A Hierarchical RAID Architecture Towards Fast Recovery and High Reliability

Yongkun Li[1,2], Neng Wang[1], Chengjin Tian[1], Si Wu[1], Yueming Zhang[1], and Yinlong Xu[1,3]

Abstract—Disk failures are very common in modern storage systems due to the large number of inexpensive disks. As a result, it takes a long time to recover a failed disk due to its large capacity and limited I/O. To speed up the recovery process and maintain a high system reliability, we propose a hierarchical code architecture with erasure codes, OI-RAID, which consists of two layers of codes, outer layer code and inner layer code. Specifically, the outer layer code is deployed with disk grouping technique based on Balanced Incomplete Block Design (BIBD) or complete graph with skewed data layout to provide efficient parallel I/O of all disks for fast failure recovery, and the inner layer code is deployed within each group of disks to provide high reliability. As an example, we deploy RAID5 in both layers to achieve fault tolerance of at least three disk failures, which meets the requirement of data availability in practical systems, as well as much higher speed up ratio for disk failure recovery than existing approaches. Besides, OI-RAID also keeps the optimal data update complexity and incurs low storage overhead in practice.

Keywords—RAID; Fast Recovery; Data Reliability; BIBD; Complete Graph

1 Introduction

With the rapid development of data acquisition devices, the volume of digital data increases exponentially. To meet the demand of huge storage capacity and high I/O bandwidth, RAID (*Redundant Arrays of Independent Disks*) [19] or distributed storage systems (e.g., Dynamo [8] and Azure [5]), which aggregate a set of independent disks, are two types of common solutions. Due to the large number of devices being deployed, component failure becomes a common event in modern storage systems. Thus, data redundancy must be introduced to prevent data loss when component failure happens.

Replication and erasure code are two common approaches to provide data redundancy. Replication can be easily deployed in storage systems and provides high I/O bandwidth, but it incurs high storage overhead. On the other hand, erasure code can achieve the same reliability in a RAID with replication, while it only incurs an order of magnitude smaller storage overhead. Thus, it is widely used in modern storage systems to provide high reliability. Typical erasure codes include RAID5 code which tolerates one disk failure, RDP [6], EVENODD [2] and X-code [30] for RAID6 tolerating two concurrent disk failures, STAR code [15] tolerating three disk failures, and RS (Reed-Solomon codes) [16] or CRS (Cauchy Reed-Solomon codes) [3] tolerating an arbitrary number of concurrent disk failures.

Upon disk failures, RAID recovers the lost 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 permanent data loss. Xiang et al. [28, 29] proposed a hybrid recovery scheme to speed up the recovery process, which reduces about 25% of the data volume for single disk failure recovery for RDP and EVENODD. Xu et al. [31] and Zhu et al. [32] used the similar approach to speed up single disk failure recovery for X-code and STAR code etc. Shen et al. [21] further speeded up the recovery process of single disk failure by reducing the number of I/Os during the recovery. Tamo et al. [23] 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 aim to 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. Writing terabytes of data to a commercial disk, e.g., a disk with 4TB, will take much more time than traditional disks without responding to any user requests. In practice, most 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 the performance significantly. Thus, in online recovery scenario, the recovery process can only be executed when the system is idle, and as a result, it may last dozens of hours.

To substantially speed up the recovery process, Wan

• [1]*The authors are with the School of Computer Science and Technology, University of Science and Technology of China.*
[2]*The author is with Collaborative Innovation Center of High Performance Computing, National University of Defense Technology.*
[3]*The author is with AnHui Province Key Laboratory of High Performance Computing.*
Emails: ykli@ustc.edu.cn, {campnou, wusi, mingfire}@mail.ustc.edu.cn, ylxu@ustc.edu.cn. Corresponding author: Yinlong Xu

et al. [26] proposed S^2-RAID, which can recover a failed disk by taking only about $1/r$ of the recovery time compared to conventional RAID5 if all disks are divided into r groups. However, S^2-RAID introduces high storage overhead and only tolerates single disk failure. Besides, since S^2-RAID divides all disks into groups, and each of which contains multiple disks, it requires a large number of disks. Thus, S^2-RAID may not be able to provide adequate reliability levels required by applications due to its limited fault tolerance.

In this paper, we propose a hierarchical RAID architecture, OI-RAID, which consists of two layers of codes: *outer layer code* and *inner layer code*. The main advantages of OI-RAID, which are also our main contributions in this paper, can be summarized as follows.

- OI-RAID 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 arbitrary three disk failures and some patterns of more than three disk failures by deploying RAID5 code in both layers, achieving higher reliability than 3-replication, which is an accepted industry standard. Clearly, we can also achieve higher level of fault tolerance by deploying other codes in both layers.
- OI-RAID needs nearly the theoretically fewest disks to achieve a certain speedup in data recovery. Meanwhile, compared with conventional RAID systems tolerating multiple disk failures, OI-RAID reads much less data from surviving disks for recovery.
- We also evaluate the performance of OI-RAID in real systems, including the recovery time in both offline and online scenarios, as well as the user response time in both normal model and failure mode.

The rest of this paper is organized as follows. We first review related works in Section 2, then provide background on erasure code and two typical RAID architectures and motivate OI-RAID in Section 3. We illustrate the construction of OI-RAID with an example in Section 4, and introduce the general design in Section 5. In Section 6, we present the construction of OI-RAID based on complete graph. In Section 7, we present the reliability analysis and the recovery algorithm of OI-RAID. We conduct numerical analysis to show the performance of OI-RAID in Section 8 and further evaluate the performance in real systems in Section 9. Finally, Section 10 concludes the paper.

2 Related Work

There have been several approaches proposed to speed up the recovery of disk failures [13, 17, 24, 27, 28, 32]. Some of them speed up the recovery through exploiting workload characteristics. For example, Tian et al. [24] 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. [28] 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. [32] 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 only speed up the recovery below two times than that of traditional algorithms. Hence rebuilding a terabyte disk using these optimized recovery algorithms still takes a very long time.

Besides, various works that focus on optimizing the data layout are also proposed [1, 9, 12, 14, 18, 25, 26], and these methods usually achieve greater recovery speed improvement because recovery process utilizes more disks to rebuild a single failed disk in parallel. For example, Wan et al. [26] 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, S^2-RAID5 system tolerates only one arbitrary disk failure. Muntz et al. [18] 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. [14] 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. [9] 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. [25] proposed a new variant of RAID organization to improve storage efficiency and reduce the performance degradation when disk failure occurs. All these existing methods still speed up the recovery in a limited range or only provide limited reliability. Different from them, our OI-RAID provides a relatively large speed-up ratio, while maintaining high reliability.

