Elastic-RAID: A New Architecture for Improved Availability of Parity-Based RAIDs by Elastic Mirroring

Jie Yao, *Member, IEEE*, Hong Jiang, *Fellow, IEEE*, Qiang Cao, *Member, IEEE*, Lei Tian, and Changsheng Xie, *Member, IEEE*

Abstract—In this paper, we propose Elastic-RAID, a new RAID architecture to achieve high performance and high reliability for large-scale distributed and parallel storage systems. The key idea behind Elastic-RAID is to smartly utilize the free space existing in parity-based disk arrays to store additional mirroring data. This additional mirroring data redundancy, when strategically and judiciously activated and exploited in a RAID system, enables improved system I/O performance, fault tolerance and recovery. Depending on the amount of free space available and whether the emphasis is on performance or reliability, the elasticity in Elastic-RAID is manifested in how each design objective is achieved. For the performance objective, Elastic-RAID improves small-write performance by writing original and mirroring data synchronously and leaving the costly parity update in the background at a later idle/lightly-loaded time. For the reliability objective, at least two concurrent disk failures can be tolerated when Elastic-RAID is employed in a RAID5 system that has 50 percent or more free space. Higher reliability is provided for important data when free space is less than 50 percent. To achieve the design goal of elasticity, we introduce a novel data layout and addressing scheme. Our extensive trace-driven evaluations on an Elastic-RAID prototype in the typical configurations of RAID5 show that Elastic-RAID boosts the small-write performance in the normal operational state by at least 40 percent, improves the user I/O performance in the reconstruction state by at least 30 percent and shortens the recovery time by at least 40 percent.

Index Terms—RAID, performance and reliability, parallel I/O

1 Introduction

As a fundamental component and essential building block of large-scale parallel and distributed storage systems, the RAID architecture [23] has been widely deployed to meet both performance and reliability requirements. For performance, RAID exploits the parallelism of data striping among multiple disks to boost I/O throughput. For reliability, RAID employs data redundancy to ensure high reliability by means of mirroring or parity encoding. The choice between mirroring and parity encoding is often a tradeoff among reliability, space-efficiency and performance requirements.

Parity-based RAIDs (e.g., RAID5 or RAID6) suffer from the small-write problem [11], a well-known challenge due to the unique parity-update process that greatly amplifies the internal I/Os owing to the necessary read-modify-write operation sequence. This problem significantly degrades the user I/O performance not only in the operational state, but also in the on-line reconstruction state. Unfortunately, small I/Os are found to be relatively frequent in many applications and datasets. Xie et al. [41] have found that in HPC datasets small file I/Os are common. Another study [10] reveals that parallel storage systems store many small files in addition to the large ones in practice and small file I/Os tend to become a performance challenge for parallel file systems. Mirroring-based RAIDs (e.g., RAID1 or RAID10) are free of the small-write problem and superior to parity-based RAIDs in failure tolerance and reconstruction performance, but their space efficiency is usually much lower than parity-based RAIDs.

AutoRAID [39] proposes a solution to integrate these two types of RAIDs by selectively storing data chunks in either the RAID1 or RAID5 set. In AutoRAID, the newly written data chunks are directed to the RAID1 set for high write performance and the "cold" data chunks will be evicted from the RAID1 set to the RAID5 set periodically. In essence, AutoRAID adopts a strategy, similar to those of disk-cached HDD (DCD) [17] and SSD-cached HDD [42], that uses the RAID1 set to log/cache all the small writes addressed to the underlying RAID5 set. While AutoRAID improves small-write performance with RAID1 set logging/caching, it fails to improve the reliability since each data chunk can only be located in either the RAID1 set or the RAID5 set, but not both. Furthermore, AutoRAID's usage of non-uniform length stripe, fully associative logical-to-physical address mapping and write buffer (NVM) increases the implementation complexity, severely limiting

- *J. Yao is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, P. R. China 430074. E-mail: jackyao@hust.edu.cn.*
- *H. Jiang and L. Tian are with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588-0150. E-mail: jiang@cse.unl.edu, leitian.hust@gmail.com.*
- *Q. Cao and C. Xie are with the Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, P. R. China 430074. E-mail: {caoqiang, cs_xie}@hust.edu.cn.*

Manuscript received 18 May 2014; revised 29 Apr. 2015; accepted 29 Apr. 2015. Date of publication 12 May 2015; date of current version 16 Mar. 2016. Recommended for acceptance by A. R. Butt.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2432808

AutoRAID's deployment during the last two decades since its introduction.

Extra storage space is the necessary cost of introducing data redundancy in RAID systems. Existing RAID systems use dedicated spare disks or fixed reserved space to accommodate data redundancy. We believe that such extra storage space can also be obtained by increasing the space utilization of the existing disks and exploiting unused space. Free space in storage systems has been shown to be ubiquitous because of overprovisioning either purposely (e.g., anticipating peak usages) or unintentionally (e.g., low capacity utilization) [14], [35], [20]. Nevertheless, the main difficulty in identifying the free space in block-level storage devices lies in the narrow block interface [33] between the storage devices and the upper levels that hides much of the semantic information from the upper levels and makes the identification of newly freed space by upper levels very hard, if not impossible.

Fortunately, several techniques have been proposed recently to overcome this difficulty by conveying relevant sematic information from the upper levels to the block-level storage devices, notably, SDS [34] and TRIM [31], with successful outcomes. For example, TRIM has already been widely adopted by SSD manufactures [4] and incorporated into a number of popular operating systems (e.g., Windows7, Linux, Mac OS X) [5], [32]. With the help of these techniques, the block-level storage devices can be made aware of the availability and locations of the free space released by the upper levels.

To this end, we propose *Elastic-RAID*, a new RAID architecture, which combines the advantages of both mirroring-based RAID systems and parity-based RAID systems by utilizing the free space existing in a parity-based RAID system to elastically incorporate the mirroring redundancy. The key enabling technique for Elastic-RAID to achieve its elasticity is a novel data layout and addressing scheme (detailed in Section 4), which is simple to implement without any additional hardware. Elastic-RAID is fundamentally different from AutoRAID in that the former is a new way of augmenting mirroring (or replication) for an entire stripe by utilizing the existing free space for both performance and reliability improvements. Unlike AutoRAID, there is no dedicated RAID-1 set in Elastic-RAID.

By employing both the parity and mirroring data redundancy judiciously, a RAID5-based Elastic-RAID is able to tolerate a minimum of two disk failures and a maximum of $n/2$ disk failures, where n is the number of component disks in the RAID5, when the available free space is equal to or more than half of the total capacity. When the free space is less than half of the total capacity, Elastic-RAID can still use the available free capacity to provide a higher level of reliability for a proportional amount of the data that is considered more important than the rest. When all the free capacity is run out due to possible peak usage, Elastic-RAID reduces to the standard parity-based RAID it is based on (e.g., RAID5 or RAID6).

Choosing to decouple the parity updates from the critical I/O path and ensuring data reliability through mirroring redundancy until parities are updated asynchronously in the background during idle or lightly loaded periods, Elastic-RAID can significantly improve the small-write performance without sacrificing reliability. Elastic-RAID also utilizes the added mirroring redundancy to rebuild the lost data and serve user requests during the reconstruction period. It is able to substantially reduce the internal I/Os induced by parity-based data reconstruction by directly reading the mirroring data. This in turn significantly shortens the on-line recovery time and alleviates the performance degradation of user I/O requests during the reconstruction.

Thus, Elastic-RAID provides higher reliability in the operational state, shorter on-line recovery time, and better user I/O performance in the reconstruction state. In summary, the elasticity in our Elastic-RAID permeates its design space in how the amount of free space, data reliability and system performance in both the operational state and the reconstruction state are balanced and traded off dynamically (see Section 3 for more details), which improves the overall system availability of parity-based RAIDs.

The main contributions of this paper include:

- We propose a novel RAID architecture, Elastic-RAID, which introduces a novel data layout and addressing scheme to efficiently combine the advantages of the mirroring-based RAID and the parity-based RAID by exploiting and utilizing the available spare/free space in a parity-based RAID system itself without any additional hardware capacity.
- Two Elastic-RAID design strategies are presented to elastically trade off between the reliability and the performance according to the amount of free space.
- We prototype and evaluate an Elastic-RAID system based on the Linux software RAID5 implementation to demonstrate its effectiveness and efficacy. Based on the evaluation results, we further analyze and compare the reliability of Elastic-RAID and different RAID levels.

The rest of the paper is organized as follows. In Section 2, we provide the necessary background information and motivate the Elastic-RAID research. In Section 3, we discuss the elasticity of the Elastic-RAID design that trades off among the three dimensions of the amount of free space, reliability and performance. The Elastic-RAID architecture and its design are detailed in Section 4. We evaluate a prototype of Elastic-RAID based on RAID5 in Section 5. Section 6 discusses related works and Section 7 concludes the paper.

