S^2-RAID: Parallel RAID Architecture for Fast Data Recovery

Jiguang Wan, Jibin Wang, Changsheng Xie, and Qing Yang, *Fellow, IEEE*

Abstract—As disk volume grows rapidly with terabyte disk becoming a norm, RAID reconstruction process in case of a failure takes prohibitively long time. This paper presents a new RAID architecture, S^2-RAID, allowing the disk array to reconstruct very quickly in case of a disk failure. The idea is to form skewed sub-arrays in the RAID structure so that reconstruction can be done in parallel dramatically speeding up data reconstruction process and hence minimizing the chance of data loss. We analyse the data recovery ability of this architecture and show its good scalability. A prototype S^2-RAID system has been built and implemented in the Linux operating system for the purpose of evaluating its performance potential. Real world I/O traces including SPC, Microsoft, and a collection of a production environment have been used to measure the performance of S^2-RAID as compared to existing baseline software RAID5, Parity Declustering, and RAID50. Experimental results show that our new S^2-RAID speeds up data reconstruction time by a factor 2 to 4 compared to the traditional RAID. Meanwhile, S^2-RAID keeps comparable production performance to that of the baseline RAID layouts while online RAID reconstruction is in progress.

Index Terms—Data storage, parallel reconstruction, RAID reconstruction, S^2-RAID.

1 Introduction

RAID is the de facto storage architecture [16] that has been widely used to store petabyte scale data as information keeps growing exponentially. In such large scale storage systems, disk failures will become daily events if not more frequent [6]. Therefore, being able to quickly rebuild disk array in case of a failure event has become critical to today's information services that cover every corner of our society nowadays. There are two key issues that make fast reconstruction of RAID upon failure essential. 1) Any additional failure during the reconstruction process may result in data loss especially for the RAID levels with single fault-tolerant layouts. Hence this reconstruction time is often referred to as "window of vulnerability" [28] that should be as small as possible. 2) Data services are either stopped completely for offline RAID reconstruction or productivity is negatively impacted for online RAID reconstruction that interferes with production I/Os.

While fast RAID rebuilding is important to minimize the "window of vulnerability", current technology trend adversely affects such reconstruction time. Disk volume continues to grow rapidly with terabytes disks becoming a norm whereas disk bandwidth and access time including seek time and rotation latency improve little. As a result, recovering terabytes of data on a failed disk of the traditional RAID architecture will take prohibitively long time increasing the chance of data loss. Such technology trend is likely to continue in the foreseeable future.

The requirement of fast RAID reconstruction coupled with the technology trend motivates us to seek for a new RAID architecture that allows high speed online RAID reconstruction but has as little negative performance impact on production I/Os as possible. This paper presents a new Skewed Sub-array RAID structure, S^2-RAID for short. The idea is to divide each large disk in the RAID into small partitions. Partitions on these disks form sub-arrays. The sub-arrays are skewed among the disks in the RAID in such a way that conflict-free parallelism is achieved during a RAID reconstruction when any disk fails. Recovered data that was on the failed disk is stored in parallel on multiple disks consisting of spare disks and available space of good disks. The parallel reading and writing of S^2-RAID can substantially speed up the RAID rebuilding process and reduce negative performance impact on service time while RAID reconstruction is going on in background.

In order to validate our design concept, we have designed and implemented a prototype software S^2-RAID on Linux OS at block device level based on the widely used software RAID, MD (multiple device). The prototype is installed inside an iSCSI target to provide data storage services to iSCSI initiators. Using the S^2-RAID prototype, we have carried out extensive experiments using real world I/O traces such as SPC [4], [17], Microsoft Research [15], and a collection of traces from production environment. We measure I/O performance, reconstruction time, and performance impact of online

- *Jiguang Wan and *Changsheng Xie are with Wuhan National Laboratory For Optoelectronics, Huazhong University of Science and Technology, 430074, Wuhan, Hubei, P.R. China.*
  *E-mail: jgwan@mail.hust.edu.cn, *corresponding author: cs_xie@mail.hust.edu.cn.*
- *Jibin Wang is with the school of computer of science and technology, Huazhong University of Science and Technology,430074, Wuhan, Hubei, P.R. China.*
  E-mail: wangjibin@gmail.com.
- *Qing Yang is with the Department of Electrical and Computer Engineering, University of Rhode Island, Kingston, RI, 02881, USA.*
  E-mail: qyang@ele.uri.edu.

reconstruction on production I/O performance. Experimental results show that S^2-RAID improves RAID reconstruction speed of the baseline RAID by a factor of 2 to 4. The frontend application performance while rebuilding RAID online using S^2-RAID keeps comparable to that of the baseline software RAID for most I/O workloads.

This paper is a substantial extension and provides comprehensive treatment of our preliminary study presented at [23]. It covers a wide variety of conditions and gives its performance evaluations. Secondly, we consider other two baselines as the comparisons with S^2-RAID. Furthermore, our performance measurements include many real world I/O traces in the evaluation section to show the performance of all aspects.

The paper is organized as follows. Section 2 presents the design and data layout of S^2-RAID. Section 3 discusses S^2-RAID's expandability and scalability. Experimental settings and workload characteristics are presented in Section 4. In Section 5, we discuss numerical results on performance, followed by the related work in Section 6. Section 7 concludes the paper.