3 Background and Motivation

In this section, we introduce erasure codes and the 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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2775231, IEEE Transactions on Parallel and Distributed Systems

3.1 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 $(k \leq l \leq m)$ out of the m encoded blocks. To better serve user requests, most of existing erasure codes are so-called 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 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 $\mathcal{P}$ by 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, the parity blocks are rotationally stored among the disks for load balancing.

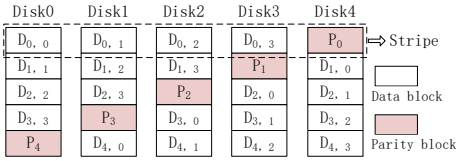


Fig. 1: Layout of the left-symmetric RAID5.

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

3.2 S^2-RAID

S^2-RAID [26] 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 and each subRAID is deployed with RAID5.

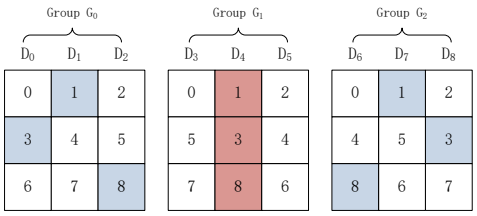


Fig. 2: Layout of S^2-RAID5(3,3).

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 XOR computation, 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 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 layout.

3.3 Parity Declustering

Parity declustering [12, 18] is designed based on balanced incomplete block design (abbr. as BIBD) [11], which is also used in the construction of our OI-RAID, so in the following, we first review the theory of BIBD.

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:

$$\begin{aligned} bk &= vr, \\ \lambda(v-1) &= r(k-1). \end{aligned}$$

Eq. (1) 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 one tuple.

$$\begin{array}{lll} \text{Tuple } \mathcal{T}_0 : 0, 2, 6 & \text{Tuple } \mathcal{T}_1 : 0, 1, 3 & \text{Tuple } \mathcal{T}_2 : 1, 2, 4 \\ \text{Tuple } \mathcal{T}_3 : 2, 3, 5 & \text{Tuple } \mathcal{T}_4 : 3, 4, 6 & \text{Tuple } \mathcal{T}_5 : 0, 4, 5 \\ \text{Tuple } \mathcal{T}_6 : 1, 5, 6 & & \end{array} \quad (1)$$

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 3 is an example of parity

1. The term *tuple* is called *block* in the literatures about block design, but it is easily confused with the commonly used definition of a block as a contiguous chunk of data in storage. So we use tuple by following the work in [12].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2775231, IEEE Transactions on Parallel and Distributed Systems

declustering constructed with the BIBD shown in Eq. (1). 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 Eq. (1) defines the layout of stripe 0 in Figure 3, which consists of the three blocks on $\mathcal{D}_0$, $\mathcal{D}_2$ and $\mathcal{D}_6$, respectively.

D_0	D_1	D_2	D_3	D_4	D_5	D_6
0	1	0	1	2	3	0
1	2	2	3	4	5	4
5	6	3	4	5	6	6

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

Now assume that disk $\mathcal{D}_3$ in Figure 3 is failed. This disk contains three blocks which are parts of stripe 1, 3 and 4, respectively. The other six surviving blocks belonging to these three stripes are stored on the other six surviving disks with one on each. So we can rebuild $\mathcal{D}_3$ in the same way as in S^2-RAID. The shaded areas show that we only need to read exactly one block, which is only one-third of the data compared to conventional RAID, from each of *all* the surviving disks to reconstruct the three failed blocks. Theoretically, we could expect that the recovery speed with parity declustering in Figure 3 is three times as that of conventional RAID.

3.4 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. However, the storage system will have more disks with a larger size of groups, which leads to more frequent disk failures. So S^2-RAID with a 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, it can not fully exploit the parallelism of all disks in the system. For parity declustering, it needs BIBDs of large scale to achieve high speed-up rate for the recovery process. On the one hand, it is difficult to find a BIBD with a large scale. On the other hand, parity declustering with BIBDs of large scale also needs many disks, so it makes a large scale storage system not sufficiently reliable as parity declustering also tolerates only one arbitrary disk failure.

Motivated by the strong demand of high-speed recovery of disks with terabytes of data and the demand of high reliability of large scale storage systems, we aim to develop a new code architecture to achieve fast recovery. Meanwhile, we also aim to tolerate at least three disk failures, which is an industry standard, with small stripe size and low storage overhead. In the following sections, we will present the design of our OI-RAID and show how it achieves these goals.

4 An Example OI-RAID Construction

OI-RAID is designed as a hierarchical architecture of two layers. We divide all disks into groups and divide each group of disks into regions, and finally we group all regions into tuples based on BIBD. The first layer, called outer layer, is a RAID5 based on all regions within a tuple, and the second layer, called inner layer, is a RAID5 within each 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 codes in both layers. In this section, we first use an example as in Figure 4 to explain the main idea of OI-RAID.

Figure 4 is based on the $(7, 7, 3, 3, 1)$-BIBD shown in Eq. (1). In Figure 4, 21 disks are divided into 7 groups $\mathcal{G}_0, \mathcal{G}_1, ..., \mathcal{G}_6$, with 3 disks in each. We further divide each disk into 3 equal-sized parts. 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 which are the first regions from $\mathcal{G}_0, \mathcal{G}_2, \mathcal{G}_6$, respectively, as shown in Figure 4 in the dashed boxes. Similarly, 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, and each group is included in 3 tuples, which correspond to the 3 regions being distributed into 3 different tuples. In Figure 4, each pair of groups are just included in one tuple, which means that there are one region out of each group and the two regions are included in just one tuple.

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

4.1 The Design of Outer Layer Code

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 4, 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, in a row-major order. For the region of tuple $\mathcal{T}_0$ from group $\mathcal{G}_2$, we circularly shift all labels in the second row to 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 one stores the parity which is the XOR-sum of the two data blocks. Take the three units labeled 0 as an example, we

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2775231, IEEE Transactions on Parallel and Distributed Systems

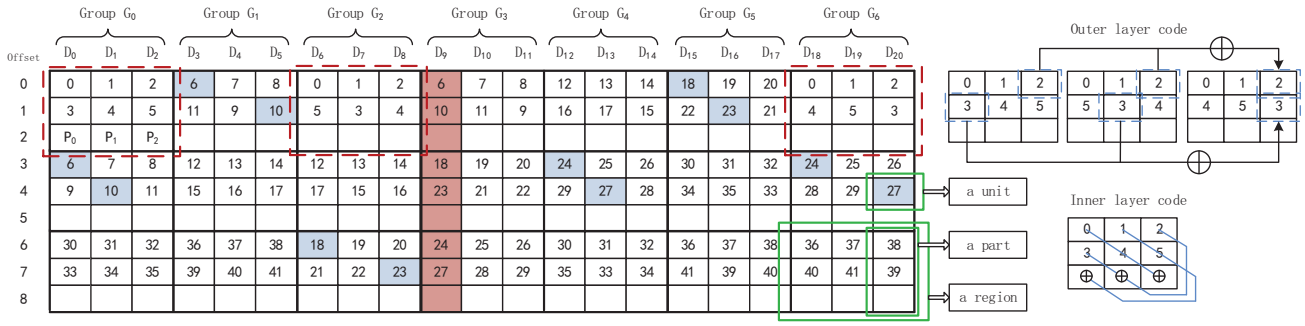


Fig. 4: An example of OI-RAID layout. Inner layer code: $P_0 = 1 \bigoplus 5$, $P_1 = 2 \bigoplus 3$, $P_2 = 0 \bigoplus 4$. Outer layer code: units with the same number form a code vector.