2 Background and Motivations

RAID [23] systems, with the exception of RAID0, can generally tolerate at least one disk failure. A RAID system operates in one of the three states, namely, the *operational state* when there is no disk failure, the *degraded state* when one or more disks failed without unrecoverable data loss while the system works at a degraded level of performance, and the *reconstruction state* when the lost data on the failed disk(s) are being reconstructed on-line from the surviving disks via data redundancy to spare disk(s).

Many studies based on quantitative analysis of massive amounts of error log data indicate that hard disk drives (HDDs) fail at a significantly higher rate than previously assumed [7], [15], [18], [24], [25]. In addition to the number of simultaneous disk failures a RAID system can tolerate, its

reliability is also significantly impacted by the on-line recovery time, particularly in the modern large-scale data centers [24], [29]. VDF [38] proposes an asymmetric cache scheme for parity-based RAID systems to improve the reconstruction performance. The basic idea is to give higher priority to caching the data associated with the faulty disks. WorkOut [40] outsources all write requests and popular read requests to a surrogate RAID set during reconstruction to speedup the user response time and shorten the recovery time. SPA [36] is designed to flexibly store and update supplementary parity units on the newly augmented spare disk(s) for online RAID systems to improve the reconstruction performance and reliability. However, none of these schemes can improve the overall reliability in both the operational and reconstruction states. IPC [13] exploits the intra-disk redundancy to deal with latent sector errors and thus improves the reliability. However, it needs fixed reserved space for the increased redundancy. Parity-Declustering RAID [16] proposes a new layout of parities to alleviate I/O load during recovery and shorten reconstruction time. It does not improve the reliability and performance in normal operational state.

Of the two most widely used types of RAID systems, the parity-based systems (e.g., RAID5 and RAID6) and the mirroring-based systems (e.g., RAID1 and RAID10), the former is known to offer higher space-efficiency while the latter is known to provide better user I/O performance in both the operational state and the reconstruction state, as well as shorter recovery time. This is because the number of additional internal I/Os induced by a user I/O request is much larger in the former than in the latter due to the *I/O amplification* problem [40].

AutoRAID [39] consolidates RAID1 and RAID5 in a hierarchical way by selectively storing data chunks in Physical Extend Groups (PEGs) organized as either of these two types of storage classes and uses non-uniform length stripes for these two PEGs. In AutoRAID, the hot data chunks are stored in the RAID1 set for high access performance and the cold data chunks will be evicted from the RAID1 set to the RAID5 set periodically. As the active set of data changes, newly active data are promoted to mirrored storage, and data that have become less active are demoted to RAID5 in order to keep the constant amount of mirrored data. For write requests, AutoRAID adopts the RAID1 set to log/cache all the small writes addressed to the underlying RAID5 set. While AutoRAID improves active data performance with RAID1 set logging/caching, it fails to improve the reliability since each data chunk can only be located in either the RAID1 set or the RAID5 set, but not both. When the free space is low, AutoRAID has to migrate data from the RAID1 set to the RAID5 set frequently to make room for new user I/Os, resulting in a storage space competition between the RAID1 and RAID5 sets. A fully associative logical-to-physical address mapping scheme and NVM buffer are needed for its implementation.

On the other hand, free space is widely available in storage systems. Studies reveal that enterprises report capacity utilization rates averaging only 19 percent when calculated using written data as a proportion of purchased capacity [14]. It was found that the storage capacity utilization in data centers is typically 50 percent in Symantec's 2008 State of the Data Center Survey [35]. Mark Levin [20] also pointed out that the average utilization is 56.6 and 46.4 percent for local storage in the UNIX and Windows environments respectively. For SAN deployment, this proportion increases to 75.5 and 55.8 percent respectively.

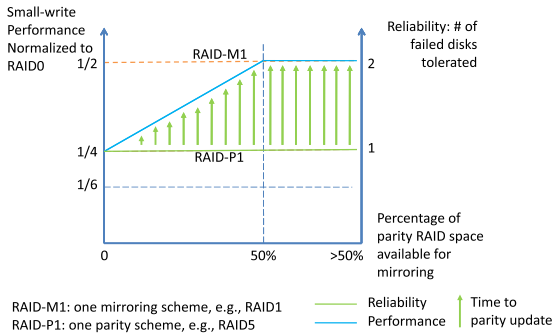


Fig. 1. Performance oriented design strategy.

The dynamically changing free space in a RAID system dictates that any scheme exploiting such space to incorporate mirroring redundancy into a parity-based system should be able to detect/identify and flexibly manage this space in the runtime. This runtime ability to identify the "liveness" of blocks at the block level had until very recently been difficult, if not impossible, to realize. This is because the narrow block interface [33] between file systems and disks prevents the file-system semantics about the block "liveness" from being conveyed to the disks. Fortunately, there has been some new and significant progress [31], [34] made in this area that makes it possible to identify free space and differentiate the importance of data.

The above observations motivate us to propose Elastic-RAID, which utilizes the free space available in parity-based RAID systems to incorporate mirroring data redundancy. It provides higher reliability than parity-based RAID by introducing a new data layout scheme, shortens the recovery time, and improves the user I/O performance in the operational state and the reconstruction state by reducing internal I/Os through a joint exploitation of parity and mirroring redundancy. By elastically trading off among the amount of free space, reliability and performance, Elastic-RAID offers a rich design space that will be discussed and explored next.

3 Elastic Design Methodology

Because of the changing nature of the free space, the amount of mirroring data changes dynamically. It is this dynamism, combined with the interplay among the often-conflicting goals of the amount of free space, reliability and performance, that offers a rich and elastic design space for Elastic-RAID. In what follows, we explore this elastic design space in terms of different design strategies, goals and constraints.

3.1 Performance Oriented Design Strategy

In a performance oriented Elastic-RAID, the write operation will be optimized if the data to be updated has mirroring, that is, write to the original data and the corresponding mirroring data synchronously and leave the parity update process to a later but lightly loaded period and in the background.

Fig. 1 shows the relationship among free space, small-write performance and reliability. For small writes, the

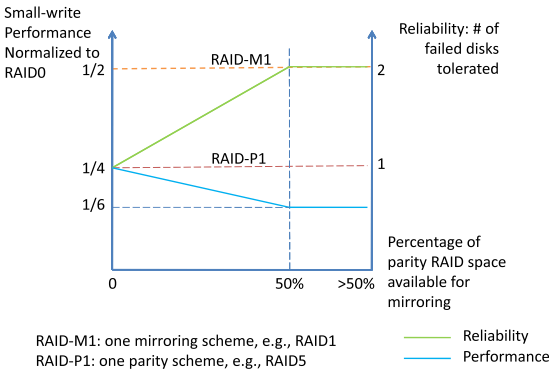


Fig. 2. Reliability oriented design strategy.

performance is estimated in terms of the number of I/Os and the performance upper bound is $1/2$ that of RAID0, since in this case one small write request will generate two internal write I/Os: one for the original data itself, the other for the mirroring data. The performance lower bound is $1/4$ that of RAID0, since in this case one small write request will generate four internal I/Os: two reads for original data and original parity, two writes for the new data and new parity. For reliability, the upper bound is two-disk-failure tolerance, since the one parity scheme provides a failure tolerance of one disk and the one mirroring scheme provides a failure tolerance of at least one disk.

3.2 Reliability Oriented Design Strategy

In a reliability oriented Elastic-RAID, the corresponding parities and the mirroring data will be updated simultaneously when a write request is issued.

Similar to the case of the performance oriented Elastic-RAID, Fig. 2 shows for a reliability oriented Elastic-RAID the relationship among free space, small-write performance and reliability. For a small write, as in the case of performance oriented design strategy, the performance is estimated in terms of the number of I/Os and the performance upper bound is $1/4$ that of RAID0, since in this case one small write request will generate four internal I/Os, two reads for the original data and original parity and two writes for the new data and new parity. The performance lower bound is $1/6$ that of RAID0, since in this case one small write request will generate six internal I/Os, two reads for the original data and original parity, four writes for the new data, new data mirroring, new parity, and new parity mirroring. The reliability upper bound is still a tolerance of at least two disk failures, one provided by the one parity scheme and at least another one provided by the one mirroring scheme.

If 50 percent or more of the RAID capacity is available for mirroring, every data chunk can have a corresponding mirrored duplicate, which results in the highest reliability and the possibility for every small write to be optimized. Because of the delayed parity update, the reliability of a performance oriented Elastic-RAID will be degraded to that provided by the mirroring redundancy (i.e., tolerating one disk failure at a minimum) until the parity is updated. After the parity update is completed at a later but idle or lightly loaded period, the reliability returns to the highest level of tolerating at least two disk failures. In contrast, the reliability oriented Elastic-RAID always achieves its highest reliability but at the cost of its small-write performance that is decreased to the lower bound. If the percentage of free space is lower than 50 percent, only a proportional fraction of the data chunks can be protected by mirroring and parity redundancy simultaneously and benefit from the corresponding small-write optimization.