2 S^2-RAID Design and Data Layout

2.1 Sub-array Formation

Consider a traditional RAID5 storage system. When a disk had failed, data in the failed disk would have been rebuilt by reading a data stripe remained in good disks, performing an Exclusive-OR computation, and writing rebuilt data in a spare disk. This process continues until all data chunks in the failed disk are reconstructed. The reconstruction bandwidth is ultimately limited by the single data stream being rebuilt, one data chunk at a time. To rebuild over terabyte of data on a failed disk, it is clearly going to take a very long time.

TABLE 1
List of Notations and Symbols Used in This Paper.

Symbol	Description
Di	the $(i+1)^{th}$ data disk label
Si	the $(i+1)^{th}$ spare disk label
m_i	subRAID data layout mapping table after shift operations based on the mapping table m_{i-1}
M	data layout mapping table for S^2-RAID
R	total number of disks in RAID array
G	number of disks in a group
K	total number of involved partitions for each disk
N	total number of groups in S^2-RAID array
β	parallel ratio of S^2-RAID
$P_{i,j}$	elements vector which denotes subRAID numbers of the $(j+1)^{th}$ partition on disks of $(i+1)^{th}$ group in S^2-RAID
$SH(P_{i,j})$	function that returns a vector for each $P_{i,j}$ through shift operations
$S.L$	disk partition label of the logic disk number L in the subRAID numbered S

Our objective is to speed up this rebuilding process by reading multiple data stripes, performing multiple Exclusive-OR computations, and writing rebuilt data chunks to multiple spare disks all in parallel. The multiple spare disks can be either physical spare disks or spare logic partitions available on good data disks. To facilitate our discussion, we summarize notations and symbols used in Table 1.

In order to achieve our objective of parallel data reconstruction, we would like to be able to read, in parallel and conflict-free, all multiple data stripes that are in remaining good disks and are needed to rebuild multiple data chunks in the failed disk. For this purpose, we divide each one of total R (not including hot spare disks) disks in the RAID into K partitions. As a result of this partitioning, we have $K \times R$ logic disk partitions. We then divide R physical disks into groups of size G resulting in $N=\lceil R/G \rceil$ physical disk groups, where G is a prime number. subRAIDs are then formed by picking up one disk partition from each one of N groups giving rise to subRAIDs of size G. Note that $1 < K$, $N \leq G$. For the sake of easy understanding, let us start our discussion on the case where N equals G. Let β be defined as degree of parallelism that is related to K. The key to S^2-RAID is how to form subRAIDs and position these subRAIDs among the R physical disks in the way that rebuilding data of any failed physical disk can be done in maximal parallelism. That is, parallel readings of multiple subRAIDs are conflict-free during data reconstruction.

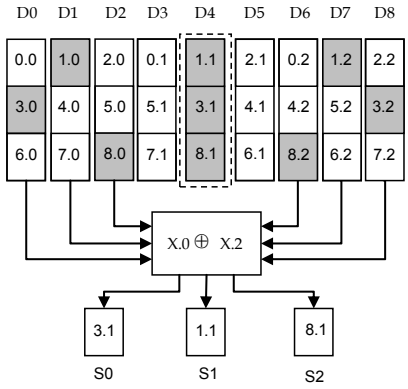

Fig. 1. An overview S^2-RAID5 data layout.

Let notation $S.L$ represent logic disk number L in subRAID number S. Fig. 1 shows an example of such subRAIDs formation with $R = 9$ and $G = N = \beta = 3$. SubRAID 0 uses physical disks D0, D3, and D6 to store logic disks 0, 1, and 2, respectively. Similarly, subRAID 1 uses physical disks D1, D4, and D7 to store logic disks 0, 1, and 2, respectively, as shown in Fig. 1. In this example, we have total 9 subRAIDs each of which contains 3 logic disks. Such skewed placement of subRAIDs allows maximal parallelism of data reconstruction.

Suppose physical disk D4 failed resulting in loss of logic partitions of 1.1, 3.1, and 8.1 as shown in shaded area of Fig. 1. To rebuild the lost logic partitions, we would need 1.0 and 1.2 to rebuild 1.1, 3.0 and 3.2 to rebuild 3.1, and 8.0 and 8.2 to rebuild 8.1. The placement

of the subRAIDs in Fig. 1 assures that all the needed data stripes to rebuild the lost data chunks reside on different physical disks allowing parallel and conflict-free reconstruction. S0, S1, and S2 in Fig. 1 are spare disks to store newly rebuilt data.

One may notice that RAID disks D3 and D5 were not involved in the above reconstruction process when we used spare disks S0, S1, and S2. Assuming there are some free spaces in these disks, D3 and D5 can play the role of spare disks. That is, the rebuilt partitions 1.1 and 3.1 can be written to the free partitions of disks D3 and D5 respectively. In this way, only one spare disk is needed. Alternatively, if we do have 3 spare disks, disks D3 and D5 can serve frontend application I/Os exclusively.

In general, consider a RAID system with R disks that are divided into N groups of size G each. Let $P_{i,j}$ be a G elements vector representing subRAIDs of partition j on disks of group i in the RAID. For example, in Fig. 1, we divide 9 disks into 3 groups with group 0 consisting of D1, D2, and D3; group 1 consisting of D3, D4, and D5; and group 2 consisting of D6, D7, and D8. Therefore, $P_{0,0} = (0 \quad 1 \quad 2)$ gives subRAID numbers of partition 0 on D0, D1, and D2 of group 0. $P_{1,1} = (5 \quad 3 \quad 4)$ gives subRAID numbers of partition 1 of the same group of disks, group 1. $P_{2,2} = (8 \quad 6 \quad 7)$ represents subRAID numbers of partition 2 of group 2 containing disks D6, D7, and D8, and so forth.