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 constructed similarly, as shown in Figure 4.

4.2 The Design of Inner Layer Code

The code of inner layer is RAID5 and it is deployed within each region. In Figure 4, 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 block is the XOR-sum of the two data blocks along the same diagonal. Take the first region in Figure 4 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 the data blocks in unit 1 and 5.

From Figure 4, 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 parity blocks in the inner layer will be recovered when the system is idle). Thus, the recovery process can be greatly accelerated. Besides, OI-RAID can also tolerate at least three disk failures when both layers are deployed with RAID5.

5 General OI-RAID Design

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

5.1 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 contains 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 equal-sized parts, and divide each part into g storage units. The g parts, one from each disk of a group, form a region. Thus, a region is a $g \times g$ matrix of storage units.

Outer layer code. According to the BIBD, all regions of all disks can be divided into b tuples, each tuple contains k regions. We introduce the design of the outer layer code by focusing on one tuple, and the code design of different tuples are the same. Suppose that we label the $(g-1) \times g$ storage units 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 $\boldsymbol{L}_l = (\mathcal{P}_{i,(j-i\times l) mod g})_{(g-1)\times g}$, where $1 \leq l \leq k-1$. Note that the last g storage units which are in last row will be used to store parity blocks of the inner layer code.

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 $\mathcal{B}_{i,j}^0, \mathcal{B}_{i,j}^1, ..., \mathcal{B}_{i,j}^{k-1}$, respectively, we can just take $\mathcal{B}_{i,j}^{k-1}$ as the parity block and simply deploy RAID5 code as $\mathcal{B}_{i,j}^{k-1} = \mathcal{B}_{i,j}^0 \oplus \mathcal{B}_{i,j}^1 \oplus ... \oplus \mathcal{B}_{i,j}^{k-2}$. Figure 5 shows an example.

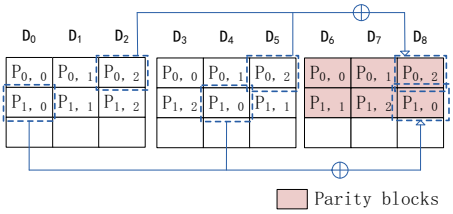


Fig. 5: An example layout of the outer layer code.

Inner layer code. The inner code is deployed within each region with RAID5, while the data blocks are encoded along the same diagonal. 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

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

where $<i–j>_g$=$(i–j) \mod g$. Figure 6 shows an example.

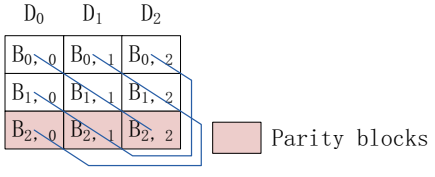


Fig. 6: An example layout of the inner layer code.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2775231, IEEE Transactions on Parallel and Distributed Systems

5.2 Implementation of OI-RAID

To construct BIBDs, we can use perfect difference sets [7], which are already listed out by Singer et al. [22]. In particular, from each of the perfect difference sets, we can construct a Youden square, which corresponds to a BIBD. Table 1 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 1: A partial list of BIBDs.

Benefits of BIBD: By constructing OI-RAID with BIBD, single failure recovery can be greatly accelerated, mainly because BIBD distributes all the code groups across the whole RAID system. Besides, the outer layer of OI-RAID is a RAID5 among all regions in the same tuple, hence the number of regions in each tuple, which is decided by parameter k in BIBD, determines the storage overhead of OI-RAID. Thus, if k is larger, then the storage overhead of OI-RAID should be lower. In other words, with BIBD, OI-RAID only incurs low storage overhead.

Limitations: For BIBD based OI-RAID design, its parameters fully depend on the BIBD used in the outer layer. As there is no general technique to directly construct a BIBD, we can just apply the known BIBDs, e.g., the ones in the partial list mentioned above. Therefore, we can not implement OI-RAID with an arbitrary number of disks. For example, for a (b, v, r, k, λ)-BIBD, OI-RAID assumes that the storage system must have $v \times g$ $(g \geq k)$ disks. Even though we can add some virtual disks which store value of 0, the restriction on the number of disks with BIBD still remains as a serious problem in practical systems. Therefore, we also develop an alternative scheme to construct OI-RAID by using complete graph. We introduce the detailed design in the next section.

6 OI-RAID BASED ON COMPLETE GRAPH

As discussed in Section 5.2, OI-RAID based on BIBD can not construct RAID with an arbitrary number of disks due to the limitation on BIBD design. To relax this restriction so as to provide OI-RAID design with general number of disks, we propose another way to construct OI-RAID by using complete graph, which is a kind of simple undirected graph in which each pair of distinct vertices are connected by a unique edge. The complete graph which contains n vertices is denoted by K_n. Figure 7 shows an example of complete graph K_4, which contains four vertices and six edges.

As discussed in Section 5, to build OI-RAID, we group all the regions into tuples based on BIBD first. Actually, BIBD can be viewed as a specific block design, if we build OI-RAID based on complete graph, we should also construct the corresponding block design first. The construction of block design based on complete graph is a mapping, in which vertices are mapped to the objects and edges are mapped to the tuples. For example, in Figure 7, edge 0 is adjacent to vertex $\mathcal{G}_0$ and $\mathcal{G}_1$, hence tuple $\mathcal{T}_0$ of the block design contains the objects of $\mathcal{G}_0$ and $\mathcal{G}_1$ as shown in Eq. (3). Similarly, edge 4 is adjacent to vertex $\mathcal{G}_1$ and $\mathcal{G}_2$, so tuple $\mathcal{T}_4$ contains $\mathcal{G}_1$ and $\mathcal{G}_2$.