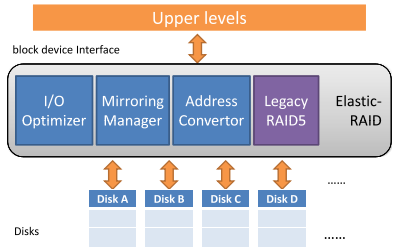


Fig. 3. Architecture of elastic-RAID.

4 Design

Elastic-RAID is designed to improve the reliability and performance of parity-based RAID systems such as RAID5 and RAID6. In this section, for the convenience of discussion and presentation, we assume that Elastic-RAID is employed in a generic parity-based RAID system where RAID5 is used to illustrate the design concepts in concrete examples.

4.1 Architecture

The design of Elastic-RAID entails adding several functional modules to the standard parity-based RAID, including *address convertor*, *mirroring manager* and *I/O optimizer*, as indicated in the Elastic-RAID architecture shown in Fig. 3. To effectively and efficiently exploit free space for improved reliability, Elastic-RAID introduces a novel data layout scheme that is different from the standard parity-based RAID, for which the *address convertor* module and the *mirroring manager* module are designed to address data, manage allocation, and release the space occupied by mirrored data copies on demand. An *I/O optimizer* module is designed to exploit the mirroring redundancy to optimize the small-write performance in the operational state and improve both the user I/O performance and the reconstruction performance in the reconstruction state.

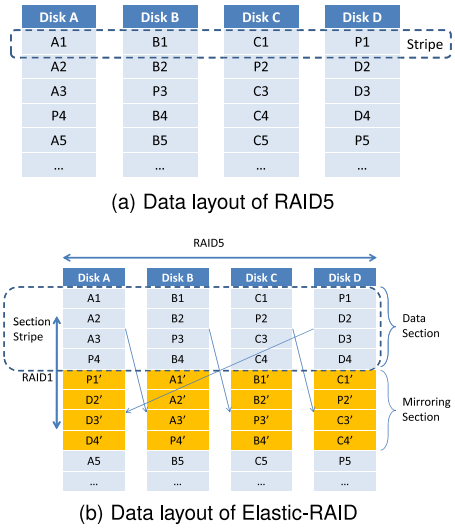
Upon a write request, Elastic-RAID first identifies the target disks of the request, including the disks storing the original data and the disks storing the parity/mirroring redundant data, then calculates the disk internal offsets according to the Elastic-RAID data layout scheme, before finally sending and completing the request to the individual target disks according to the design strategies described in Sections 3.1 and 3.2.

Upon a read request, Elastic-RAID first determines if this request falls on a failed disk. If yes, Elastic-RAID tries to directly read the mirroring copy of the requested data instead of reconstructing the data from all other surviving disks on the fly. If no, Elastic-RAID simply identifies the target disks of the request and calculates the disk internal offsets according to the data layout scheme, before sending the read request to the target disks.

During the reconstruction period, Elastic-RAID always tries to recover the lost data by directly reading the

TABLE 1
Variables and Definitions

Symbols	Definition
N	The number of disks in a RAID system
LBA_{in}	The LBA sent to a RAID system
LBA_5	The LBA inside a single disk in a RAID5 system
LBA_{er}	The LBA inside a single disk in an Elastic-RAID system
LBA'_{er}	The LBA of a mirrored data copy inside a disk in an Elastic-RAID system
LBA_{sec}	The LBA of a data section inside a disk in an Elastic-RAID system
LBA'_{sec}	The LBA of a mirroring section inside a disk in an Elastic-RAID system
L	The *section length*
C	The capacity of a RAID system
C_1	The capacity of a single disk
$disk_n$	The disk number of a data chunk
$disk'_n$	The disk number of a mirrored data chunk
R_{io_amp}	Ratio of internal I/Os to user I/Os



(a) Data layout of RAID5

(b) Data layout of Elastic-RAID

Fig. 4. Data layouts of RAID5 and elastic-RAID.

corresponding mirroring redundant (i.e., mirrored) data from other surviving disks whenever possible. If the mirrored data is also lost or does not exist due to insufficient free space, it performs the standard data recovery operations for parity-based RAIDs.

In what follows, we will discuss the key design issues of Elastic-RAID. For convenience, we summarize the notations and their definitions in Table 1.

4.2 Data Layout and Addressing

The data layout scheme is designed to provide improved reliability, small-write performance, and reconstruction performance without sacrificing the user I/O performance, as elaborated next.

4.2.1 Data Layout

In a standard parity-based RAID system, the data is accessed in the unit of a *data chunk* whose size is specified by the user when the parity-based RAID system is configured. Fig. 4a illustrates a four-disk RAID5 consisting of component disks A, B, C, and D. A_n, B_n, C_n, D_n and P_n, where $n = 1, 2, 3, 4, ...$, represent the data chunks and the parity chunks respectively. A *stripe* consists of data chunks and the parity chunk that have the same offsets in the component disks. For example, chunks A_1, B_1, C_1, and P_1 belong to the same *stripe*.

To add mirroring data redundancy to a parity-based RAID system, we introduce a new data layout for Elastic-RAID, which is illustrated in Fig. 4b using RAID5 as an example. Elastic-RAID uses *section* as the basic unit for performing mirroring operations while keeping *data chunk* as the basic I/O access unit as in the standard RAID5. A *section* in Elastic-RAID is of multiple *data chunks* in size that is indicated by *section length*. *Section length* can also be specified by the user at the Elastic-RAID configuration time. We prefer to specify variable *section lengths* according to workload characteristics because of our observation of its significant impact on the performance.

For the example in Fig. 4b, the Elastic-RAID system contains four disks, A, B, C, and D, where A_1, A_2, A_3, and P_4 constitute a *data section* and A'_1, A'_2, A'_3, and P'_4 constitute the corresponding *mirroring section*. To add to RAID5 the ability to tolerate a second disk failure, the corresponding *mirroring section* of a *data section* is generated, right-shifted and written to the adjacent disk in a circular way, as demonstrated in Fig. 4b. For example, the *data section* $\{A_1, A_2, A_3, P_4\}$ is on disk A and its corresponding *mirroring section* $\{A'_1, A'_2, A'_3, P'_4\}$ is on disk B, and so on. Similar to the *stripe* in RAID5, a *section stripe* in Elastic-RAID consists of the *sections* that have the same address in all component disks. A *data section stripe* and its corresponding *mirroring section stripe* are by default always adjacent in the vertical orientation, although they can be scattered in some situations, as described in Section 4.2.3.

In Elastic-RAID, for every section stripe that has a corresponding mirroring section stripe, it is guaranteed to be able to tolerate two arbitrary disk failures. The reason is that any failed section can be recovered either by reading its mirroring if the mirroring section on the right adjacent disk is alive, or by XOR calculation as a traditional RAID5 does in case of the failure of the right adjacent disk. Therefore in the scenario of two disk failures, if two failed disks are not adjacent, Elastic-RAID can recover the sections that have mirroring sections by reading their mirroring; otherwise, Elastic RAID first recovers the sections on the right one of the two failed disks by using mirroring data, and then recovers the sections on the other one by XORing with the involvement of previously recovered sections.

The reason why we right shift a section and write its mirroring to the adjacent disk is to ensure that at least one failed section can be recovered by reading mirroring when two disks are failed. The other failed section can be recovered by either reading mirroring (if its mirroring is alive) or reconstructing using RAID5 scheme. This means that *for every section stripe that has a corresponding mirroring section stripe in Elastic-RAID, it is guaranteed to be able to tolerate two arbitrary disk failures, regardless of the total amount of available free space*.

This unique mirroring-redundancy data layout of Elastic-RAID on top of the original RAID5 parity-redundancy data layout can be thought of as two orientations of data layout, with the former being *vertical* while the latter being

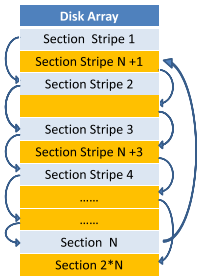


Fig. 5. Addressing of elastic-RAID.

horizontal. In the vertical orientation, every *data section stripe* has a corresponding *mirroring section stripe* that can be considered as RAID1. It is this combination of horizontal orientation (i.e., parity-based RAID) and vertical orientation (i.e., mirroring-based RAID) that can be used to exploit the advantages of both of these two RAID types. This vertical orientation of data layout has some similarity to and improves upon a non-standard RAID10 implemented in Linux [19] in that the non-standard RAID10 is now employed in a parity-based RAID to flexibly and smartly utilize the free space.

Unlike AutoRAID using non-uniform length stripes for two different RAID sets that causes more overhead to migrate data between RAID1 and RAID5, Elastic-RAID uses a uniform section stripe for both original data and mirroring data. This design reduces the complexity of implementation.

4.2.2 Addressing

To address an incoming I/O request whose address is LBA_{in}, we need to figure out the target disk number and the internal offset in the disk. Fig. 5 shows the addressing scheme. Section stripes *1* through *N* are interlaced with Section stripes *N + 1* through *2*N*, etc. The basic principle of the placement of the *mirroring section stripes* is to let a *mirroring section stripe* be as close to its corresponding *section stripe* as possible so as to reduce disk head movement. For example, after an Elastic-RAID is freshly built, the data written to *section stripe 1* will have a mirroring section stripe in *section stipe N + 1*, and so on.

