as

$$\mathcal{B}_{1,1} = \mathcal{B}_{1,4} \oplus \mathcal{B}_{1,6}$$

$$\mathcal{B}_{1,3} = \mathcal{B}_{1,0} \oplus \mathcal{B}_{1,8}$$

$$\mathcal{B}_{1,7} = \mathcal{B}_{1,2} \oplus \mathcal{B}_{1,5}$$

It reads three storage units from each of the disks $\mathcal{D}_3, \mathcal{D}_5$.

From the above analysis, we should reconstruct a data/parity block with outer layer code prior to inner layer code to leverage device-level parallelism. So if there are less than 4 disks fail in OI-RAID, the failed disks can be recovered by the Algorithm 1.

Algorithm 1 Recovery Algorithm of OI-RAID

```
Require:
    The array of failed storage units $U[0...N]$;
    The flags of each failed storage unit $F[0...N]$;
 1: $f = N$;
 2: for $i = 0$ to $N - 1$ do
 3:     if $F[i] == 0$ and $U[i]$ is the single failed unit in its outer
        layer parity set then
 4:         Reconstruct $U[i]$ in its outer layer parity set;
 5:         $F[i] = 1$;
 6:         $f - -$;
 7:     end if
 8:     if $F[i] == 0$ and $U[i]$ is the single failed unit in its inner
        layer parity set then
 9:         Reconstruct $U[i]$ in its inner layer parity set;
10:         $F[i] = 1$;
11:         $f - -$;
12:     end if
13: end for
14: Repeat Steps 2-13 until $f == 0$;
```

VI. Performance Analysis

Schroeder et al. [23] show that 99.75% of the system failures are single disk failure. Upon a single disk failure, we should recover the failed disk as early as possible to reduce the window size of vulnerability of data loss. Thus, the recovery speed of single disk failure in RAID system is very important. To show the efficiency of OI-RAID, we compare the recovery

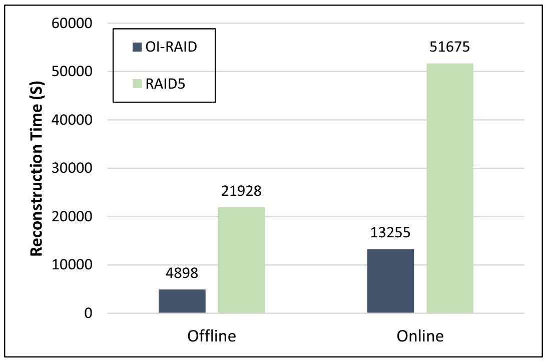

Fig. 9: The reconstruction time of OI-RAID

performance of single disk failure of OI-RAID with that of other two typical RAID architectures, S^2-RAID and parity declustering, which are also based on disk grouping.

We point out that there are also some approaches in the literature proposed to speed up the failure recovery process in RAID systems, such as [24, 26, 31, 32, 34, 35], while all of them do not use the idea of disk grouping. In this work, we evaluate only the performance gain benefited from the disk grouping technique of different RAID architectures, i.e., S^2-RAID, parity declustering, and our OI-RAID. In particular, we compare their recovery performance with the following two metrics.

- **Speed-up Ratio**: It is defined as the ratio of data volume in the failed disk to the maximum data volume read from each surviving disk required for recovery. This metric reflects how much speed improvement the storage system can achieve in a single-disk recovery process. For example, the speed-up ratio in Figure 2 is 3 because the failed disk $\mathcal{D}_4$ contains three storage units and at most one storage unit is read from each surviving disk for the recovery. The speed-up ratio in Figure 4 is also 3 for the same reason.
- **Read Volume Ratio**: It is defined as the total number of data/parity blocks read from all the surviving disks for the recovery of one failed data/parity block. This metric reflects the total volume of data that needs to be read during the recovery process. The read volume ratio in Figure 2 and Figure 4 are both 2 because we need to read two blocks to reconstruct one block.

A. Experimental Validation

Recall that Figure 5 shows an example of OI-RAID which is designed based on the $(7, 7, 3, 3, 1)$-BIBD. Since we read up to one storage unit from each of the surviving disk to reconstruct the six failed storage units in parallel, the speed-up ratio of OI-RAID in Figure 5 is 6. In other words, OI-RAID only takes $\frac{1}{6}$ of the time to recover a failed disk compared to conventional RAID systems.