SubRAIDs are mapped to physical disks in S^2-RAID5 using a mapping table defined by a matrix M, where $M = (m_0, m_1, \ldots, m_{N-1})$. Each sub-matrix m_i in mapping table M is defined recursively by

$$m_0 = \begin{pmatrix} P_{0,0} \\ P_{0,1} \\ P_{0,2} \\ \cdots \\ P_{0,K-1} \end{pmatrix}, m_1 = \begin{pmatrix} P_{1,0} \\ P_{1,1} \\ P_{1,2} \\ \cdots \\ P_{1,K-1} \end{pmatrix} = \begin{pmatrix} SH_r^0(P_{0,0}) \\ SH_r^1(P_{0,1}) \\ SH_r^2(P_{0,2}) \\ \cdots \\ SH_r^{K-1}(P_{0,K-1}) \end{pmatrix}$$

,

$$m_i = \begin{pmatrix} P_{i,0} \\ P_{i,1} \\ P_{i,2} \\ \cdots \\ P_{i,K-1} \end{pmatrix} = \begin{pmatrix} SH_r^0(P_{i-1,0}) \\ SH_r^1(P_{i-1,1}) \\ SH_r^2(P_{i-1,2}) \\ \cdots \\ SH_r^{K-1}(P_{i-1,K-1}) \end{pmatrix},$$

..., and

$$m_{N-1} = \begin{pmatrix} SH_r^0(P_{N-2,0}) \\ SH_r^1(P_{N-2,1}) \\ SH_r^2(P_{N-2,2}) \\ \cdots \\ SH_r^{K-1}(P_{N-2,K-1}) \end{pmatrix}, \quad (1)$$

where $SH_r^b(P_{i,j})$ is a cyclic shift operator that shifts vector $P_{i,j}$ cyclically to the right by b positions and we denote the shift direction to right as r. For example, $SH_r^1(P_{1,0}) = SH_r^1(3 \quad 4 \quad 5) = (5 \quad 3 \quad 4)$.

Consider again the example shown in Fig. 1. We have

$$m_0 = \begin{pmatrix} P_{0,0} \\ P_{0,1} \\ P_{0,2} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix},$$

$$m_1 = \begin{pmatrix} P_{1,0} \\ P_{1,1} \\ P_{1,2} \end{pmatrix} = \begin{pmatrix} SH_r^0(P_{0,0}) \\ SH_r^1(P_{0,1}) \\ SH_r^2(P_{0,2}) \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 \\ 5 & 3 & 4 \\ 7 & 8 & 6 \end{pmatrix},$$

and

$$m_2 = \begin{pmatrix} P_{2,0} \\ P_{2,1} \\ P_{2,2} \end{pmatrix} = \begin{pmatrix} SH_r^0(P_{1,0}) \\ SH_r^1(P_{1,1}) \\ SH_r^2(P_{1,2}) \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 \\ 4 & 5 & 3 \\ 8 & 6 & 7 \end{pmatrix}.$$

The final mapping table M is given by

$$M = (m_0, m_1, m_2) = \begin{pmatrix} 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\ 3 & 4 & 5 & 5 & 3 & 4 & 4 & 5 & 3 \\ 6 & 7 & 8 & 7 & 8 & 6 & 8 & 6 & 7 \end{pmatrix}.$$

The resultant mapping of subRAIDs to disks is shown in Fig. 1. Recall the notation of *S.L* where *S* represents subRAID number whereas *L* represents logic disk number. Mapping subRAID to disks in this way can achieve maximal parallelism during data reconstruction, as will be discussed in section 2.2. While it is an optimal distribution of subRAID, there are limitations because of the requirement that the size of each subRAID group G (except subRAID10) must be a prime number. Proof of this fact is given in Appendix A.1 in Supplemental File. Besides, we will discuss more flexible but suboptimal subRAID distributions in Section 3.

The design concept presented above can also be applied to other RAID levels. For instance, instead of striping data regularly as done in traditional RAID10, S^2-RAID10 forms subRAIDs using smaller partitions and skew data partitions in a similar way as S^2-RAID5.

2.2 Mapping of Sub-arrays to LUNs

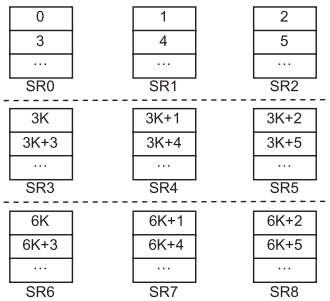

Fig. 2. Mapping of subRAIDs (SR for short) to LUNs.

In this section, we give the method of how the unified addressing space of S^2-RAID is constructed. From above description we know, S^2-RAID divides disks into subRAIDs resulting in smaller stripe sizes that may

adversely affect overall I/O throughput during normal data services. However, such performance impact can be easily eliminated by proper mapping of LUNs seen by clients to the S^2-RAID storage system.

Fig. 2 shows an example of mapping of subRAID to user LUNs allowing the same level of parallelism as the traditional RAID. This mapping is based on the S^2-RAID shown in Fig. 1. Suppose each subRAID can hold *K* units of data. Users' data will be stored on the first three subRAIDs, SR0, SR1, and SR2, first. We store data units 0, 1, 2 on SR0, SR1, and SR2, respectively. Each of these three data units is composed of two data chunks and one parity chunk to be stored on three separate physical disks (refer to Fig. 1). The three data units together form a stripe as seen by storage users across the three sub-arrays. As a result, the 3 data units are physically stored on 9 disks. This organization is similar to RAID5+RAID0 or RAID50.