Suppose that $disk_n$ and LBA_5 are respectively the target disk number and the internal offset on that disk calculated by the RAID5 scheme. According to the data layout of Elastic-RAID, the corresponding target disk number in Elastic-RAID is the same as in RAID5. To calculate the internal offset LBA_{er} of Elastic-RAID, we can use Equation (1) if LBA_{in} is less than half of C (the capacity of the whole RAID system), otherwise use Equation (2), which reflects the Elastic-RAID specific addressing.

$$LBA_{er} = \left\lfloor \frac{LBA_5}{L} \right\rfloor \times 2 \times L + LBA_5 \bmod L \quad (1)$$

$$LBA_{er} = \left\lfloor 1 + \frac{LBA_5 - C_1/2}{L} \right\rfloor \times L + LBA_5 \bmod L. \quad (2)$$

It is different between Elastic-RAID and AutoRAID in addressing. All incoming data to be written have fixed internal addresses in Elastic-RAID according to Equation (1) and Equation (2). That means Elastic-RAID can directly compute the internal addresses for the incoming I/Os instead of looking up a mapping table as in AutoRAID. This can simplify the implementation and increase performance.

4.2.3 Elastic Mirroring

The management of the mirroring data redundancy in Elastic-RAID is based on the notion of *section stripe*. In general, Elastic-RAID performs the allocation and release of *mirroring section stripes* upon a write request.

Elastic-RAID first tries to allocate the mirroring of the current write request locally in its *local section stripe pair*. If it fails due to the occupation of the desired *section stripe*, Elastic-RAID allocates a remote mirroring under specific conditions. If the capacity utilization is low (e.g., less than 50 percent), or the capacity utilization is high but the current data to be written is more important (e.g., meta data of the file system with the help of a semantics detection scheme [34]), Elastic-RAID uses the first fit algorithm to get a free *section stripe* to write the mirroring. Otherwise, there is no mirroring allocation for current write request.

For mirroring release, if Elastic-RAID detects the current written data's location is formerly used by a *mirroring section stripe*, or the TRIM command indicates the address range of a *section stripe* is freed, it just changes the state bits of related *section stripes* to complete the release, as explained next.

We employ a *section stripe bitmap* to track the status of *section stripes*. Each *section stripe* has two bits that represent four states: (1) a *section stripe* is free ("00"), meaning that the space either has never been used or the file system indicates it as invalid (e.g., via the TRIM command), (2) a *section stripe* is occupied (i.e., written with data) ("01"), (3) a *section stripe* is occupied and it is a *data section stripe* whose *mirroring section stripe* is adjacent to it ("10"), and (4) a *section stripe* is occupied and it is a *data section stripe* whose *mirroring section stripe* is not adjacent and to be determined by the *mirroring mapping table* ("11").

The *local section stripe pair* consists of a section stripe A and its adjacent section stripe A'. Suppose the offsets of A and A' in a component disk are LBA_A and $LBA_{A'}$ accordingly. When $LBA_A < C_1/2$, $LBA_{A'} = LBA_A + L$. Otherwise when $LBA_A >= C_1/2$, $LBA_{A'} = LBA_A - L$. The possible status combinations of *local section stripe pair* are shown in Table 2. *Remote mirroring* means the *mirroring section stripe* is not adjacent with the corresponding *data section stripe*.

The *mirroring mapping table* is further used to track the relationship between a *data section stripe* and its remote *mirroring section stripe*. When assigning a remote mirroring section stripe, this corresponding relationship is recorded in *mirroring mapping table* and state bits in the *mirroring mapping table* are updated as following: "11" for the data section stripe, "01" for the *mirroring section stripe*.

To support the performance-oriented design strategy described in Section 3.1, Elastic-RAID also has a *parity delay queue* that records the offsets of *data section stripes* required to compute and update the parity data later in idle or lightly loaded time periods in the background. For the reliability-oriented design strategy described in Section 3.2, all original data, new computed parities and their mirroring (if they have one) will be written at the same time. During the delay time windows, the original data and the corresponding

TABLE 2
Status Combination of Local Section Stripe Pair

Section stripe A	Section stripe A'	Description
00	00	Both A and A' are free
00	01	A is free, A' is occupied and has no mirroring
00	10	Invalid case
00	11	A is free, A' is occupied and has a remote mirroring indicated in *mirroring mapping table*
01	00	A is occupied and has no mirroring, A' is free
01	01	Both A and A' are occupied and have no mirroring
01	10	A is the mirroring of A'
01	11	A is occupied, A' is occupied and has a remote mirroring indicated in *mirroring mapping table*
10	00	Invalid case
10	01	A' is the mirroring of A
10	10	Invalid case
10	11	Invalid case
11	00	A is occupied and has a remote mirroring, A' is free
11	01	A is occupied and has a remote mirroring, A' is occupied
11	10	Invalid case
11	11	Both A and A' are occupied and have remote mirroring indicated in *mirroring mapping table*

mirroring data always are consistent, but their corresponding parities can be inconsistent. These parities eventually are updated and the corresponding records are removed from *parity delay queue*.

If there is an incoming write request in the address range of a *mirroring section stripe* (by checking *section stripe bitmap* and *mirroring mapping table*), the status bits of the corresponding *data section stripe* will be changed to "01" to indicate that it does not have mirroring any more. At the same time, if its former status is "11," the corresponding entry in *mirroring mapping table* will also be removed.

Fig. 6 shows an example illustrating the use of *section stripe bitmap*, *mirroring mapping table* and *parity delay queue*. The section stripe 1 is a *data section stripe* and the adjacent *section stripe* A is its local mirroring, respectively denoted by "10" and "01" in the *section stripe bitmap*. Both the *section stripe* 2 and 3 denoted by "11" have their remote mirroring *section stripes*, thus we need *mirroring mapping table* to record the locations of their mirroring *section stripes*. The parities of section stripes in *parity delay queue* are delayed to be updated.

To address the mirroring of a specific data request, we also need to obtain the disk number $disk'_n$ and the internal address LBA'_{er}. Suppose that the specific data request is on $disk_n$ and has an internal offset LBA_{er}. According to the data layout of Elastic-RAID, $disk'_n$ is always equal to the disk that is right adjacent to $disk_n$. To get LBA'_{er}, we need to check the *section stripe bitmap* and *mirroring mapping table* first. If the *section stripe* that contains the data at LBA_{er} has a "10" value in the bitmap, we use Equation (3) to calculate LBA'_{er} if LBA_{er} is less than $C_1/2$, otherwise use Equation (4)

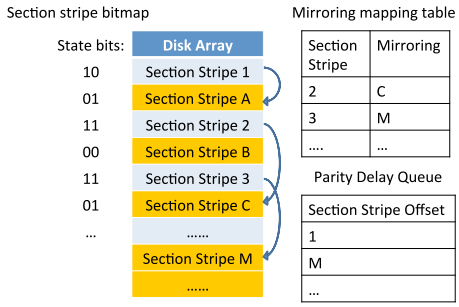


Fig. 6. Elastic mirroring.

to calculate LBA'_{er}. If it has a "11" value, we need to further query the *mirroring mapping table* to get the mirroring section offset LBA'_{sec}, then use Equation (5) to retrieve the data from LBA'_{er},

$$LBA'_{er} = LBA_{er} + L \tag{3}$$

$$LBA'_{er} = LBA_{er} - L \tag{4}$$

$$LBA'_{er} = LBA'_{sec} + LBA_{er} \bmod L. \tag{5}$$

The main memory overhead of Elastic-RAID comes from these data structures. We believe that this overhead is acceptable. For example, assuming a single-disk capacity of 1TB and a *section length* of 64 KB without any local mirroring (i.e., a worse-case scenario for Elastic-RAID), the typical *parity delayed queue* overhead is 4 MB (i.e., capable of tracking a maximum of 64 GB written data), the maximum *section stripe bitmap* and *mirroring mapping table* overhead is 4 MB and 64 MB respectively, for a total of 72 MB. For performance and consistency reasons, a small NVRAM (e.g., PCRAM) may be used for them. Alternatively, DRAM may be used, with a dedicated area in the top section of Elastic-RAID for persistent storage, duplicated on every component disk with a check-point version identifier, via a periodic flush process from DRAM. After a crash, these data structures can be retrieved from the NVRAM or surviving disks.