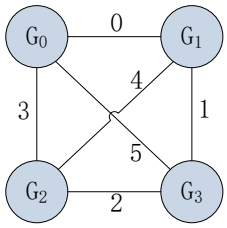


Fig. 7: An example of complete graph K_4.

$$\begin{array}{lll} \text{Tuple } \mathcal{T}_0 : \mathcal{G}_0, \mathcal{G}_1 & \text{Tuple } \mathcal{T}_1 : \mathcal{G}_1, \mathcal{G}_2 & \text{Tuple } \mathcal{T}_2 : \mathcal{G}_2, \mathcal{G}_3 \\ \text{Tuple } \mathcal{T}_3 : \mathcal{G}_0, \mathcal{G}_3 & \text{Tuple } \mathcal{T}_4 : \mathcal{G}_1, \mathcal{G}_3 & \text{Tuple } \mathcal{T}_5 : \mathcal{G}_0, \mathcal{G}_2 \end{array} \quad (3)$$

With the block design, the construction of OI-RAID based on complete graph is similar to that based on BIBD. Suppose that the block design contains v objects, then the OI-RAID based on complete graph needs $v \times g$ disks, where g is also a prime. Figure 8 shows the layout of OI-RAID based on complete graph K_4 with g being 3. There are 12 disks, which are divided into 4 groups, with 3 disks each. With the same method in Section 4, we divide the whole disk array into a region matrix, and each region is a 3×3 matrix of storage units. Then we can divide all the regions into tuples based on the block design shown in Eq. (3). For example, as tuple $\mathcal{T}_0$ = $\{\mathcal{G}_0, \mathcal{G}_1\}$, so the first region in group $\mathcal{G}_0$ and $\mathcal{G}_1$ are allocated into the same tuple. Similarly, as tuple $\mathcal{T}_1$ = $\{\mathcal{G}_1, \mathcal{G}_2\}$, the second region in group $\mathcal{G}_1$ and the first region in group $\mathcal{G}_2$ are allocated into the same tuple. In the same way, we group all the regions into six tuples.

The outer layer code is also deployed among all the regions in the same tuple. However, as each tuple only contains two regions in Figure 7, we can not deploy RAID5 or some other erasure codes. Instead, we use replication here. All the storage units are labeled in the skewed way as mentioned in Section 4.1. Every two storage units with the same label store the same raw data and they are two replicas, hence the outer layer of OI-RAID could tolerate one arbitrary disk failure. The inner layer code is also the same as in Section 4.2, so OI-RAID based on complete graph and BIBD can achieve the same level of fault-tolerant.

When we build OI-RAID based on complete graph, as the outer layer adopts replication instead of erasure codes, so over half of the storage space is stored with redundancy, and thus introduces a high storage overhead. Therefore, it is more cost-efficient to build OI-RAID based on BIBD in terms of storage overhead. However, OI-RAID based on complete graph provides us an another choice, and this scheme can make the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2775231, IEEE Transactions on Parallel and Distributed Systems

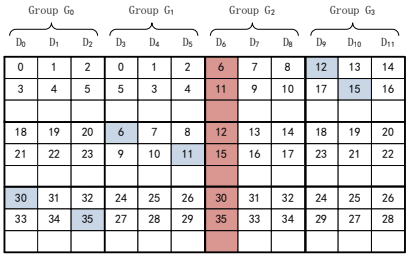


Fig. 8: OI-RAID based on complete graph K_4.

system have a small scale. To avoid confusion, in the following of this paper, we refer OI-RAID to the scheme that is based on BIBD, while for OI-RAID based on complete graph, we will use the full name.

7 Reliability and Failure Recovery

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

7.1 Reliability

Reliability analysis of OI-RAID. The reliability of OI-RAID can be characterized by the following theorem.

Theorem: OI-RAID tolerates 3 concurrent disk failures.

Proof: We first define the concept of parity set. A parity set of a parity block is the set of blocks consisting of the parity block and all data blocks from which the parity block is generated. For example, in Figure 9, 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 $PS(\mathcal{B}_{2,5}) = \{\mathcal{B}_{0,5}, \mathcal{B}_{1,5}, \mathcal{B}_{2,5}\}$.

There are three cases for three concurrent disk failures: (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. 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. Thus, all failed data/parity blocks can be reconstructed with RAID5 code. If OI-RAID is based on complete graph, the every two replicas in the outer layer are stored in different groups, hence all the failed disks in the same group can also be recovered.

Case 2: In the second case, the only one disk which is failed in a group can be recovered with the 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, which is 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 the inner layer code.

By summarizing the above cases, we conclude that, either based on BIBD or complete graph, OI-RAID could recover three arbitrary failed disks. ∎

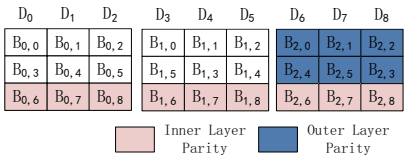


Fig. 9: An example of the layout of a tuple (region set).

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 9 can also tolerate some patterns of four or even more disk failures. So OI-RAID achieves higher reliability than 3-replication which is an accepted industry standard.

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. With another layer of RAID5 in each group, OI-RAID tolerates at least three disk failures. Thus, the system reliability of OI-RAID is much higher than that of S^2-RAID and parity declustering.

OI-RAID design for tolerating more than three disk failures. Note that OI-RAID can only tolerate three arbitrary failures if RAID5 is used in both layers. To support higher reliability, we can use erasure codes with higher liability in both layers. The simplest and also most commonly used erasure code is RS code, so we propose to use RS code in both layers so as to provide higher fault tolerance. For example, in Figure 10, we deploy Reed-Solomon code RS(3,5) in the inner layer and use RAID5 in the outer layer. As the inner layer tolerates two arbitrary disk failures and the outer layer tolerates one disk failure, we can prove that the whole OI-RAID could tolerate all cases of five disk failures.

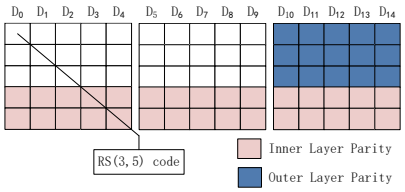


Fig. 10: OI-RAID with RS(3,5) code in the inner layer.

Table 2 shows the fault tolerance level of OI-RAID with different erasure codes, in which all the numbers denote the number of disk failures that could be tolerated. For example, if the inner layer tolerates 2 disk failures and the outer layer tolerates 3 disk failures, then the whole OI-RAID tolerates 11 arbitrary disk failures, which achieves a really high reliability.

7.2 Failure Recovery

In this subsection, we first explain how to speed up the recovery of OI-RAID, and then present the scheme.

If we rebuild a failed disk with outer layer code, we conclude that we only read up to one storage unit from

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2775231, IEEE Transactions on Parallel and Distributed Systems

Outer Layer \ Inner Layer	1	2	3
1	3	5	7
2	5	8	11
3	7	11	15

TABLE 2: Fault tolerance of OI-RAID by deploying codes with different fault tolerance levels in both layers.

each disk in other groups to reconstruct all units in the parity sets in outer layer in this disk. For example, in Figure 9, if disk $\mathcal{D}_4$ fails, we can reconstruct unit $\mathcal{B}_{1,1}$ and $\mathcal{B}_{1,3}$ in $\mathcal{D}_4$ with outer layer code as

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

We see that it reads only one 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.