To show the effectiveness of OI-RAID, we first validate the performance gain of recovery for OI-RAID defined in Figure 5. We conduct experiments on top of a disk system simulator

Parameter	Value
Disk capacity	146.8GB
I/O bus bandwidth	512MBps
XOR bandwidth	1GBps
Sustainable disk transfer rate	99MBps
Rotation speed	15000RPM
Single track seek	0.3ms
Full seek	7.2ms

TABLE II: The parameters of the simulated disk

called DiskSim [6], which is efficient, accurate and highly-configurable for validation purpose. In the simulation, there are 21 disks. Each disk is attached to a disk controller, and every seven disk controllers are under an independent I/O bus. That is, we have three I/O buses, and they all connect to a core switch. The parameters of the simulated disk are obtained from an enterprise hard disk drive [1], and we list them in Table II. We deploy our simulator on a PC with *Intel(R) Core(TM) i3-4130 3.40GHz* CPU, *4GB* DRAM, and *Ubuntu 12.04(32bit)* operating system.

In the simulation, we fix the volume of each disk to 100GB. For comparison, we deploy two RAID schemes on the 21 disks, which have the same read volume ratio. One is the OI-RAID shown in Figure 5, and the other is a variant of conventional RAID5, which contains seven groups of conventional 2+1 RAID5 arrays. We implement a new array controller which is responsible for address mapping, and simulate the offline recovery and online recovery processes on both OI-RAID and RAID5. In the online scenario, the storage system also serves user requests during the recovery process, and the I/O trace we use to simulate user requests is from OLTP applications running at two large financial institutions.

The results are shown in Figure 9. We see that OI-RAID takes only about $\frac{1}{4.5}$ of reconstruction time compared to RAID5 in offline recovery. Recall that the theoretical speed-up ratio of OI-RAID considered in this experiment is 6, so the actual recovery performance of OI-RAID in our experiment is a little worse than the theoretical value, mainly because we write the reconstructed data into the available space on surviving disks in the recovery of OI-RAID, and these write operations also consume some disk bandwidth and thus increase the recovery time. Moreover, OI-RAID takes about $\frac{1}{3.9}$ of reconstruction time in online recovery, because serving user requests intensifies I/O competition.

B. Speed-up Ratio

In this subsection, we evaluate the recovery performance of OI-RAID, S^2-RAID and parity declustering in terms of speed-up ratio. We first formally derive the speed-up ratio of the three schemes, and then evaluate their performance via numerical analysis.

S^2-RAID: Suppose that a S^2-RAID layout introduced in Section II-B consists of l groups, each of which contains g disks, then there are $l \times g$ disks in total. To reconstruct g blocks under single disk failure, we only need to read one block from each disk in other groups. Thus, the speed-up ratio of S^2-RAID is g.

Parity Declustering: Suppose that we deploy parity declustering layout with a (b, v, r, k, λ)-BIBD introduced in

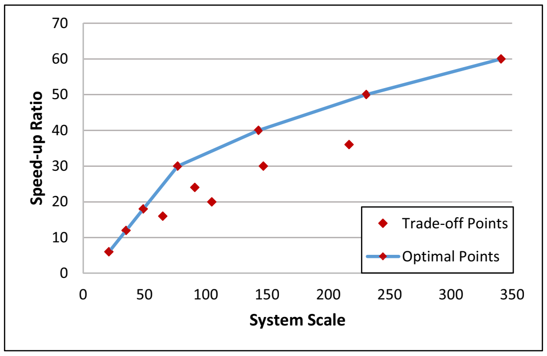

Fig. 10: The speed-up ratio of OI-RAID

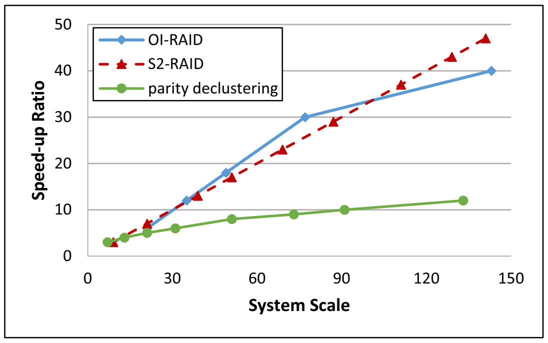

Fig. 11: Comparison of speed-up ratio.

Section II-C, then each disk is included in r tuples. Thus, all data/parity blocks in a failed disk can be reconstructed by r groups of disks. Note that each pair of disks are exactly in λ tuples, so each pair of data/parity blocks participate in the construction of λ failed blocks. To reconstruct r data blocks in a failed disk, we need to read exact λ data/parity blocks from each surviving disk. That is, the speed-up ratio of parity declustering is $\frac{r}{\lambda}$.