4.2.4 Performance Considerations

In Elastic-RAID, the read performance overhead mainly stems from the data layout where the *section length* parameter plays a key role. In practice, we find that the read performance degrades significantly when the specified section length is between 64 KB and 1 MB. The reason is twofold. First, since HDD is optimized for sequential I/O accesses and has considerable read-ahead length, a small *section length* will render much of the read-ahead data useless. Second, when the *section length* is sufficiently large, the disk will perform seek operations between two adjacent *sections*, which makes the read-ahead data meaningful and beneficial for user requests and internal I/Os. On the other hand, for the small random write requests during recovery, a large *section length* will prolong the response times because the locations of the mirroring copies are far from those of the corresponding original data, forcing the disk head to move back and forth frequently. A large *section length* also decreases the utilization efficiency of free space, since only an entire free *section stripe* can be used as *mirroring section stripe* to store mirroring data. Experimental results in

Section 5.3.4 demonstrate the impact of section length, and we use them as an optimization guidance to parameterize Elastic-RAID.

The other I/O performance overhead of Elastic-RAID is the need to update the mirrored copies upon a user's write request. Since it is an extra I/O for every write request, it is understandable that the write performance will be adversely impacted for the write-intensive workloads. Although the mirrored and original data are on different disks that can be performed in parallel, extra I/Os due to mirroring updates still impact the overall performance when the workload is write intensive. However, we argue that in typical storage workloads write intensity is generally lower than read intensity [28] so that Elastic-RAID is still suitable for typical I/O workloads, an argument that is in part also supported by our evaluation results presented in Section 5.

The additional mirroring redundancy also has a few advantages. First, it can significantly reduce I/Os when the RAID system is in the reconstruction state, thus improving the user I/O performance during recovery and shortening the recovery time. We define I/O amplification by the ratio R_{io_amp} for the RAID systems: $R_{io\_amp} = IO_{Internal}/IO_{User}$. For an eight-disk RAID5 system with one failed disk, if a user read request falls on the failed disk, the R_{io_amp} value for this request is 7 since all the seven surviving disks must be read to reconstruct the failed data. But for the Elastic-RAID system, it can reduce the R_{io_amp} value to 1 in this case by directly reading from the mirroring section. Considering the interactions among I/Os, a low R_{io_amp} value can provide more potential benefits. We provide a detailed discussion of this in Section 5.3.2.

The second advantage is that the small write performance can be optimized by adopting the performance-oriented design strategy as described in Section 3.1 and verified in our evaluations in Section 5.

The third advantage lies in the fact that the read performance in the operational state can also benefit from mirroring. By tracking the last accessed LBA (representing the position of the disk head) of each disk, we can decide to read the original data section or the mirroring section by selecting whichever that has the shortest distance from the last accessed LBA, since the original data section and the corresponding mirroring section are in different disks.

5 Evaluations

5.1 Experimental Setup

To evaluate the effectiveness and performance of our Elastic-RAID, we developed an Elastic-RAID prototype in the Linux software RAID (MD). Because MD is implemented as a standard block device in the Linux system, any workload in the real world can be run on top of it. The prototype Elastic-RAID is installed on Ubuntu Linux 11.04 $\times$86_64 (Kernel version 2.6.38) running on a Dell PowerEdge T110 with 2.53 GHz Intel Xeon CPU and 4 GB RAM. We use two Highpoint RocketRaid $2,640 \times 4$ SATA cards to house eight HDD disks and use two SATA ports on the motherboard to house the other two HDD disks. The 10 disks are all Seagate ST31000524AS SATA3.0: 1TB capacity, 7,200 RPM, and 32 MB cache.

TABLE 3
The Workload Characteristics

Trace	Write Ratio	IOps	Avg. I/O size
SPC WebSearch2	1%	334	15 KB
SPC Financial1	68%	69	7.5 KB
MSR Src1_0	2%	651	52 KB

We conduct trace-driven experiments with three I/O traces collected from real-world applications to evaluate the I/O performance of the Elastic-RAID against the parity-based standard RAID5 and RAID6 as the baselines. The Financial1 trace was collected from OLTP (On-Line Transaction Processing) applications running at a large financial institution. The WebSearch2 trace was collected from a machine running a web search engine. The MSR Src1_0 trace was collected from a machine used for source control. The first two are obtained from the *UMass Trace Repository* [3] while the third is from the *SNIA IOTTA repository* [1]. The characteristics of these three traces are summarized in Table 3. The Financial1 trace is write intensive while both the WebSearch2 and Src1_0 traces are read intensive.

To save trace replay time, we use one-hour-long pieces of the traces to conduct our experiments. The typical Elastic-RAID configuration consists of eight disks, where the *chunk size* and *section length* are set to 64 KB and 10 MB respectively. The same number of disks are used to build RAID5 and RAID6, with the same chunk size, for performance comparison. Since the maximum offset of all these three traces does not exceed half of the capacity of Elastic-RAID, all data chunks can have mirroring except for the experiments in Section 5.3.3, which are designed to test the performance at higher system capacities. Because the maximum LBA in the traces does not exceed 160 GB, we use the first 160 GB partition of each disk to build the disk array for the trace-driven experiments.

5.2 Evaluation Methodology

We use an open-loop [21], [30] benchmark model to replay the traces, and use the average I/O response time as the metric to evaluate the performance. In particular, the trace replay tool fio [6] is used and configured to use the libaio [2] library to perform asynchronous I/O operations. In reconstruction performance evaluations, we set the minimum reconstruction speed of MD to 200 MB per second to ensure the smallest MTTR (Mean Time To Repair).

For tests in the operational state, we conduct two groups of experiments corresponding to the performance oriented design strategy, described in Section 3.1 and labeled "Elastic-RAID-P," and the reliability oriented design strategy, described in Section 3.2 and labeled "Elastic-RAID-R," to show the elastic design impact on the small-write performance. For tests in the reconstruction state, we use only the reliability oriented design strategy because these two design strategies perform the same routines in the reconstruction state since the main objective for both of them is to complete the reconstruction as soon as possible, meaning that they both will try to minimize the internal I/Os whenever possible.

To evaluate the performance impact of different capacity utilization, we conduct two micro benchmark tests for both

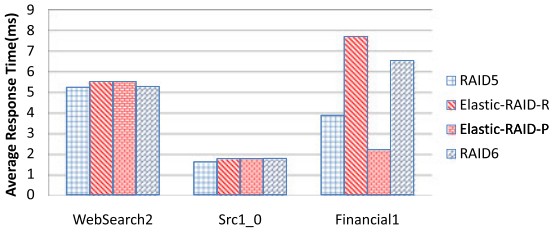


Fig. 7. Results in normal operational state.

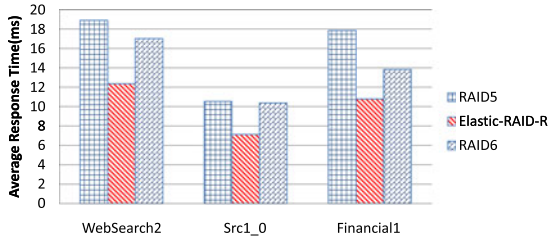


Fig. 8. Results in the reconstruction state.

of "Elastic-RAID-P" and "Elastic-RAID-R." We use fio to generate I/O access pattern and use IOps as the metric.

We also use micro benchmark tests to study the sensitivity of different section size and number of HDD disks.

5.3 Results and Analysis

5.3.1 Results in the Operational State

We replay the traces when the RAID systems are in the operational state to examine the performance overhead of Elastic-RAID caused by its data layout and mirroring updates. The average user response time for the three traces are shown in Fig. 7.

For the WebSearch2 trace, there is about 5 percent performance degradation for Elastic-RAID compared with the standard RAID5. Since WebSearch2 is read intensive, the impact of mirroring writes is negligible. The main reason for the performance degradation is the data layout of Elastic-RAID.

For the Src1_0 trace, the performance overhead relative to RAID5 increases to 10 percent since it has higher I/O intensity and sequentiality than WebSearch2. It is clear that the sequential pattern of I/Os plays an important role in Elastic-RAID's performance overhead due to its data layout scheme.

From Fig. 7 we can see that the reliability-oriented Elastic-RAID almost doubles the response time of RAID5 under the write-intensive Financial1 trace because of the additional mirroring updates. When the reliability is the top priority, the use of reliability-oriented Elastic-RAID may be justified. On the other hand, for the performance-sensitive applications with small-write-intensive workload characteristics, performance-oriented Elastic-RAID should be used to improve the small-write performance, as described in Section 3.1. The "Elastic-RAID-P" bar in Fig. 7 shows that the performance-oriented Elastic-RAID outperforms RAID5 by 40 percent under the write-intensive Financial1 trace.

Because of the additional parity updates required of RAID6, we note that the performance gap between Elastic-RAID-R and RAID6 is much smaller than the one between Elastic-RAID-R and RAID5 for these three traces. Similarly, performance-oriented Elastic-RAID significantly outperforms RAID6 under the write-intensive Financial1 trace.

5.3.2 Results in the Reconstruction State

One of the main goals of Elastic-RAID is to improve user I/O performance in the reconstruction state and reduce the recovery time. Figs. 8 and 9 show that Elastic-RAID achieves a reconstruction-performance boost of at least 30 percent over RAID5 by exploiting the mirroring redundancy during recovery under all traces. In particular, Elastic-RAID outperforms RAID5 in average user response time by 34.5, 32.3, 39.4 percent and in recovery time by 44.8, 52.9, 57.8 percent under WebSearch2, Src1_0, and Financial1 respectively. In addition, the performance of RAID6 is slightly better than RAID5 because the number of internal I/Os during reconstruction in RAID6 is smaller than that in RAID5 in the one-disk-failure case. This stems from the fact that the former, with two of the eight disks being parity disks, has fewer disks participating in the reconstruction process than in the latter.