On the other hand, 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 9, if disk $\mathcal{D}_4$ fails, we can rebuild $\mathcal{D}_4$ with inner layer code as follows, but three storage units are read from each of the disks $\mathcal{D}_3, \mathcal{D}_5$.

$$\begin{aligned} \mathcal{B}_{1,1} &= \mathcal{B}_{1,4} \oplus \mathcal{B}_{1,6}, \\ \mathcal{B}_{1,3} &= \mathcal{B}_{1,0} \oplus \mathcal{B}_{1,8}, \\ \mathcal{B}_{1,7} &= \mathcal{B}_{1,2} \oplus \mathcal{B}_{1,5}. \end{aligned}$$

From the above analysis, we should reconstruct a data/parity block with outer layer code prior to inner layer code so as to leverage device-level parallelism. In particular, if there are less than 4 disk failures, the failed disks can be recovered by using 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 $f == 0$;

8 Numerical Analysis

In this section, we study the performance of OI-RAID via numerical analysis. Specifically, since 99.75% of the system failures are single disk failure [20], we show the efficiency of OI-RAID by comparing the recovery performance of single disk failure of OI-RAID with that of other two typical RAID architectures, S^2-RAID and parity declustering, which are all based on disk grouping. In particular, we compare their recovery performance with two metrics: (1) speed-up ratio, which 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, and (2) read volume ratio, which is defined as the total number of data/parity blocks read from all surviving disks for the recovery of one failed block. The two metrics reflect how much speed improvement the storage system can achieve and the total volume of data that needs to be read during the recovery, respectively.

8.1 Speed-up Ratio

We first derive the speed-up ratio of OI-RAID, parity declustering, and S^2-RAID, then compare their performance via numerical analysis.

S^2-RAID: Suppose that a S^2-RAID layout introduced in Section 3.2 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 Section 3.3, 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 exactly λ 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 1, 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 unit from each disk in other regions to rebuild $g - 1$ units in parallel without conflict. Thus, OI-RAID could achieve a speed-up ratio of $r \times (g - 1)$.

OI-RAID based on complete graph: Suppose we build OI-RAID with complete graph K_n, i.e., each vertex

is connected by $n-1$ vertices and each pair of vertices share one unique edge. Accordingly, in the block design, each object is contained in $n-1$ tuples and each pair of objects are included in exact one tuple. Therefore, based on the mapping scheme, each pair of disk groups participate in the construction of one region set and each disk group contains $n-1$ regions. As a result, the outer layer contributes $n-1$ times of speed-up. For each region set, the skewed data layout and inner layer code are the same with OI-RAID based on BIBD, if each disk group contains g disks, the inner layer contributes $g-1$ times of speed-up. Thus, OI-RAID based on complete graph could achieve a speed-up ratio of $(n-1)\times(g-1)$.

Numerical Results: We first evaluate the speed-up ratio of OI-RAID. Figure 11(a) shows the speed-up ratios of OI-RAID based on the BIBDs in Table 1 and parameter g at different system scales. In particular, each point in the figure shows the speed-up ratio of OI-RAID under a particular setting. Note that we always aim to achieve a high speed-up ratio with as few disks as possible, so in Figure 11(a), the optimal curve shows the best choices of (b,v,r,k,λ) and g, which can achieve high speed-up ratio with the fewest disks. We see that OI-RAID achieves a speed-up ratio of 60 with less than 350 disks.

Figure 11(b) shows the speed-up ratios of OI-RAID based on complete graph. Similarly, each point in this figure denotes an OI-RAID based on complete graph K_n and parameter g. The optimal curve is obtained in the same way as in Figure 11(a). The results show that OI-RAID based on complete graph could achieve a speed-up ratio of 100 with 121 disks. From the angle of speed-up ratio, the OI-RAID based on complete graph may achieve much better recovery performance than that based on BIBD, however, OI-RAID based on complete graph costs much higher storage overhead.

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(c), where the speed-up ratios of OI-RAID are from the optimal curve in Figure 11(a), those of parity deculstering are based on the BIBDs in Table 1, and those of S^2-RAID are derived with group number of 3. 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. The reason is that parity declustering is based on BIBDs, its speed-up ratio equals to $\frac{r}{\lambda}$, which heavily depends on the configuration of the BIBD even for a larger system scale. Besides, OI-RAID achieves similar speed up ratio with S^2-RAID. 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.

8.2 Read Volume Ratio

In this section, we evaluate the read volume ratio of different RAID schemes, including S^2-RAID, parity declustering, MDS codes, and OI-RAID.

S^2-RAID: In a S^2-RAID layout consisting of l groups, each of which contains g disks, there are l storage units in a 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, the read volume ratio 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 $k-1$.

OI-RAID based on complete graph: In an OI-RAID based on complete graph k_n, as the outer layer is deployed with replication strategy, the read volume ratio is 1, which is optimum.

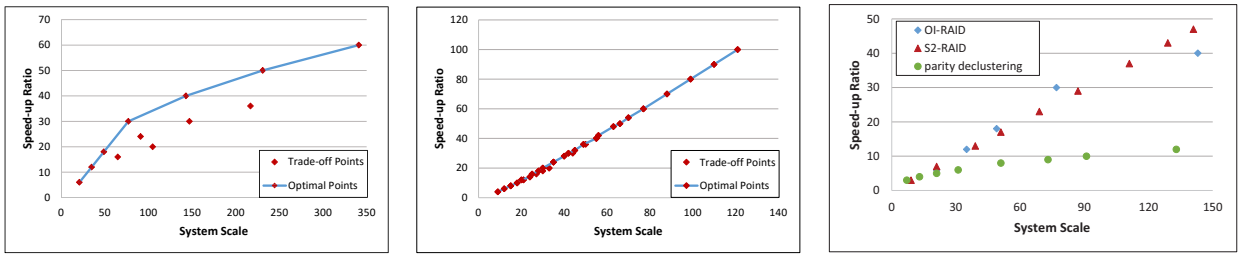
Numerical Results: The comparison of the read volume ratio among OI-RAID ,S^2-RAID, parity declustering and Reed-Solomon code is shown in Figure 12. In this figure, we can see that S^2-RAID and parity declustering achieve the same performance, mainly because the read volume ratio is decided by the deployed 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 [10], the read volume ratio of OI-RAID under the setting at point A is reduced from 6 to 3. That is, OI-RAID under the setting at point A saves 50% of reconstruction cost.

8.3 Lower Bound of 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 scale of a storage system, we define $L_{bound}(\alpha,\beta)$ as follows.

Definition of Lower Bound $L_{bound}(\alpha,\beta)$: It is defined as the minimum number of disks needed to construct

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2775231, IEEE Transactions on Parallel and Distributed Systems



(a) OI-RAID based on BIBD (b) OI-RAID based on complete graph (c) Comparison

Fig. 11: Speed-up ratio under different system scales.