OI-RAID: Suppose we build OI-RAID with a (b, v, r, k, λ)-BIBD in Table I, where each pair of objects are included in exact one tuple. Accordingly, in the outer layout, each pair of disk groups participate in the construction of exact one region set. Since each disk group contains r regions, the outer layout contributes r times of speed-up. We note that in OI-RAID, all storage units in the last row of regions store parity blocks, so they will never get accessed by users. Other storage units are encoded in the outer layer so that they may contain data blocks or parity blocks. During the recovery process, we treat all the storage units encoded in the outer layer as data units. Thus, when a single disk fails, we should rebuild the units encoded in the outer layer immediately, while the units which store inner layer parity blocks could be rebuilt later when the storage system is idle. Therefore, for each region set, we can read one storage unit from each disk in other regions to rebuild $g - 1$ storage units in parallel without conflict. As a result, the whole OI-RAID could achieve a speed-up ratio of $r \times (g - 1)$.

Numerical Results: We first evaluate the speed-up ratio of OI-RAID. Our OI-RAID is based on (b, v, r, k, λ)-BIBDs with group size g, which denotes the number of disks in a group. With different settings of (b, v, r, k, λ) and g, OI-RAID achieves different speed-up ratios. Here g is a prime number, and we limit it in the range of $3 \leq g \leq 11$, mainly because the stripe size of RAID5 in the inner layer will be large for a large g, and large stripe size will make inner layer parity blocks be updated frequently.

Figure 10 shows the speed-up ratios at different system scales which are based on the BIBDs in Table I and the parameter g. In particular, each point in the figure shows the speed-up ratio of OI-RAID under a particular setting based on BIBDs and g. Note that we always aim to achieve a high speed-up ratio with as few disks as possible, so in Figure 10, the optimal curve shows the best choices of (b, v, r, k, λ) and g, which can achieve high speed-up ratio with as few disks as possible. We see that OI-RAID achieves a speed-up ratio of 60 with less than 350 disks.

Now we compare the speed-up ratio of OI-RAID with that of S^2-RAID and parity declustering. The results are shown in Figure 11, where the speed-up ratios of OI-RAID are from the optimal curve in Figure 10, those of parity deculstering are based on the BIBDs in Table I, and those of S^2-RAID are derived with group number of 3. From Figure 11, we see that the increase of the speed-up ratio of parity declustering is more flat compared to OI-RAID and S^2-RAID as the system scale increases. So it is inefficient to construct a large-scale storage system with high speed-up ratio using only parity declustering. The reason is that parity declustering is based on BIBDs, its speed-up ratio equals to $\frac{r}{\lambda}$, while a larger system scale, namely v, may not bring a larger r as well as $\frac{r}{\lambda}$. However, even though our OI-RAID is also based on BIBDs, its speed-up ratio equals to $r \times (g - 1)$, so it achieves much higher speed-up ratio than parity declustering at the same system scale. In particular, OI-RAID achieves almost the same speed up ratio with S^2-RAID. We also see that the speed-up ratio of OI-RAID is larger than that of S^2-RAID when the system scale is smaller than one hundred. The main reason is when the system scale of OI-RAID is small, with a larger g and a smaller r, the skewed data layout could contribute more to the speed-up ratio. However, since we limit the parameter g as $g \leq 11$, when the system scale increases, the increase of r is finally limited as in parity declustering. So the speed-up ratio of OI-RAID is smaller than that of S^2-RAID when the system scale is larger than one hundred. Nevertheless, recall that S^2-RAID can only tolerate a single disk failure, so it may not satisfy the reliability requirement in applications. On the contrary, OI-RAID tolerates three disk failures, so it provides both high speed-up ratio and high reliability.

C. Read Volume Ratio

In this section, we evaluate the read volume ratio, which reflects the total amount of data that needs to be read during the recovery process, of different RAID schemes, including S^2-RAID, parity declustering, MDS codes, and OI-RAID. Again, we first derive the closed-form solutions, and then compare different schemes via numerical analysis.

S^2-RAID: In a S^2-RAID layout which consists of l groups, each of which contains g disks, there are l storage units in a