The significant performance advantage of Elastic-RAID in the reconstruction state can be attributed mainly to its ability to substantially reduce the internal I/Os induced by parity updates. We define I/O amplification by the ratio R_{io_amp} for the RAID systems: $R_{io\_amp} = IO_{Internal}/IO_{User}$. Our Elastic-RAID prototype monitors and records the number of internal I/Os during the reconstruction period and uses this number to calculate R_{io_amp}. We also modified the MD source code to add similar functions in the standard RAID5 and RAID6 implementations. Fig. 10 shows the results, where the internal I/Os include the I/Os for reconstruction. We can see that the R_{io_amp} values of Elastic-RAID are substantially lower than those of RAID5 and RAID6 under all three traces. This means that, by reading the mirroring copies to reconstruct the lost data rather than by parity computation, the recovery scheme of Elastic-RAID is able to significantly reduce the internal I/O counts of the the standard RAID5 and RAID6 schemes. Obviously, this significant performance improvement in the reconstruction state translates to a higher reliability for the system and better I/O performance for the user applications, an increasingly important requirement for cloud storage nowadays.

5.3.3 Results for Different Capacity Utilization

As discussed in Section 3, the improvements of both reliability and performance depend on the capacity utilization. If the capacity utilization is low, the free space is high, and vice versa.

Fig. 11 shows the performance results of Elastic-RAID-P in 20, 40, 50, 60 and 80 percent capacity utilization conditions. For 4 KB block size random read and 4 KB block size sequential read, the performance of Elastic-RAID-P in

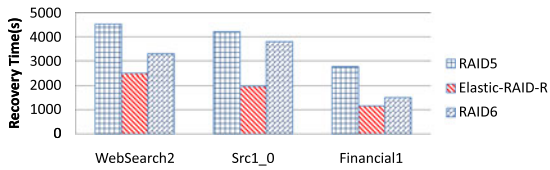


Fig. 9. The recovery time.

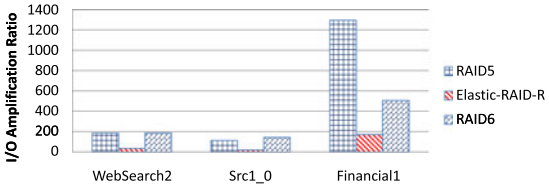


Fig. 10. The I/O amplification ratio in the reconstruction state.

different capacity utilization conditions has no obvious difference compare to the standard RAID5. For 64 KB block size sequential read, the performance of Elastic-RAID is slightly lower than the standard RAID5. That is reasonable due to the data layout of Elastic-RAID which is discussed in Section 4.2.4.

For write performance, that is the major optimization objective of Elastic-RAID-P, it shows the remarkable difference. For 4 KB block size random write, when the capacity utilization is below 50 percent, the performance of Elastic-RAID-P is about nine times higher than the standard RAID5. That is because the small writes in Elastic-RAID-P have been optimized to avoid the Read-Modify-Write routine by writing the mirroring. When the capacity utilization increases to 60 and 80 percent, the advantage of Elastic-RAID-P reduces because only some part of the writes which have mirroring can be optimized. For 4 and 64 KB block size sequential writes, the standard RAID5 uses Reconstruction-Write routine, which has lower overhead than Read-Modify-Write, thus reduces the performance gap compared to Elastic-RAID-P.

Fig. 12 shows the performance results of Elastic-RAID-R in different capacity utilization conditions. We can see the overall read performance of Elastic-RAID-R is slightly lower than the standard RAID5. The overall write performance is lower than the standard RAID5 due to the synchronous mirroring writes. For the 4 KB block size random write case, Elastic-RAID-R has a slightly lower performance, but for the sequential write cases, Elastic-RAID-R's performance has an obvious degradation, especially when the capacity utilization is 50 percent. When the capacity utilization increases to 60 and 80 percent, the performance goes up because only part of the written data which has mirroring generate the mirroring writes overhead.

5.3.4 Sensitivity Study

In general, the performance of Elastic-RAID is influenced by several factors. The *section length* discussed in Section 4.2.4 is an important factor. The number of disks can also affect the performance.

Section length. In the experiments studying the section length, the access patterns of 4 KB random I/Os and 64 KB

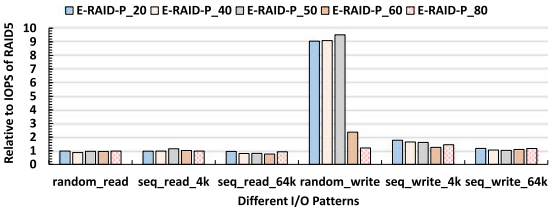


Fig. 11. The performance of Elastic-RAID-P in different capacity utilization.

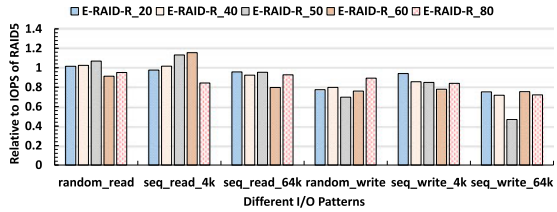


Fig. 12. The Performance of elastic-RAID-R in different capacity utilization.

sequential I/Os are assumed. We use IOps (I/O per second) as the metric to evaluate the random I/O performance and use the I/O throughput as the metric to evaluate the sequential I/O performance. The results of the random I/O tests are shown in Fig. 13, which suggests that the random I/O performance is relatively insensitive to the *section length*. On the other hand, the results of the sequential I/O tests, shown in Fig. 14, clearly indicate that the read performance is highly sensitive to the *section length*. When the *section length* is less than 10 MB, the performance is only about 50 percent of RAID5. Starting with a 10 MB section length, the performance of Elastic-RAID can reach 95 percent of RAID5. But as the *section length* continues to increase, the sequential read performance in the degraded state also decreases. This indicates that we must select a suitable value for the *section length* according to the specific environment (hardware platform, characteristics of workload, etc.) and users' requirements. On the other hand, we can see that the sequential and random write performances are insensitive to the section length, as shown in Figs. 14 and 13.

Number of disks. To evaluate how the number of disks affects the performance, we conduct the experiments on Elastic-RAID under the Financial1 trace by varying the number of disks, 6, 8, 10, respectively, with the section length set to 10 MB. Fig. 15 shows the test results in the operational state and in the reconstruction state respectively. Clearly, more disks imply higher I/O parallelism, which improves the performance. The average user response time, as shown in Fig. 15, is very sensitive to the number of disks.

5.4 Reliability Analysis

In this section, we analyze the reliability of Elastic-RAID and compare it with other standard RAID levels RAID5, RAID6 and RAID10.

For Elastic-RAID system, the reliability is dynamically changed with the amount of available free space. The Elastic-RAID system can tolerate any arbitrary two disk failures with less than half of its space usage. Otherwise, only the *data sections* that have their corresponding *mirroring sections* can tolerate two disk failures and thus be protected

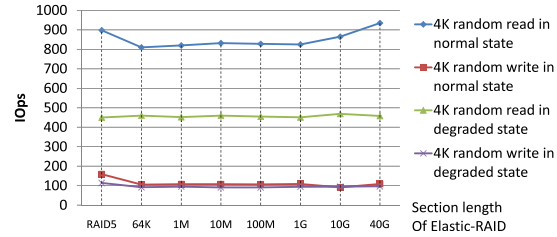


Fig. 13. Random I/O performance benchmark.

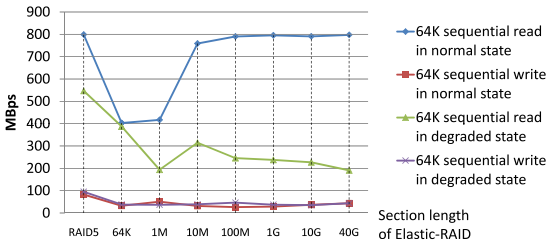


Fig. 14. Sequential I/O performance benchmark.

from data loss, while the rest of the *data sections* will only be able to tolerate a single disk failure as in the standard RAID5 system. When all the space is run out, Elastic-RAID degrades to the standard RAID5 with special addressing (see Section 4.2.2).

Since Elastic-RAID incorporates a RAID10-like mirroring redundancy, it can tolerate a maximum of $N/2$ disk failures in the same way as RAID10 does. That is, when N is an even number, and all odd-numbered disks $\{1, 3, 5, \ldots N/2-1\}$ or all even-numbered disks $\{2, 4, 6, \ldots N/2\}$ have failed. It is obvious that the lost data on the failed disks can be recovered by directly reading their mirroring copies that are on the adjacent disks.