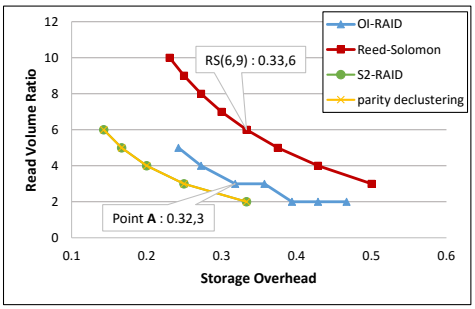


Fig. 12: Comparison of read volume ratio.

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 Ω/α. As a result, we need at least $\lceil(\beta \times \Omega)/(\Omega/\alpha) = \alpha \times \beta\rceil$ disks to read out all the data. Note that α may not be an integer. Containing the failed disk itself, the lower bound $L_{bound}(\alpha, \beta)$ is

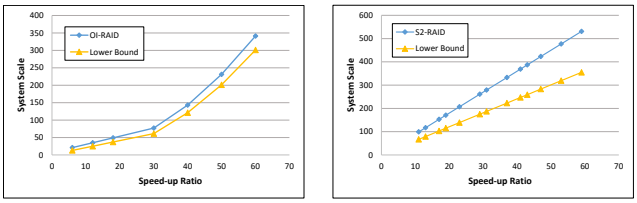
$$L_{bound}(\alpha, \beta) = \lceil \alpha \times \beta \rceil + 1. \quad (4)$$

Recall that there are $v \times g$ disks in an OI-RAID mentioned above. According to Equation (1), 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 are relatively small, e.g., $r = k \leq 6$ and $g \leq 11$. So OI-RAID needs only a few more disks than the theoretical minimum, so as to achieve the highest speed up of single disk failure recovery.

OI-RAID: In Figure 13(a), 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., $60\times$ of speed-up compared to conventional RAID.

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 deploy Reed-Solomon code in S^2-RAID. Figure 13(b) 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.

Parity Declustering: As stated in Section 8.1 and Section 8.2, the speed-up ratio of parity declustering is $\frac{r}{\lambda}$ and its read volume ratio is exactly $k-1$. According to Equation (1), we have $v = r(k-1)/\lambda + 1$. According to Equation (4), we can calculate the lower bound of number of disks required by parity declustering as $L_{bound}(r/\lambda, k-1) = r(k-1)/\lambda + 1$. 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.



(a) OI-RAID (b) S^2-RAID with RS(6,9)

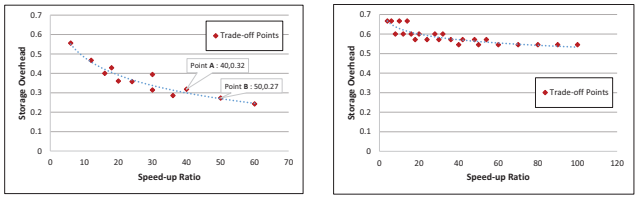
Fig. 13: The lower bound of system scale.

8.4 Storage Overhead of OI-RAID

We point out that some trade-off points on the optimal curve in Figure 11(a) may not meet the needs due to its high storage overhead. Therefore, we further show the relationship between the speed-up ratio and the storage overhead for OI-RAID with BIBD in Figure 14(a). We define the storage overhead as the ratio of redundancy here. For example, if the system contains n storage units and m of them are redundancy, then the storage

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2775231, IEEE Transactions on Parallel and Distributed Systems

overhead is m/n. 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. Figure 14(b) shows the storage overhead of OI-RAID based on complete graph. We see that the lowest storage overhead is 0.54, meaning that over half of storage units are redundancy, which is much higher than that of OI-RAID based on BIBD. Therefore, OI-RAID based on complete graph trades storage for recovery performance.



(a) OI-RAID via BIBD (b) OI-RAID via complete graph

Fig. 14: Storage overhead.

For comparison, we further consider two typical codes in Table 3. 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 [4] and it can tolerate four arbitrary disk failures. In particular, the trade-off point **A** in Figure 14(a) 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.

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

TABLE 3: Storage overhead of two Reed-Solomon codes.

8.5 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 4, 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. OICG(α,g) represents an OI-RAID based on complete graph K_α and group size g. Finally, system scale means the number of disks in the system. From Table 4, we can have the following conclusions.

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
OICG(11,11)	100	1	101	121	0.54

TABLE 4: Performance comparison of various codes under various typical settings.

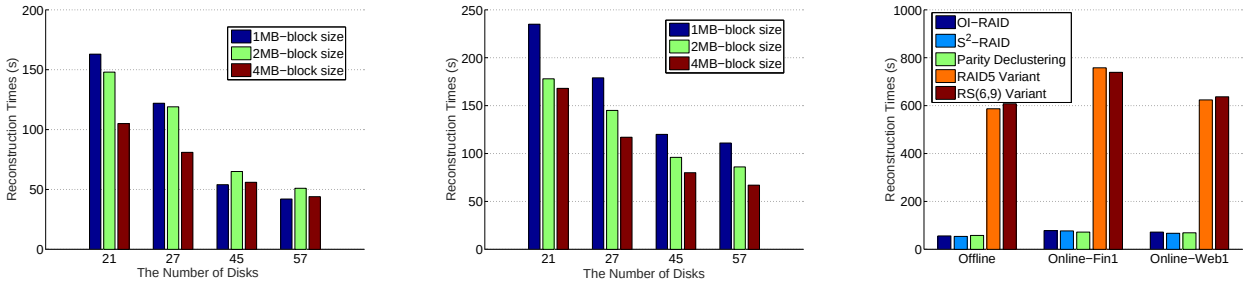
- OI-RAID could achieve a high speed-up ratio while keeping a low read volume ratio. For example, the read volume ratios of OI(21,5,1,11) and PD(21,5,1) are both 4, but OI(21,5,1,11) could achieve a speed-up ratio of 50, while PD(21,5,1) only achieves a speed-up ratio of 5.
- OI-RAID achieves much higher 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.
- OI-RAID based on complete graph achieves much better recovery performance than that based on BIBD while costing much higher storage overhead. For example, OICG(11,11) utilizes 110 fewer disks than OI(21,5,1,11) to achieve more than 50$\times$ of speed-up ratio, but the storage overhead of OICG(11,11) is also doubled.
- 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 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.

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

9 Performance in Real Systems

In this section, we evaluate the performance of OI-RAID in real systems. In the following, we first describe the experiment setting, and then show both the recovery performance and user performance.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2775231, IEEE Transactions on Parallel and Distributed Systems



(a) OI-RAID: offline recovery (b) OI-RAID: online recovery (c) Comparison

Fig. 15: Recovery performance.