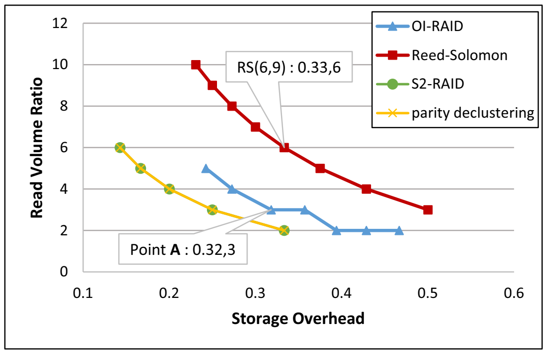

Fig. 12: Comparison of the read volume ratio

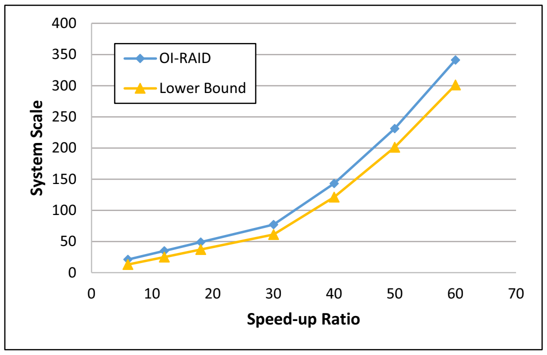

Fig. 13: The lower bound of OI-RAID

subRAID. Note that RAID5 is deployed within each subRAID, so the read volume ratio of S^2-RAID is $l-1$.

Parity Declustering: In a parity declustering layout deployed with a (b,v,r,k,λ)-BIBD, a tuple contains k disks, which are grouped to form RAID5, so the read volume ratio of parity declustering is $k-1$.

MDS code: For a (k,m) systematic MDS code tolerating $m-k$ arbitrary disk failures, e.g., Reed-Solomon code $RS(k,m)$, there are k data blocks and $m-k$ parity blocks in a stripe. Since any k of the m blocks could reconstruct the original data blocks, the read volume ratio of $RS(k,m)$ is k.

OI-RAID: In an OI-RAID deployed with (b,v,r,k,λ)-BIBD, we only use the outer-layer parities to rebuild data under single disk failure. Since RAID5 is deployed in the outer layer and there are k blocks in each parity set, the read volume ratio of OI-RAID is $k-1$.

Numerical Results: The comparison of the read volume ratio among OI-RAID ,S^2-RAID, parity declustering and Reed-Solomon code is shown in Figure 12. In this figure, we can see that S^2-RAID and parity declustering achieve the same performance, mainly because the read volume ratio is decided by the deployed code, which is RAID5 code for both S^2-RAID and parity declustering. We also see that the read volume ratio of RAID5 is smaller than OI-RAID when they cost the same storage overhead, but we emphasize that OI-RAID has higher reliability than RAID5 as it tolerates three disk failures while RAID5 can only tolerate one arbitrary disk failure. For fair comparison, all the settings of Reed-Solomon code shown in the figure can tolerate three disk failures, so they have the same fault tolerance as OI-RAID. We point out that the curve representing OI-RAID only contains the optimal trade-off points which can achieve a lower read volume ratio while incurring a lower storage overhead. From this figure, we see that the read volume ratio of OI-RAID is evidently smaller than that of the Reed-Solomon code. This implies that OI-RAID requires less network bandwidth and fewer I/Os to perform single-disk recovery than Reed-Solomon code with the same storage overhead. In particular, compared with RS(6,9), which is also used in GFS II in Google [12], the read volume ratio of OI-RAID under the setting defined by point A is reduced from 6 to 3. That is, OI-RAID under the setting at point A saves 50% of reconstruction cost.

D. Lower Bound on System Scale

To achieve the maximum speed-up ratio and the minimum read volume ratio, a storage system may need a large number of disks. To evaluate the disks cost of a storage system, we define

- **Lower Bound** $L_{bound}(\alpha,\beta)$: It is defined as the minimum number of disks needed to construct a storage system that can achieve a speed-up ratio of α and a read volume ratio of β.

Suppose that the volume of data in a failed disk is Ω. For a RAID which reaches the lower bound $L_{bound}(\alpha,\beta)$, the total volume of data needs to be read for reconstruction is $\beta\times\Omega$. Because the speed-up ratio is α, the maximum volume of data we should read from each surviving disk is $\Omega\div\alpha$. As a result, we need at least $\lceil(\beta\times\Omega)\div(\Omega\div\alpha)=\alpha\times\beta\rceil$ disks to read out all the data. Note that α may not be an integer. Containing the failed disk itself, the lower bound $L_{bound}(\alpha,\beta)$ can be derived as follows.

$$L_{bound}(\alpha,\beta)=\lceil\alpha\times\beta\rceil+1. \qquad (4)$$

Recall that there are $v\times g$ disks in an OI-RAID mentioned above. According to Equation (2), we have $v=r(k-1)+1$ since $\lambda=1$, so an OI-RAID consists of $rg(k-1)+g$ disks. On the other hand, $L_{bound}(r(g-1),k-1)$ is equal to $r(g-1)(k-1)+1$. Therefore, OI-RAID needs $r(k-1)+g-1$ more disks than the lower bound. We emphasize that the parameters r, k, and the group size g is relatively small, e.g., $r=k\le 6$ and $g\le 11$. So OI-RAID needs only a few more disks than the theoretical minimum, so as to achieve the highest speed up of single-disk recovery.