To quantitatively compare the reliability of Elastic-RAID with other RAID levels, RAID5, RAID6, and RAID10, we use the Continuous-Time Markov Chain (CTMC) model [13] to compute the Mean Time To Data Loss (MTTDL) for different RAIDs. While the Sequential Monte-Carlo (SMC) model [9] is known to be more accurate than CTMC, the study on SPA [36] shows that the results generated by the CTMC model and the SMC model share the same trend. Since our primary goal for this reliability analysis is to obtain a first-order estimate and the relative strength of Elastic-RAID in reliability, we use the CTMC model here for its simplicity.

We use eight disks to build different RAID levels. The MTTF of disks are $750,000$ hours according to their datasheet. Due to the space constraints, we omit the detailed description on the reliability models of RAID5, RAID6, and RAID10, and the corresponding matrix constitution, deduction and solution processes to obtain their corresponding MTTDL, which is a standardized analysis procedure that can be found in [13], [37]. According to disk failing events in Elastic-RAID, the nine state transition diagram of the reliability model of Elastic-RAID is defined according to the corresponding transition conditions, as shown in Fig. 16. Note that since Elastic-RAID can exploit the parallelism between disks, the MTTR values at four cases are approximately equal as the recovery time for single disk failure. Finally, the MTTDL of Elastic-RAID can be similarly derived by the standardized analysis procedure.

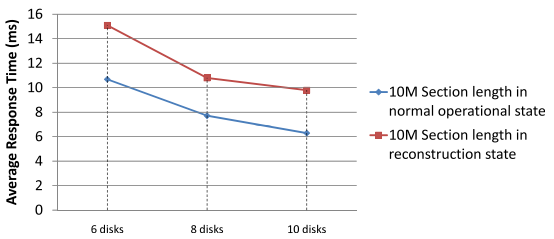


Fig. 15. Sensitivity tests for different number of disks.

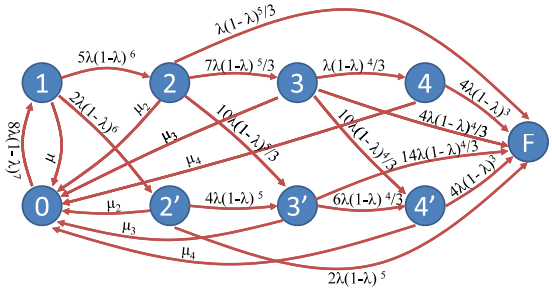


Fig. 16. State transition diagram for eight-disk Elastic-RAID. *State 0* denotes the normal operational state; *State 1* denotes the one-disk-failure state; *State 2* denotes the state in which two non-adjacent disks have failed; *State 2′* denotes the state in which two adjacent disks have failed; *State 3* denotes the three-disk-failure state in which no two failed disks are adjacent; *State 3′* denotes the three-disk-failure state in which two of the failed disks are adjacent; *State 4* denotes the four-disk-failure state in which no two failed disks are adjacent; *State 4′* denotes the four-disk-failure state in which two of the failed disks are adjacent; *State F* denotes the data loss state. λ denotes $1/MTTF$ of disks. μ, μ_2, μ_3, μ_4 denotes $1/MTTR$ for the corresponding failing states respectively.

Fig. 17 reveals the impact of the recovery time on the reliability (MTTDL) for the four RAID schemes. The MTTDL of Elastic-RAID with adequate free space is consistently higher than RAID6, and significantly higher than RAID5 and RAID10 under the recovery time from 0.5 to 10 hours. Note that this MTTDL is the upper bound of Elastic-RAID. With the increasing space utilization over half, more section stripes will lose their mirroring and the MTTDL of Elastic-RAID also gradually drops to the lower bound as that of RAID5. In addition, since the fast recovery time also significantly improves the reliability of each RAID level, Elastic-RAID can benefit from its accelerated reconstruction.

6 Related Works

6.1 Improving RAID Reliability

Several double-parity encoding mechanisms, known as RAID6 level, are proposed to further improve the reliability of RAID systems. Reed-Solomon code [26], EVENODD code [8], RDP code [12] and Liberation Codes [27] are designed to tolerate a second disk failure. As RAID6 coding techniques, all these schemes use fixed reserved space to store the additional parity and suffer from the overhead of two parity updating. IPC [13] also needs fixed reserved space within disks for storing interleaved parity check information. Different from the above approaches, Elastic-RAID improves RAID reliability through data mirroring by smartly utilizing the spare/free space existing in parity-based RAID systems.

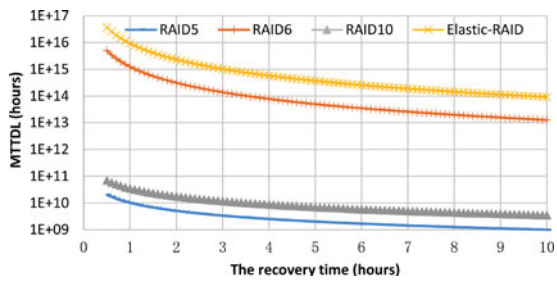


Fig. 17. Comparison of reliability.

6.2 Improving Reconstruction Performance

VDF [38] proposes an optimized caching scheme to improve the reconstruction performance. WorkOut [40] needs either extra RAID sets or spare disk(s) for its performance guarantee, and is thus not very space efficient. SPA [36] is designed to store and update supplementary parity units on the spare disk(s) for on-line RAID systems to improve the reconstruction performance and reliability. CRAID [22] uses a dedicated caching partition to capture hot data and redistributes this partition to incremental devices, thus accelerating the upgrade process of traditional RAIDs. In contrast to the above approaches, Elastic-RAID combines the advantage of both mirroring-based and parity-based RAID schemes to improve the performance in the reconstruction state. At the same time, it relies on the free space instead of extra dedicated spare disks.

7 Conclusions

In this paper, we propose Elastic-RAID, a new RAID architecture that combines the advantages of both mirroring-based and parity-based RAID systems by utilizing the free space existing in parity-based RAID systems to incorporate the mirroring redundancy. By employing both parity and mirroring data redundancy in a smart way, an Elastic-RAID provides higher reliability, better user I/O performance during recovery, and shorter reconstruction time. In doing so, Elastic-RAID is shown to significantly improve the overall availability of parity-based RAID systems.

Acknowledgments

This work is sponsored in part by the National Basic Research Program of China (973 Program) under Grant No. 2011CB302303. This work is also supported by Key Laboratory of Data Storage System, Ministry of Education of China. Qiang Cao is the corresponding author of this paper.