9.1 Experiment Setting

Our experimental testbed consists of a cluster system with a number of Sugon I620-G20 nodes, the hardware details of each node are listed in Table 5. The nodes are interconnected through an Mellanox FDR MSX6025F InfiniBand Switch, we use iSER (iSCSI Extensions for RDMA) over the network to eliminate TCP/IP processing overhead. We use the libaio library to asynchronously access each storage node and implement a new array controller to handle address mapping and data recovery. User requests also go through the array controller to access the underlying storage servers. We implement parallel recovery at device level, precisely, we use a queue in the controller to cache some requests, then send them in batches to disks and issue them in parallel.

OS	CentOS 7.3
CPU	2x Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
Memory	64 GB
Network card	Mellanox MT27500 ConnectX-3
Disks	12x Seagate ST1000NM0023, 1TB, 7200RPM

TABLE 5: Hardware details of each node.

To evaluate online recovery performance, we consider two real-world I/O traces, Financial1 and WebSearch1. Financial1 trace was collected from OLTP applications running at a large financial institution, and this workload is write dominated (76.84% writes). WebSearch1 trace was collected from a web search engine, and it is read dominated (99.98% reads). We replay the traces by using the btreplay tool in Linux. We implement online recovery with two threads, which are responsible for recovery and user requests respectively, and let them compete for the same resources of CPU and disk bandwidth.

9.2 Recovery Performance

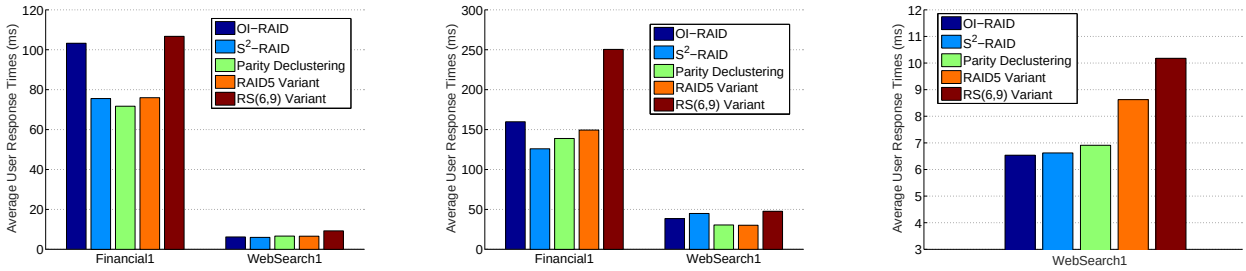
We show the recovery time of OI-RAID in both offline and online scenarios, and also consider different system scales and block sizes. Besides, we also compare OI-RAID with other schemes including parity declustering, S^2-RAID, RAID5 and RS code.

Figure 15(a) and Figure 15(b) show the reconstruction time of OI-RAID in both offline and online scenarios. For offline recovery, all resources of CPU and disk bandwidth are used for recovery, while for online recover, we also run Financial1 during recovery and use two threads to compete for the resources, one for recovery and the other for user requests. In this experiment, we consider different system scales by varying the number of disks from 21 to 57, and also vary the block size from 1MB to 4MB. To save experiment time, we limit the amount of reconstruction data as 100GB. From both figures, we see that the reconstruction time decreases as system scales increases. That is, we can achieve a higher speed-up ratio when deploying OI-RAID with more disks. This result is consistent with the numerical analysis in Figure 11. Besides, we see that the reconstruction time can get reduced with a larger block size in general cases. This is mainly because with OI-RAID, only a part of data on each surviving disk is required for recovery, so surviving disks are not accessed continuously and larger block size can amortize the disk seek time. However, when system scale becomes large, large block size may not lead to the reduction of reconstruction time, e.g., as shown in Figure 15(a), when the number of disks increases to 45, the reconstruction time is not reduced as the block size increases. This is mainly because when the number of disks become large, the aggregated disk bandwidth is even larger than the network bandwidth, so the system bottleneck is not the disk seek time caused by random I/O any more.

We further compare OI-RAID with other four RAID schemes, and we also show the recovery performance of these schemes in both offline and online scenarios. In this experiment, we fix the number of disks as 45, and the block size as 4MB. So OI-RAID is designed based on the (35, 15, 7, 3, 1)-BIBD with $g = 3$. For fair comparison, we configure the RAID schemes to make them have the same read volume ratio. In particular, For S^2-RAID, we make it consist of 3 groups, each of which contains 15 disks. We deploy the parity declustering scheme by using a (33, 45, 22, 3, 1)-BIBD. For RAID5, we deploy a variant which consists of 15 groups of conventional 2+1 RAID5 arrays. For RS code, we also deploy a variant which consists of 5 groups of RS(6,9) arrays.

Figure 15(c) shows the comparison results. We see that comparing to conventional RAID schemes like RAID5 and RS code, OI-RAID can significantly reduce the reconstruction time, e.g., by up to 10×. For S^2-RAID and parity declustering, both of which are already optimized for speeding up recovery, OI-RAID achieves very similar

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2775231, IEEE Transactions on Parallel and Distributed Systems



(a) Response time in normal mode (b) Response time during recovery (c) Degraded read performance

Fig. 16: Average user response time in normal mode and failure mode.

recovery performance. However, OI-RAID provides at least three arbitrary disk failures, while both S^2-RAID and parity declustering considered in this experiment only tolerate single disk failure. Therefore, OI-RAID simultaneously achieves both high reliability and high recovery performance.

9.3 User Performance

Now we show the user performance of OI-RAID and compare it with other schemes studied above. In this experiment, we still use the same configuration as before.

We first focus on the average user response time, and the results are shown in Figure 16. We see that in normal or failure-free mode (Figure 16(a)), OI-RAID has similar performance with RS code under Financial1 trace, but both OI-RAID and RS perform worse than the other three schemes. This is mainly because OI-RAID and RS tolerate three disk failures, so they have more parities and introduce more writes due to parity update as the Financial1 trace is write dominated. However, for the read-dominated workload WebSearch1, all the schemes provide similar user performance as parities have no impact on read performance in failure-free mode.

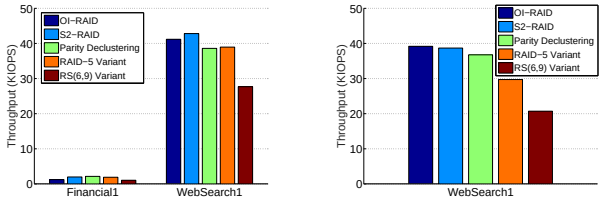
When disk failure happens and the recovery process runs simultaneously with user requests, different results can be observed, see Figure 16(b). In particular, since OI-RAID speeds up the recovery process comparing to RS code, which provides the same fault tolerance with OI-RAID, the average response time of OI-RAID gets significantly reduced by comparing to that of RS code.

Finally, to show the degraded read performance, we focus on the read-dominant workload only. As shown in Figure 16(c), we see that OI-RAID outperforms RAID5 and RS, this is due to the benefit of accessing only part of data from each surviving disk to reconstruct failed data.