OI-RAID: In Figure 13, we show the optimal curve of OI-RAID and the associated lower bound. We can see that the system scale of OI-RAID is just a little bit larger than the lower bound for achieving the same speed-up ratio. Even in the worst case, OI-RAID requires $40/301=13.3\%$ more disks than the theoretical lower bound so as to achieve the same speed-up ratio, i.e., 60× of speed-up in single disk failure recovery compared to conventional RAID systems.

S^2-RAID: The S^2-RAID5 can tolerate just one arbitrary disk failure, so it can not satisfy the reliability requirement of large-scale storage systems. To solve this problem, we

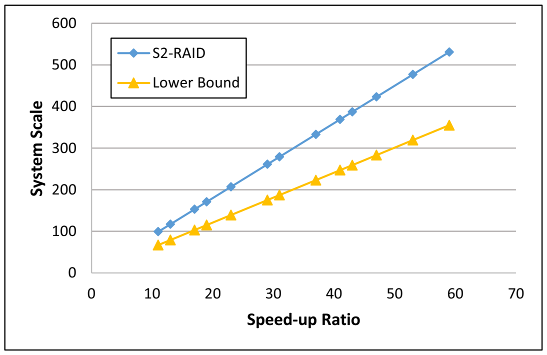

Fig. 14: The lower bound of S^2-RAID deployed with RS(6,9) code

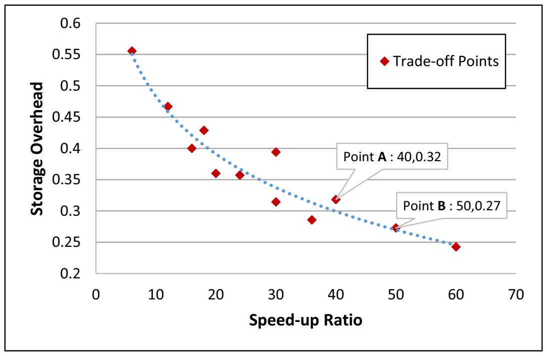

Fig. 15: The storage overhead of OI-RAID

deploy Reed-Solomon code in S^2-RAID. Figure 14 shows the relationship between the speed-up ratio and the system scale when we deploy RS(6,9) code into S^2-RAID layout. In this figure, the number of disks that S^2-RAID needs is far more than the lower bound. In the worst case, S^2-RAID needs $176/355 = 49.6\%$ more disks than the lower bound. Due to the extremely large system scale meaning a large disks cost, it is impractical to deploy Reed-Solomon code in S^2-RAID. Thus, there is a tradeoff between the high speed-up ratio and high reliability for S^2-RAID, while OI-RAID can achieve both of them.

Parity Declustering: As stated in Section VI-B and Section VI-C, the speed-up ratio of parity declustering is $\frac{r}{\lambda}$ and its read volume ratio is exactly $k-1$. According to Equation (2), we have $v = \frac{r(k-1)}{\lambda} + 1$. According to Equation (4), we can calculate the lower bound of number of disks required by parity declustering as $L_{bound}(\frac{r}{\lambda}, k-1) = \frac{r(k-1)}{\lambda} + 1$. The result implies that parity declustering reaches the lower bound on the number of disks, which benefits from the features of BIBD. These features also makes OI-RAID require only a fewer more disks than the lower bound because the outer layer of OI-RAID is also based on BIBDs.

E. Storage Overhead of OI-RAID

We point out that some trade-off points on the optimal curve in Figure 10 may not meet the needs due to its high storage overhead. Therefore, we further show the relationship

code	overhead	reliability	storage system
RS(6,9)	0.33	3	GFS II
RS(10,14)	0.29	4	HDFS-RAID

TABLE III: Storage overhead of two typical Reed-Solomon codes

between the speed-up ratio and the storage overhead in Figure 15. Assume that the system contains n storage units and m of them are parity units, then the storage overhead is defined as m/n here. In Figure 15, we see that the storage overhead of OI-RAID under all appropriate settings are between 0.2 and 0.6, and the setting with higher speed-up ratio usually incurs lower storage overhead.

For comparison, we further consider two typical codes in Table III. As mentioned before, RS(6,9) is used in GFS II in Google and it can tolerate three arbitrary disk failures. RS(10,14) is used in HDFS-RAID in Facebook [5] and it can tolerate four arbitrary disk failures. In particular, the trade-off point **A** in Figure 15 represents the OI-RAID system defined by (13,13,4,4,1)-BIBD with g being equal to 11. This setting costs a little bit lower storage overhead than RS(6,9), and achieves a speed-up ratio of 40. That is, it only takes $\frac{1}{40}$ of time to recover a failed disk. Since the system scale is 143, it is practical to build a storage system by using the parameters at this point. Similarly, point **B** represents the OI-RAID system defined by (21,21,5,5,1)-BIBD with g being equal to 11. This setting costs lower storage overhead than RS(10,14) while achieving a speed-up ratio of 50.