After the first three subRAIDs are filled up with *K* data units each, we move on to the next three subRAIDs, SR3, SR4, and SR5 starting from data unit 3*K* as shown in Fig. 2. The same process repeats until all the three subRAIDs are filled out. After that, we move on to the next three subRAIDs and so forth. This mapping ensures that storage users see the S^2-RAID in the same way as the original RAID in terms of data striping and parallel disk accesses. All the subRAID partitions and data mappings are done transparently to users at lower storage level.

However, there is an additional storage overhead brought by S^2-RAID to store additional parity blocks. For the example in Fig. 1, S^2-RAID5 requires one parity chunk for every two data chunks because the subRAID size is three. The original RAID, on the other hand, needs 1 parity chunk for every 8 data chunks because the stripe size is 8 plus one parity. Such additional overhead can be easily justified with increased disk volume that is very inexpensive and the increased importance of data reliability and availability.

2.3 Reliability of S^2-RAID5

While each subRAID5 in S^2-RAID5 provides fault tolerance in the same way as traditional RAID5, as a whole storage system our S^2-RAID5 can tolerate multiple disk failures in many cases. This is an extra benefit that S^2-RAID5 architecture offers in addition to the high reconstruction performance when disk failures occur.

Fault tolerance of multiple disk failures of S^2-RAID5 can be achieved because of the fact that every stripe is stored across different groups. Lost data in any group can be reconstructed from remaining groups. As a result, even all *G* disks failed in the same group, their data can be reconstructed from remaining $N-1$ groups. Therefore, S^2-RAID5 can tolerate up to *G* disk failures provided that all these failures happen to be in the same group.

Although data reconstruction of multiple failed disks may take longer time than that of single disk failure, it is expected that the probability of simultaneous failures of multiple disks is small. However, our experiments have shown that the reconstruction time of three disk failures is comparable to the reconstruction time of single disk failure of traditional RAID5 and RAID50. Substantial shorter reconstruction time have also demonstrated the superb advantage of our S^2-RAID5 to minimize data loss. Therefore, we claim that our S^2-RAID5 provides much higher reliability than traditional RAID systems.

3 S^2-RAID5 Expansion Strategy

As we can see from the reconstruction process, conflict-free parallel operations in S^2-RAID5 require extra spare disks, and thereby give rise to high hardware cost. Furthermore, strictly conflict-free constraints the RAID system to only a limited number of configurations. To generalize S^2-RAID5 to more configurations, this subsection presents several methods that can configure a general S^2-RAID5 flexibly although they are not the optimal solutions. Storage designers can exercise tradeoffs between high reconstruction performance and low cost in S^2-RAID5 configurations by tuning subRAID size and the number of disk partitions. Besides, we also propose the layout of S^2-RAID10 in Appendices A.2 and A.3, and their evaluations in Appendix C.4 in Supplemental File.

3.1 Conventional Data Layout Strategy

The following example illustrates one method of generalizing S^2-RAID5 with $R = 16$. Since *R* is in the range of 3^2 and 5^2, the matrix layout of these disks can adopt the final mapping table of $G = 5$. And 25 disks are divided into five groups as shown in Fig. 3. Then we can generate four matrixes m_1, m_2, m_3, and m_4 based on the initial matrix m_0 and have 25 subRAIDs in total. The first sixteen columns of the mapping table can be used as the layout of $R = 16$. As we can see from the figure, the layout of $R = 16$ is composed by different sizes of subRAIDs. For example, the size of subRAID 0 is four and subRAID 1 is formed by three partitions. Even though the layout of $R = 16$ goes against the principle of *G* being a prime number, this hybrid subRAIDs still have the parallel reconstruction characteristic.

Considering the data layout of $R = 12$ as shown in Fig. 3, one can see that only a few subRAIDs can be built (e.g., subRAID 0, subRAID 15 etc.). But for subRAID 10 or subRAID 17, only two partitions are available with the least disk number equaling 3 for RAID5. Similar limitations are also found in the case of $R = 13$ and $R = 14$ and the alternative solution will be discussed below.

Generally, the number of disks, *R*, should meet the conditions of $G_a^2 < R \leq G_b^2$ and $R \geq 3 \times G_b$, where G_a and G_b must be prime numbers. And the final mapping table of *R* disks is the first *R* columns of the table M_b, which is generated by Equation (1). The reason why *R* must meet the criterion $R \geq 3 \times G_b$ is that each subRAID has at least 3 disks to build a standard RAID5 level. If this criterion is not met, some disk space will be

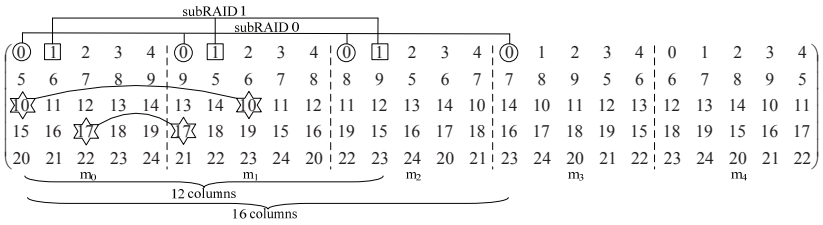

Fig. 3. Data layouts of S^2-RAID5 with $R = 16$ and $R = 12$.

wasted since some disk partitions can not be used in constructing subRAIDs.