References

[1] Block i/o traces. [Online]. Available: http://iotta.snia.org/tracetypes/3, 2011.
[2] Kernel asynchronous i/o (aio) support for linux. [Online]. Available: http://lse.sourceforge.net/io/aio.html, 2011.
[3] Oltp application i/o and search engine i/o. [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage, 2007.
[4] Solid-state drive review. [Online]. Available: http://solid-state-drive-review.toptenreviews.com/, 2014.
[5] "Storage changes in Linux 2.6.33," [Online]. Available: http://kernelnewbies.org/Linux_2_6_33#headb9b8a40358aaef60a61fcf12e9055900709a1cfb, 2010.
[6] J. Axboe, fio: Flexible i/o tester. [Online]. Available: http://freshmeat.net/projects/fio/, 2011.
[7] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An analysis of latent sector errors in disk drives," in *Proc. Int. Conf. Meas. Modeling Computer Syst.*, New York, NY, USA, 2007, pp. 289–300.
[8] M. Blaum, J. Brady, J. Bruck, and J. Menon, "Evenodd: An efficient scheme for tolerating double disk failures in raid architectures," *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 192–202, Feb. 1995.
[9] C. Borges, D. Falcao, J. Mello, and A. Melo, "Composite reliability evaluation by sequential monte carlo simulation on parallel and distributed operating environments," *IEEE Trans. Power Syst.*, vol. 16, no. 2, pp. 203–209, May 2001.
[10] P. H. Carns, S. Lang, R. B. Ross, M. Vilayannur, J. M. Kunkel, and T. Ludwig, "Small-file access in parallel file systems," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2009, pp. 1–11.
[11] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "Raid: High-performance, reliable secondary storage," *ACM Comput. Surv.*, vol. 26, no. 2, pp. 145–185, Jun. 1994.
[12] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Awarded best paper!-row-diagonal parity for double disk failure correction," in *Proc. 3rd USENIX Conf. File Storage Technol.*, Berkeley, CA, USA, 2004, pp. 1–14.
[13] A. Dholakia, E. Eleftheriou, X.-Y. Hu, I. Iliadis, J. Menon, and K. Rao, "A new intra-disk redundancy scheme for high-reliability raid storage systems in the presence of unrecoverable errors," *Trans. Storage*, vol. 4, pp. 1:1–1:42, May 2008.
[14] S. Foskett, "Integration," in *Storage Magazine*, Apr. 2003.
[15] G. A. Gibson, "Reflections on failure in post-terascale parallel computing," in *Proc. Int'l. Conf. Parallel Process. (ICPP '07)*, Sep. 2007.
[16] M. Holland and G. A. Gibson, "Parity declustering for continuous operation in redundant disk arrays," in *Proc. 5th Int. Conf. Archit. Support Programm. Lang. Operating Syst.*, New York, NY, USA, 1992, pp. 23–35.
[17] Y. Hu and Q. Yang, "Dcddisk caching disk: A new approach for boosting i/o performance," in *Proc. 23rd Annu. Int. Symp. Comput. Archit.*, New York, NY, USA, 1996 pp. 169–178.
[18] W. Jiang, C. Hu, Y. Zhou, and A. Kanevsky, "Are disks the dominant contributor for storage failures?: A comprehensive study of storage subsystem failure characteristics," in *Proc. 6th USENIX Conf. File Storage Technol.*, Berkeley, CA, USA, 2008, pp. 8:1–8:15.
[19] J. B. Layton, "Intro to nested-raid: Raid-01 and raid-10," in *Linux Magazine*, Jun. 2011.
[20] M. Levin. (2006). Storage management disciplines are declining. [Online]. Available: http://www.computereconomics.com/article.cfm?id=1129
[21] M. P. Mesnier, M. Wachs, R. R. Sambasivan, J. Lopez, J. Hendricks, G. R. Ganger, and D. O'Hallaron, "Trace: Parallel trace replay with approximate causal events," in *Proc. 5th USENIX Conf. File Storage Technol.*, Berkeley, CA, USA, 2007, pp. 24–24.
[22] A. Miranda and T. Cortes, "Craid: Online raid upgrades using dynamic hot data reorganization," in *Proc. 12th USENIX Conf. File Storage Technol.*, 2014, pp. 133–146.
[23] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (raid)," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, New York, NY, USA, 1988, pp. 109–116.
[24] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *Proc. 5th USENIX Conf. File Storage Technol.*, Berkeley, CA, USA, 2007, pp. 2–2.
[25] J. S. Plank, M. Blaum, and J. L. Hafner, "SD codes: Erasure codes designed for how storage systems really fail," in *Proc. 11th Usenix Conf. File Storage Technol. (FAST '13)*, Berkeley, CA, USA, Feb. 2013, pp. 95–104.
[26] J. S. Plank, "A tutorial on reed-solomon coding for fault-tolerance in raid-like systems," *Softw. Pract. Exp.*, vol. 27, pp. 995–1012, Sep. 1997.
[27] J. S. Plank, "The raid-6 liberation codes," in *Proc. 6th USENIX Conf. File Storage Technol.*, Berkeley, CA, USA, 2008, pp. 7:1–7:14.
[28] D. Roselli, J. R. Lorch, and T. E. Anderson, "A comparison of file system workloads," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf.*, Berkeley, CA, USA, 2000, pp. 4–4.
[29] 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 *Proc. 5th USENIX Conf. File Storage Technol.*, Berkeley, CA, USA, 2007.
[30] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open versus closed: A cautionary tale," in *Proc. 3rd Conf. Netw. Syst. Des. Implementation-Volume 3*, Berkeley, CA, USA, 2006, pp. 18–18.
[31] F. Shu, "Data set management commands proposal for ata8-acs2," *T13 Technical Committee, United States: At Attachment:e07154r1*, 2007.
[32] F. Shu, "Windows 7 enhancements for solid-state drives," [Online]. Available: http://download.microsoft.com/download/F/A/7/FA70E919-8F82-4C4E-8D02-97DB3CF79AD5/CORT558_Shu_Taiwan.pdf, 2008.
[33] M. Sivathanu, L. N. Bairavasundaram, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Life or death at block-level," in *Proc. 6th Conf. Symp. Opearting Syst. Des. Implementation-Volume 6*, Berkeley, CA, USA, 2004 pp. 26–26.
[34] M. Sivathanu, V. Prabhakaran, F. I. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Semantically-smart disk systems," in *Proc. 2nd USENIX Conf. File Storage Technol.*, Berkeley, CA, USA, 2003, pp. 73–88.

[35] P. Steege. (2006). 50 storage utilization: Are data centers half empty or half full?. [Online]. Available: http://storageeffect.media.seagate.com/2009/0-1/storageeffect/50-stora ge-utilization-are-datacenters-half-empty-orhalf-full/
[36] L. Tian, Q. Cao, H. Jiang, D. Feng, C. Xie, and Q. Xin, "Online availability upgrades for parity-based raids through supplementary parity augmentations," *Trans. Storage*, vol. 6, pp. 17:1–17:23, Jun. 2011.
[37] K. S. Trivedi, *Probabilistic and Statistics with Reliability, Queueing and Computer Science Applications*, 2nd ed. New york, NY, USA: Wiley, 2002.
[38] S. Wan, Q. Cao, J. Huang, S. Li, X. Li, S. Zhan, L. Yu, C. Xie, and X. He, "Victim disk first: An asymmetric cache to boost the performance of disk arrays under faulty conditions," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, Berkeley, CA, USA, 2011, pp. 13–13.
[39] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan, "The hp autoraid hierarchical storage system," in *Proc. 15th ACM Symp. Operating Syst. Principles*, New York, NY, USA, 1995, pp. 96–108.
[40] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao, "Workout: I/o workload outsourcing for boosting raid reconstruction performance," in *Proc. 7th Conf. File Storage Technol.*, Berkeley, CA, USA, 2009, pp. 239–252.
[41] B. Xie, J. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, and N. Podhorszki, "Characterizing output bottlenecks in a supercomputer," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Los Alamitos, CA, USA, 2012, pp. 8:1–8:11.
[42] X. Zhang, K. Davis, and S. Jiang, "itransformer: Using SSD to improve disk scheduling for high-performance i/o," in *Proc. IEEE 26th Int. Parallel Distrib. Process. Symp.*, 2012, pp. 715–726.

Jie Yao received the BS degree in computer science and technology and the MS and PhD degrees in computer architecture in 2001, 2004, and 2009, respectively, from the Huazhong University of Science and Technology, Wuhan, China. He is currently a lecturer at the Huazhong University of Science and Technology. His research interests include computer architecture, large scale storage systems, and performance evaluation. He is a member of China Computer Federation (CCF) and a member of the IEEE and ACM.

Hong Jiang received the BSc degree in computer engineering in 1982 from the Huazhong University of Science and Technology, Wuhan, China, the MASc degree in computer engineering in 1987 from the University of Toronto, Toronto, Canada, and the PhD degree in computer science in 1991 from the Texas A&M University, College Station, Texas. Since August 1991, he has been at the University of Nebraska-Lincoln, Lincoln, Nebraska, where he is Willa Cather professor of computer science and engineering. He is currently on leave as a program director at the National Science Foundation. At UNL, he has graduated 13 PhD students who upon their graduations either landed academic tenure-track positions in PhD-granting US institutions or were employed by major US IT corporations. His present research interests include computer architecture, computer storage systems and parallel I/O, high-performance computing, big data computing, cloud computing, and performance evaluation. He recently served as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. He has more than 200 publications in major journals and international Conferences in these areas, including *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *ACM Transactions on Architecture and Code Optimization*, *Journal of Parallel and Distributed Computing*, *International Science Congress Association*, MICRO, USENIX ATC, FAST, LISA, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, INFOCOM, ICPP, etc., and his research has been supported by US National Science Foundation (NSF), DOD and the State of Nebraska. He is a fellow of the IEEE, and a member of ACM.

Qiang Cao received the BS degree in applied physics in 1997 from Nanjing University, the MS degree in computer technology, and the PhD degree in computer architecture in 2000 and 2003, respectively, from the Huazhong University of Science and Technology. He is currently a full professor of the Wuhan National Laboratory for Optoelectronics at the Huazhong University of Science and Technology. His research interests include computer architecture, large scale storage systems, and performance evaluation. He is a senior member of China Computer Federation (CCF) and a member of the IEEE and ACM.

Lei Tian received the PhD degree in computer engineering the from Huazhong University of Science and Technology in 2010. He is a senior member of Technical Staff at Tintri. Prior to joining Tintri, he was a research assistant professor in the Department of Computer Science and Engineering at the University of Nebraska-Lincoln. His research interests mainly lie in storage systems, distributed systems, cloud computing and big data. He has more than 40 publications in major journals and conferences including FAST, HOTSTORAGE, ICS, SC, HPDC, ICDCS, MSST, MASCOTS, ICPP, IPDPS, CLUSTER, *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, *ACM Transactions on Storage*, etc.

Changsheng Xie received the BS and MS degrees in computer science both from the Huazhong University of Science and Technology, Wuhan, China, in 1982 and 1988, respectively. He is currently a professor in the Department of Computer Engineering at HUST. He is also the director of the Data Storage Systems Laboratory of HUST and the deputy director of the Wuhan National Laboratory for Optoelectronics. His research interests include computer architecture, I/O system, and networked storage system. He is the vice chair of the expert committee of Storage Networking Industry Association (SNIA), China. He is a member of the IEEE.