F. Update Complexity of OI-RAID

In Figure 8, we show an example of a tuple in OI-RAID. If we update a data block, say $\mathcal{B}_{0,0}$, the parity blocks $\mathcal{B}_{2,0}$ and $\mathcal{B}_{0,8}$ should be updated because the $\mathcal{B}_{0,0}$ is involved in $PS(\mathcal{B}_{2,0})$ and $PS(\mathcal{B}_{0,8})$. After $\mathcal{B}_{2,0}$ being updated, $\mathcal{B}_{2,8}$ should also be updated. In total, any data block update will cause four updates in OI-RAID. Since OI-RAID tolerates three arbitrary disk failures, the update overhead of OI-RAID is optimum.

G. Summary

To summarize the results presented before, we list some typical configurations for S^2-RAID, parity declustering, and OI-RAID, as well as their speed-up ratios, read volume ratios and the associated lower bound on the number of disks required for system construction. In Table IV, S$^2(\alpha,\beta)$ represents a S^2-RAID layout which consists of α groups, each of which contains β disks. PD(α,β,λ) represents a parity declustering layout deployed with a $(\alpha,\alpha,\beta,\beta,\lambda)$-BIBD. OI$(\alpha,\beta,\lambda,g)$ represents an OI-RAID deployed with $(\alpha,\alpha,\beta,\beta,\lambda)$-BIBD and each disk group consists of g disks. Finally, system scale means the number of disks in the storage system.

From Table IV, we can have the following conclusions.

- OI-RAID could utilize fewer disks than S^2-RAID to achieve an even higher speed-up ratio. For example, S$^2(3,17)$ utilizes 51 disks to achieve a speed-up ratio

layout	speed-up ratio(α)	read volume ratio(β)	L_{bound} (α, β)	system scale	storage over-head
$S^2(3,3)$	3	2	7	9	0.33
$S^2(5,5)$	5	4	21	25	0.2
$S^2(3,17)$	17	2	35	51	0.33
PD(7,3,1)	3	2	7	7	0.33
PD(21,5,1)	5	4	21	21	0.2
PD(51,8,1)	8	7	51	51	0.13
OI(7,3,1,3)	6	2	13	21	0.56
OI(7,3,1,7)	18	2	37	49	0.43
OI(13,4,1,11)	40	3	121	143	0.32
OI(21,5,1,11)	50	4	201	231	0.27

TABLE IV: A list of typical layouts

of 17, but OI(7,3,1,7) utilizes 2 fewer disks to achieve a higher speed-up ratio of 18.

- Parity declustering achieves the lower bound on the system scale, while the speed-up ratio of the system constructed with the lower bound number of disks is also small. For example, PD(51,8,1) utilizes only 51 disks, but the speed-up ratio is only 8. By comparison, OI(7,3,1,7) utilizes 49 disks to achieve a speed-up ratio of 18.
- OI-RAID could achieve a high speed-up ratio while keeping a low read volume ratio. For example, the read volume ratios of OI(21,5,1,11) and PD(21,5,1) are both 4, but OI(21,5,1,11) could achieve a speed-up ratio of 50, while PD(21,5,1) only achieves a speed-up ratio of 5.
- OI-RAID achieves much higher reliability (tolerating three disk failures) than S^2-RAID (tolerating only one disk failure) with only a small storage overhead. For example, OI(7,3,1,7) only incurs 10% larger storage overhead than $S^2(3, 17)$, while still guarantees the same read volume ratio and an even higher speed-up ratio.

In practical scenarios, we should chose an appropriate configuration to construct OI-RAID by overall consideration according to actual demand.

VII. Related Work

There have been several approaches [15, 20, 27, 30, 31, 35] proposed to speed up the recovery of disk failures. Some of them speed up the recovery through exploiting workload characteristics. For example, Tian et al. [27] proposed a popularity-based mutlti-threaded algorithm which reconstructs the data in frequently accessed areas prior to that in infrequently accessed areas to exploit access locality. This method shortens reconstruction time and alleviates system access performance degradation. Some other approaches speed up the recovery by minimizing the total volume of data that need to be read from the surviving disks. For example, Xiang et al. [31] proposed an optimal hybrid recovery method which uses both row parity and diagonal parity during recovery process to minimize the number of disk reads in RDP code storage systems. Zhu et al. [35] used a replace recovery algorithm to achieve near-optimal single-disk recovery performance in general XOR-based erasure code systems. However, all these methods optimize recovery algorithm based on the existing storage systems with concrete data layout. They speed up the recovery below 2 times than that of traditional algorithms. Hence rebuilding a terabyte disk using these optimized recovery algorithms will still take a a lot of time.