3.2 Compromise Data Layout Strategy

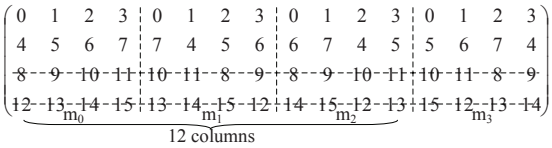

Fig. 4. Data layout of S^2-RAID5 with $R = 12$.

Of course, the above strategy is not applicable to all values of R. For instance, if R equals 12 shown in Fig. 3, even though R satisfies the constraint condition ($G_a^2 < R \leq G_b^2$), some subRAIDs may not be constructed. In order to deal with special case like this, we give the tailored layouts for 12, 13, and 14 disks using the layout of $G = 4$ in this section. The configuration process is also based on Equation (1). As seen from Fig. 4, the mapping table of $G = 4$ does not satisfy conflict-free requirements (e.g., partitions 4 and 12 in m_0 and m_2 respectively). To construct the parallel data layouts of 12-14 disks, we present a tradeoff layout based on the distribution of $G = 4$. Take the layout of $R = 12$ as an example, we give an optimized solution by erasing the third and fourth rows and select the first twelve columns to form the final distribution of $R = 12$. And then the degree of parallelism drops to 2. When the number of disks R equals to 13 or 14, accordingly the data layouts also take the mapping table shown in Fig. 4 with $\beta = 2$.

Considering the criteria of S^2-RAID5 layout, there is no reasonable solution for the layouts of 10 and 11 disks either conventional data layout in Fig. 3 or compromise way in Fig. 4 due to their low space utilization. We believe that S^2-RAID5 is not suitable to the layouts of 10 and 11 disks and using the layout of $R = 9$ is a good choice for them.

4 Evaluation Methodology

This section presents our experimental settings and methodology that we use to study quantitatively the performance of S^2-RAID. We also have implemented another baseline Parity Declustering [7], [8] prototype on Linux OS and an existing RAID architecture that share similar characteristics to S^2-RAID. The details of these two prototypes can be found in Appendix B. in Supplemental File.

4.1 Experimental Settings

The prototype S^2-RAID is installed on a storage server that is embedded in iSCSI target. Storage client is connected to the storage server using a gigabit Ethernet switch. The storage server and client have the similar settings, including operating system (Fedora 14 with kernel-2.6.35), and 8GB DDR3 memory. We use 2 HBA cards in the server to house 15 disks and the details of software and hardware in the server and client are listed in Appendix C.1. The *chunk size* and *speed_limit_min* in evaluations are set to 64KB and 8MB/s respectively.

In our experiments, all I/O requests are generated from client by replaying real world I/O traces. The trace replay tool is *btreplay* that replays traces at block level. As results of the replay, I/O requests are generated to the storage server in the form of iSCSI requests.

For performance comparison purpose, some baseline RAID layouts are built on the same testing environment:

1) RAID5: RAID5 is configured by MD with 8 data disks and one parity disk. For a fair comparison, S^2-RAID5 takes the same settings as RAID5.
2) Parity Declustering (PD for short): The second baseline is one type of 9-disk PD based on the block design shown in Appendix B.2. All settings are also the same as S^2-RAID5 and an extra mapping table is used for this layout.
3) 9-disk RAID50: The third baseline is a hybrid RAID assembled with RAID5 and RAID0 since the current MD release does not support RAID50 directly. One RAID0 is composed of three RAID5s, and each RAID5 is rebuilt by three disks. This alternative scheme maintains parallel reconstruction feature.
4) 12-15 disks RAID50: The last one is the RAID50 setting scheme for 12-15 disks. Each RAID50 is organized as follows: four RAID5s and each consists 3 or 4 disks for 12, 13, and 14 disks RAID50, respectively; five RAID5s for 15-disk RAID50, and each consists of 3 disks.

Moreover, our experiments assume one spare disk unless otherwise specified. Parallel reconstruction is realized by writing rebuilt data into the spare disk and available partitions on other data disks that are not involved in the reconstruction.

4.2 Workload Characteristics

The I/O traces used in our evaluations are obtained from the following three sources: Storage Performance

Council, Microsoft Research, and multimedia server. The details of these traces are described below:

Three traces, Financial-1 (Fin1), Financial-2 (Fin2), and Websearch2 (Web), come from SPC traces [4], [17]. Among them, Fin1 and Fin2 were collected from OLTP applications in a financial institution. Web trace was collected from a search engine thus all I/Os are read requests.

The Usr and Rsrch traces were collected from LVMs in user home directory and research projects servers in Microsoft Research over a week, and only LVM0s of Usr and Rsrch are used in our evaluations. Their characteristics and details can be found in [15].

The last one that drives our experiments is the multimedia trace (MM). We setup a trace collector in a multimedia server under temporary authorization, which provides multimedia sharing services to an entire university campus. The traced period was about 8 hours starting from 16pm GMT-8 on March 9, 2013.

TABLE 2
Traces Characteristics.

Traces	Write Ratio	Ave Req Size:KB	Total Req
Rsrch	90.68%	9.14	1, 433, 655
Fin1	76.84%	3.38	5, 334, 987
Usr	59.58%	23.21	2, 237, 889
Fin2	17.65%	2.39	3, 699, 195
Web	0%	15.07	4, 579, 809
MM	0%	64	3, 976, 436

Table 2 summaries I/O traces used in our experiments. Rsrch trace has the highest write ratio, followed by Fin1, Usr and Fin2. While the traces of Web and MM are two different types of read-intensive applications, with MM trace having the largest request size of 64KB and mostly sequential read I/Os.