Now we focus on the user throughout. Figure 17(a) shows the throughput results in normal mode, and Figure 17(b) shows the results of degraded read throughout, in which case we run only the read-dominated workload WebSearch1. We find that the results are consistent with those of average user response time shown in Figure 16.

10 Conclusion

In this paper, we propose a new RAID architecture, OI-RAID, which achieves fast recovery and high reliability.



(a) Throughout in normal mode (b) Degraded read throughput

Fig. 17: User throughput in normal mode and degraded read throughput.

OI-RAID consists of two layers of codes. The outer layer is organized by using BIBDs or complete graph, and the inner layer is encoded with RAID5 along diagonal lines. OI-RAID can tolerate at least three arbitrary disk failures if RAID5 are deployed in both layers. In summary, 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.

References

[1] 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*, 1998.

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

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

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

[5] B. Calder, J. Wang, A. Ogus, et al. Windows azure storage: a highly available cloud storage service with strong consistency. In *ACM SOSP*, pages 143–157, 2011.

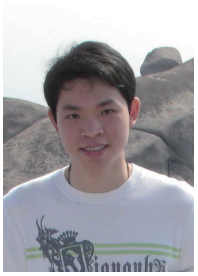
[6] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row-diagonal parity for double disk failure correction. In *USENIX FAST*, 2004.

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

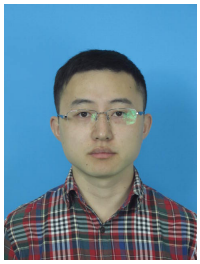
[8] G. DeCandia, D. Hastorun, M. Jampani, et al. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220, 2007.

[9] S. El Rouayheb and K. Ramchandran. Fractional repetition

codes for repair in distributed storage systems. In *Allerton*, 2010.
[10] A. Fikes. Storage architecture and challenges. *Talk at the Google Faculty Summit*, 2010.
[11] M. Hall. *Combinatorial theory*, volume 71. John Wiley & Sons, 1998.
[12] M. Holland and G. A. Gibson. *Parity declustering for continuous operation in redundant disk arrays*. ACM ASPLOS, 1992.
[13] M. Holland, G. A. Gibson, and D. P. Siewiorek. Fast, online failure recovery in redundant disk arrays. In *The 23rd International Symposium on Fault-Tolerant Computing*, 1993.
[14] C. Huang, H. Simitci, Y. Xu, A. Ogus, et al. Erasure coding in windows azure storage. In *USENIX ATC*, 2012.
[15] C. Huang and L. Xu. Star: An efficient coding scheme for correcting triple storage node failures. *IEEE ToC*, 57:889–901, 2008.
[16] R. J. McEliece and D. V. Sarwate. On sharing secrets and reed-solomon codes. *Communications of the ACM*, 24:583–584, 1981.
[17] J. Menon and D. Mattson. Distributed sparing in disk arrays. In *Compcon Spring'92. Thirty-Seventh IEEE Computer Society International Conference, Digest of Papers*, pages 410–421, 1992.
[18] R. R. Muntz and J. C. S. Lui. *Performance analysis of disk arrays under failure*. VLDB, 1990.
[19] D. A. Patterson, G. Gibson, and R. H. Katz. *A case for redundant arrays of inexpensive disks (RAID)*. ACM SIGMOD, 1988.
[20] 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 *USENIX FAST*, 2007.
[21] 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.
[22] J. Singer. A theorem in finite projective geometry and some applications to number theory. *Transactions of the American Mathematical Society*, 43:377–385, 1938.
[23] I. Tamo, Z. Wang, and J. Bruck. Zigzag codes: Mds array codes with optimal rebuilding. *IEEE TIT*, 59:1597–1616, 2013.
[24] 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 *USENIX FAST*, 2007.
[25] 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.
[26] J. Wan, J. Wang, Q. Yang, and C. Xie. S2-raid: A new raid architecture for fast data recovery. In *IEEE MSST*, 2010.
[27] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao. Workout: I/o workload outsourcing for boosting raid reconstruction performance. In *USENIX FAST*, 2009.
[28] L. Xiang, Y. Xu, J. Lui, and Q. Chang. Optimal recovery of single disk failure in rdp code storage systems. In *ACM SIGMETRICS*, 2010.
[29] 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 ToS*, 7:11, 2011.
[30] L. Xu and J. Bruck. X-code: Mds array codes with optimal encoding. *IEEE TIT*, 45:272–276, 1999.
[31] 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 ToC*, 63:995–1007, 2014.
[32] 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 TPDS*, 25:1830–1840, 2014.

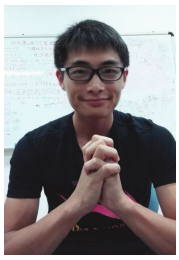


Yongkun Li is currently an associate professor in School of Computer Science and Technology, University of Science and Technology of China. He received the B.Eng. degree in Computer Science from USTC in 2008, and the Ph.D. degree in Computer Science and Engineering from The Chinese University of Hong Kong in 2012. After that, he worked as a postdoctoral fellow in Institute of Network Coding at The Chinese University of Hong Kong. His research mainly focuses on file and storage systems.

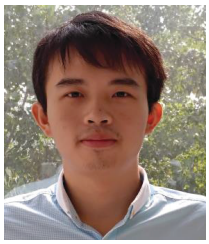


Neng Wang received the Master degree from the School of Computer Science and Technology at University of Science and Technology of China in 2016, and received his Bachelor's degree in Computer Science from Sichuan University in 2013. His research mainly focuses on erasure codes and distributed storage systems.

Chengjin Tian is now a master student in the school of Computer Science and Technology, University of Science and Technology of China. He received his Bachelor's degree in computer science from University of Science and Technology of China in 2016. His research mainly focuses on various aspects of storage systems, including the scaling and reliability issues.



Si Wu received his Ph.D. degree from University of Science and Technology of China in 2016, and received his Bachelor's degree in computer science from University of Science and Technology of China in 2011. His research mainly focuses on erasure codes and distributed storage systems.



Yueming Zhang is currently working toward the master degree with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China. He received his bachelor's degree in computer science from Anhui University of Technology in 2015. His research interests mainly focus on various aspects of distributed storage systems.

Yinlong Xu received the B.S. degree in mathematics from Peking University in 1983, and the M.S. and Ph.D. degrees in computer science from University of Science and Technology of China (USTC) in 1989 and 2004, respectively. He is currently a professor with the School of Computer Science and Technology at USTC, and is leading a research group in doing some networking and high performance computing research. His research interests include network coding, storage systems, combinatorial optimization, design and analysis of parallel algorithms, parallel programming tools, etc. He received the Excellent PhD Advisor Award of Chinese Academy of Sciences in 2006.