There are also multiple works that focus on optimizing the data layout [2, 11, 14, 16, 21, 28, 29]. These methods usually achieve greater recovery speed improvement because recovery process utilizes more disks to rebuild a single failed disk in parallel. For example, as mentioned in section II, Wan et al. [29] proposed a skewed sub-array RAID architecture called S^2-RAID. They divide all the disks into several groups and divide each physical disk into several logic storage units. Each subRAID consists of certain logic storage units from different groups so that recovery can be done in parallel. However, it is easy to find that the surviving disks in the same group with the failed disk do nothing for the recovery. When the group number is large, to achieve higher speed-up during recovery the storage system needs many more disks that are actually not necessary. Moreover, a S^2-RAID5 system is not dependable because it can tolerate just one arbitrary disk failure. To tolerate multi-disk failure the S^2-RAID architecture requires more disk groups to accommodate parity disks and thus makes the system size larger and larger. Muntz et al. [21] proposed a parity declustering layout organized by balanced incomplete block designs. As there is no general techniques to directly construct a small block design, the parity declustering layout does not have the ability to provide dozens of times of speed-up in recovery. Outside these two methods, Huang et al. [16] deployed a local reconstruction code which reduces the bandwidth and I/Os required for recovery, and the code was deployed in the Windows Azure Storage. Rouayheb et al. [11] proposed a series of regeneration codes which consist of an outer MDS code and an inner repetition code. These codes cost minimum bandwidth in recovery. Tsai et al. [28] proposed a new variant of RAID organization to improve storage efficiency and reduce the performance degradation when disk failure occurs.

All these existing methods speed up the recovery in a limited range. Different from them, our OI-RAID provide a relatively large speed-up ratio, while maintaining high reliability. In the OI-RAID system, we can rebuild a terabyte disk in a very short period of time.

VIII. Conclusion

In this paper, we proposed a new RAID architecture, OI-RAID, which achieves fast recovery and high reliability so as to make it feasible to rebuild a terabyte disk. OI-RAID consists of two layers of layouts. The outer layout is organized by using BIBDs and helps to speed up the recovery of disk failures. The inner layer is RAID5s encoded in same diagonal and helps for the reliability of the whole storage system. OI-RAID can tolerate up to three arbitrary disk failures and some patterns of more than three disk failures. Since OI-RAID can achieve a relatively low read volume ratio, it costs smaller network overhead than traditional three-failure tolerable codes to recover a failed disk. Performance evaluations show that we could use nearly the theoretically fewest disks to build OI-RAID with a relatively small storage overhead, while achieving up to 60 times of speed-up in single-disk recovery than conventional RAID systems. Therefore, OI-RAID provides an effective option to build a storage system with fast data recovery, high reliability, low reconstruction cost and storage overhead, as well as an acceptable system scale.

Acknowledgments

This work was supported in part by National Nature Science Foundation of China (61379038 and 61303048), Anhui Provincial Natural Science Foundation (1508085SQF214), Huawei Innovation Research Program.

References

[1] *Cheetah 15K.5 Fibre Channel 146-GB Hard Drive ST3146855FC Product Mannual*. Seagate Inc.

[2] G. A. Alvarez, W. A. Burkhard, L. J. Stockmeyer, and F. Cristian. Declustered disk array architectures with optimal and near-optimal parallelism. In *ACM SIGARCH Computer Architecture News*, volume 26, pages 109–120, 1998.

[3] M. Blaum, J. Brady, J. Bruck, and J. Menon. Evenodd: An efficient scheme for tolerating double disk failures in raid architectures. *IEEE Transactions on Computers*, 44:192–202, 1995.

[4] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. An xor-based erasure-resilient coding scheme. 1999.

[5] D. Borthakur, R. Schmidt, R. Vadali, S. Chen, and P. Kling. Hdfs raid. In *Hadoop User Group Meeting*, 2010.

[6] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger. The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101). *Parallel Data Laboratory*, page 26, 2008.

[7] B. Calder, J. Wang, A. Ogus, et al. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157, 2011.

[8] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row-diagonal parity for double disk failure correction. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, pages 1–14, 2004.

[9] G. M. Cox. Enumeration and construction of balanced incomplete block configurations. *The Annals of Mathematical Statistics*, 11:72–85, 1940.