5 Numerical Results and Discussions

Using our prototype implementation and the experimental settings described in the previous sections, we measure the performance of S^2-RAID as compared to other RAID layouts. In order to have enough confidence on our numerical results and eliminate any discrepancy caused by different runs, we carry out 4 independent runs for each set of results and report the average across the 4 runs. To make the large number of experiments manageable with long traces (e.g., Microsoft traces for Usr and Rsrch are 168 hours long), we speedup the replay of traces by 16× for traces Fin1, Fin2, Web, and MM; and by 32× for traces Usr and Rsrch. Furthermore, we assume the volume of each disk to be 10GB. All experimental results are recorded and analysed in the storage server.

5.1 S^2-RAID5 Performance under One Disk Failure

Our first experiment is to evaluate the reconstruction performance under one disk failure. All four RAID layouts are configured using 9 disks. We then artificially make one disk fail and activate RAID reconstruction to recover data on failed disk.

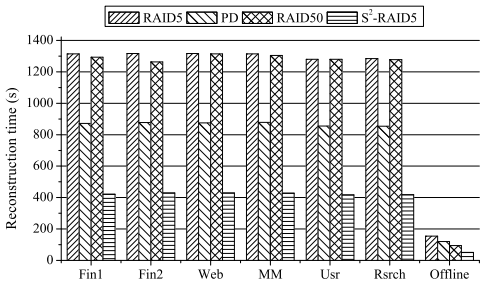

Fig. 5. Data reconstruction times of different RAID layouts for different I/O traces.

Fig. 5 shows the results of reconstruction time of four different RAID layouts for different I/O traces. As we can see from the figure, S^2-RAID5 speeds up the online reconstruction process by a factor of 3 compared to RAID5 and RAID50. Compared to PD, S^2-RAID5 doubles the speed of reconstruction. These dramatic speedups can be attributed to the conflict-free parallel reconstructions of subRAIDs. Recall Fig. 1 where subRAIDs 1, 3, and 8 can be reconstructed in parallel without conflict when physical disk D4 failed. For the traditional RAID5, on the other hand, failed disk is rebuilt only one data chunk at a time using remaining chunks of each stripe residing on good disks. Therefore, S^2-RAID5 can clearly speed up the reconstruction process. Because the stripe width of RAID50 is narrow, the reconstruction process is limited to a smaller number of disks even though this parallel processing is possible. As a result, RAID50 has a close performance to traditional RAID5.

From our experiments, we observed that PD has high rebuilding bandwidth compared to RAID5 due to its stripe width. Parallel reconstruction in PD can be used to accelerate reconstruction process between different reconstruction requests. However, there are many reconstruction request conflicts among different requests during reconstruction for the failed disk. Moreover, as seen from the layout of PD, all reconstruction requests to each physical disk read only one chunk after every three chunks, which may cause more head movements and thus give an impact on reconstruction performance. Therefore, PD takes much longer time than S^2-RAID5. We also noticed that the reconstruction time of different RAID layouts is very close under different traces. The reason is the rate-limiting mechanism in Linux software RAID. When we set the reconstruction parameter *speed_limit_min* to a fixed value, the speed of reconstruction is always maintained at this value as long as the frontend I/O request remains. This limitation is gone when we perform offline reconstruction, in which case all RAID layouts have shorter reconstruction time. And the time reductions of offline reconstruction from those of online reconstruction are 88.17%, 86.21%, 92.68% and

88.02% for RAID5, PD, RAID50, and S^2-RAID5 respectively, as shown in Fig. 5.

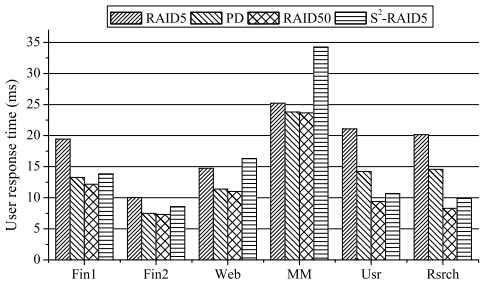

Fig. 6. Frontend production I/O response times while data reconstruction is in progress.

If data reconstruction is done online while frontend production I/Os are still being served, another interesting performance measure is the average I/O response time. Fig. 6 shows the results of frontend performance among different RAID layouts under the same set of I/O traces. It can be seen that compared to RAID50, the performance of S^2-RAID drops by approximately 19.56% on average under six traces, especially for the read-intensive traces Web and MM, up to 32.48% and 30.89% respectively. Likewise, the similar trend was also observed with PD. By contrast, S^2-RAID5 outperforms RAID5 under write-intensive traces Rsrch, Fin1, and Usr with approximate improvements being 50.77%, 28.91%, and 49.37%, respectively. However, S^2-RAID performance drops as the read/write ratio increases. Careful analysis of the RAID structures unveiled two main reasons for the write and read performance difference as summarized below.

First of all, we noticed that S^2-RAID5 shows better performance for high write ratio. The reason is the less I/O required when an I/O request is to perform a write to the *failed disk*. S^2-RAID5 only needs 1 read I/O to compute parity and 1 write I/O for parity update while traditional RAID5 needs 7 read I/Os to compute new parity and 1 write I/O for parity update. Since there are extensive small writes in the traces, S^2-RAID5 shows much better performance.

The write pattern of RAID50 is similar to that of S^2-RAID5 for one stripe, but RAID50 has a lighter reconstruction workload, compared to RAID5 and S^2-RAID5. Because the reconstruction process in RAID50 is limited at a small number of disks, 3 disks in this case. That is, only 3 disks are dedicated to the reconstruction process while the remaining 6 disks in other two RAID5 groups are mainly in charge of responding frontend I/O requests in parallel. Therefore, RAID50 shows fairly good performance under all traces.

PD also has a lighter reconstruction workload than RAID5 and S^2-RAID5. But PD causes more I/O conflicts for disks among stripes since many stripes share the same disk and the large write I/O requests (e.g., Rsrch and Usr traces) may give rise to access conflicts further. Moreover, all reconstruction requests to each disk in PD only read one chunk after every three chunks, which would cause more head movements and has an impact on frontend performance.

Secondly, S^2-RAID5 shows slight performance degradation for high read ratio. From the descriptions of S^2-RAID5, we know that there are 8 disks involved in reconstruction, with 6 disks for reading and 2 disks for saving rebuilt data. Considering another fact that the capacity of parity data in S^2-RAID5 occupies a third of whole RAID size in total. Therefore, the handling of frontend read requests may need more disk I/Os to get same amount of data compared with RAID5. For instance, if a read I/O requests 8 chunks of data, one stripe read operation completes the read I/O in a 9-disk RAID5. However, in S^2-RAID5, at least two stripe read operations are required, the first reads 6 chunks of data and the second reads the remaining 2 chunks. PD and RAID50 would behave similarly to S^2-RAID5 for such large read I/Os, although PD and RAID50 show better frontend performance than RAID5 and S^2-RAID5 due to their lighter reconstruction workloads than RAID5 and S^2-RAID5.

Additional experiments driven by standard benchmarks are shown in Appendices C.2 and C.3.

5.2 S^2-RAID5 Performance in Normal Mode

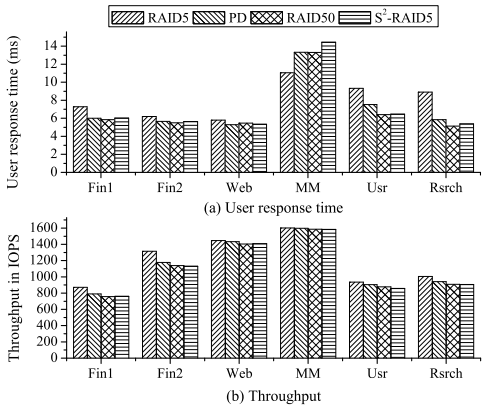

Fig. 7. Comparisons of performance in normal mode under different RAID layouts.

In order to see how S^2-RAID5 performs while no reconstruction is being done, we have also measured I/O performances of four RAID layouts under normal working conditions. The results are shown in Fig. 7 in terms of user response time and I/O throughput driven by six I/O traces. As we can see from the Fig. 7(a), traditional RAID5 shows large average user response time for the write-intensive applications due to its large stripe width. With large stripes, most of write I/Os in the traces become small writes giving rise to read-modify-write penalty and 4 disk operations for each such small write in RAID5. While the other three RAID layouts with only 3 disks in each sub-array, make most small writes in the traces be treated as read-reconstruct-write eliminating one additional disk read for parity computation.

For traces with most large sequential read I/Os, traditional RAID5 performs well because of large stripes allowing high degree of parallelism among disks in the array. In this case, S^2-RAID5 suffers from slightly larger average user response time than those of PD and RAID50 under MM trace.

The main reason is the different implementations between S^2-RAID5 and other three RAID layouts. S^2-RAID is based on an open source iSCSI target located at the top of MD, and all subRAIDs are remapped into a uniform LUN based on mapping in Fig. 2. There is a request handling mechanism in iSCSI target, that is, all requests are handled via 8 parallel threads and every thread does one request with synchronous mode, which also limit the parallelism between subRAIDs. While the other three RAID layouts are all based on MD, in which many special parallel optimizations are taken for large sequential read requests such as asynchronous requests and so on. Therefore, S^2-RAID5 suffers from a performance disadvantage compared to other RAID layouts under MM trace.

Fig. 7(b) shows the I/O performance of four RAID layouts under the open-loop mode (set the parameter "-N" *-no-stalls* in *btreplay* tool). One can see that, PD, RAID50, and S^2-RAID show the similar performance due to their similar stripe structures, with less than 4 percent difference, while RAID5 achieves higher throughput than them under all traces. Especially for the write-intensive applications, this superiority is more significant. The reason is the high ratio of write combining for the same stripe in RAID5 since small write I/Os (see characteristics of Fin1 and Fin2 in Table 2) are more likely to be.

5.3 S^2-RAID5 Expansion Capabilities

Because of slot limitation of hardware, we only choose 12, 13, 14, and 15 disks to evaluate performance of the suboptimal S^2-RAID5 under one disk failure. Their layouts adopt the principles presented in Section 3.1. And the layouts of 12, 13, and 14 disks are based on the optimized descriptions of Fig. 4, while the layout of 15 disks takes the standard mapping table in Fig. 3. By contrast, traditional RAID5 and RAID50 are our baselines due to their flexible configurations and we omit the PD for its complex layout. As shown in Fig. 8(a), S^2-RAID5 speeds up reconstruction time by a factor of 2, 2, 2, 5 for 12, 13, 14, and 15 disks respectively over RAID5 and RAID50. Recalling the parallelism degree β for the layouts in Fig. 3 and Fig. 4, the improvements of these four layouts in S^2-RAID5 already reach their theoretical values. The results also tell us a fact that the structural difference between the two data layouts may take a significant toll on performance.