[10] G. DeCandia, D. Hastorun, M. Jampani, et al. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220, 2007.

[11] S. El Rouayheb and K. Ramchandran. Fractional repetition codes for repair in distributed storage systems. In *48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1510–1517, 2010.

[12] A. Fikes. Storage architecture and challenges. *Talk at the Google Faculty Summit*, 2010.

[13] M. Hall. *Combinatorial theory*, volume 71. John Wiley & Sons, 1998.

[14] M. Holland and G. A. Gibson. *Parity declustering for continuous operation in redundant disk arrays*, volume 27. ACM, 1992.

[15] M. Holland, G. A. Gibson, and D. P. Siewiorek. Fast, on-line failure recovery in redundant disk arrays. In *The Twenty-Third International Symposium on Fault-Tolerant Computing*, pages 422–431, 1993.

[16] C. Huang, H. Simitci, Y. Xu, A. Ogus, et al. Erasure coding in windows azure storage. In *USENIX Annual Technical Conference*, pages 15–26, 2012.

[17] C. Huang and L. Xu. Star: An efficient coding scheme for correcting triple storage node failures. *IEEE Transactions on Computers*, 57:889–901, 2008.

[18] P. Lyman and H. Varian. How much information 2003? 2004. http://groups.ischool.berkeley.edu/archive/how-much-info-2003.

[19] R. J. McEliece and D. V. Sarwate. On sharing secrets and reed-solomon codes. *Communications of the ACM*, 24:583–584, 1981.

[20] J. Menon and D. Mattson. Distributed sparing in disk arrays. In *Compcon Spring'92. Thirty-Seventh IEEE Computer Society International Conference*, pages 410–421, 1992.

[21] R. R. Muntz and J. C. Lui. *Performance analysis of disk arrays under failure*. Computer Science Department, University of California, 1990.

[22] D. A. Patterson, G. Gibson, and R. H. Katz. *A case for redundant arrays of inexpensive disks (RAID)*, volume 17. ACM, 1988.

[23] 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 *FAST*, volume 7, pages 1–16, 2007.

[24] Z. Shen, J. Shu, and Y. Fu. Seek-efficient i/o optimization in single failure recovery for xor-coded storage systems. In *Proceedings of the 34th International Symposium on Reliable Distributed Systems*, pages 228–237, 2015.

[25] J. Singer. A theorem in finite projective geometry and some applications to number theory. *Transactions of the American Mathematical Society*, 43:377–385, 1938.

[26] I. Tamo, Z. Wang, and J. Bruck. Zigzag codes: Mds array codes with optimal rebuilding. *IEEE Transactions on Information Theory*, 59:1597–1616, 2013.

[27] L. Tian, D. Feng, H. Jiang, K. Zhou, L. Zeng, J. Chen, Z. Wang, and Z. Song. Pro: A popularity-based multi-threaded reconstruction optimization for raid-structured storage systems. In *FAST*, volume 7, pages 301–314, 2007.

[28] W.-J. Tsai and S.-Y. Lee. Multi-partition raid: A new method for improving performance of disk arrays under failure. *The Computer Journal*, 40:30–42, 1997.

[29] J. Wan, J. Wang, Q. Yang, and C. Xie. S2-raid: A new raid architecture for fast data recovery. In *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–9, 2010.

[30] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao. Workout: I/o workload outsourcing for boosting raid reconstruction performance. In *FAST*, volume 9, pages 239–252, 2009.

[31] L. Xiang, Y. Xu, J. Lui, and Q. Chang. Optimal recovery of single disk failure in rdp code storage systems. In *ACM SIGMETRICS Performance Evaluation Review*, volume 38, pages 119–130, 2010.

[32] L. Xiang, Y. Xu, J. Lui, Q. Chang, Y. Pan, and R. Li. A hybrid approach to failed disk recovery using raid-6 codes: algorithms and performance evaluation. *ACM Transactions on Storage*, 7:11, 2011.

[33] L. Xu and J. Bruck. X-code: Mds array codes with optimal encoding. *IEEE Transactions on Information Theory*, 45:272–276, 1999.

[34] S. Xu, R. Li, P. P. Lee, Y. Zhu, L. Xiang, Y. Xu, and J. Lui. Single disk failure recovery for x-code-based parallel storage systems. *IEEE Transactions on Computers*, 63:995–1007, 2014.

[35] Y. Zhu, P. P. Lee, Y. Xu, Y. Hu, and L. Xiang. On the speedup of recovery in large-scale erasure-coded storage systems. *IEEE Transactions on Parallel and Distributed Systems*, 25:1830–1840, 2014.