Fig. 8(b) shows frontend performance of four data layouts under traces. As we have just seen, S^2-RAID5 outperforms RAID5 in term of average user response time by up to 62.87%, and all four layouts of S^2-RAID5

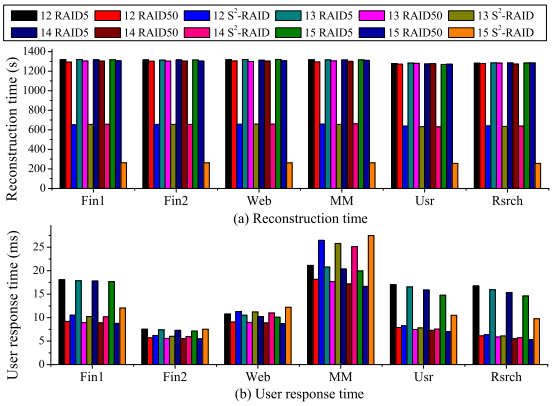

Fig. 8. Effects of number of array disks under three data layouts. Note that all RAID50 setting schemes are based on the descriptions in Section 4.1.

achieve excellent performance under most traces. But for the read-intensive applications, they have shown poor frontend performance, at an average decrease of 27.67% under MM trace. Especially for the layout of 15 disks, there is about 40% decrease. Compared to RAID50, S^2-RAID5 still shows poor frontend performance, with an average reduction of 17%, 16.9%, 16.57%, and 52.03% for 12, 13, 14, and 15 disks respectively. Besides the reason discussed in the evaluation of 9-disk RAID layouts, another more important reason is the different data layouts between 12-14 disks and 15 disks in S^2-RAID5. For instance, S^2-RAID5 with 15 disks has higher reconstruction parallelism, and all disks involve in reconstruction, while only 6-8 disks involve for 12-14 disks respectively. Thus, we observed the frontend performance differences between 15 disks and the other three layouts as shown in Fig. 8(b).

Based on the analysis of the results of Fig. 8, we have reasons to believe that the expansion schemes of S^2-RAID5 trade off frontend performance for faster reconstruction time especially under sequential read applications, in which case using standard S^2-RAID5 layout would be a better option.

5.4 S^2-RAID5 Performance under Multiple Disk Failures

To evaluate the performance of multiple disk failures, we concentrate on RAID50, and compare it with S^2-RAID5 under multiple disk failures. Fig. 9(a) shows the reconstruction time results of 1-failure, 2-failure, and 3-failure, respectively. S^2-RAID5 has shown excellent reconstruction performance over RAID50 under one or two disk failures. And the reconstruction time of S^2-RAID5 becomes longer as the number of disk failures increases. The main reason behind this phenomenon increased amount of the data to be rebuilt. But still S^2-RAID5 has shown the comparative performance under 3-failure comparing with RAID50 as shown in the figure.

As we expected, there is trade off between reconstruction time and average user response time. S^2-RAID5

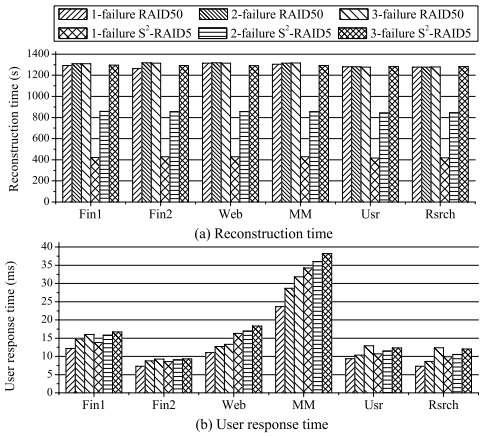

Fig. 9. Comparisons of reconstruction performance with respect to different disk failures.

and RAID50 all show performance degradation in the cases of 2-failure and 3-failure as shown in Fig. 9(b). The frontend performance of S^2-RAID5 is slightly slower under 2-failure and 3-failure, decreased by the average of 13.87% and 4.1%, respectively as compared with that of RAID50. Especially in the case of 3-failure, the decline of S^2-RAID5 is not so sharp, and the reason behind this is that the all available disks are more busy with faster reconstruction for both RAID layouts. After all, a series of experiments have demonstrated the excellent reconstruction performance of S^2-RAID5 under most applications, with slight sacrifice of frontend performance during online reconstruction.

6 Related Work

Data layout: There have been several approaches to improving RAID performance by means of data layouts [5], [8], [9], [12]. A Parity Declustering layout was proposed by Muntz and Lui [12] to utilize as few disks as possible in data reconstruction so that the rest of the disks can serve frontend requests resulting in improved reconstruction time and user response time. Holland and Gibson [7], [8] have implemented and analysed this layout, and extended it to the Panasas file system [14], [24]. Some optimizations for Parity Declustering layout [1], [2] focus on how to build data layouts that meet the six layout goals proposed by Holland and Gibson [8].

There are several differences between Parity Declustering and S^2-RAID5. Firstly, Parity Declustering needs a mapping table to handle block requests while the mapping of S^2-RAID5 is the same as traditional RAID. The S^2-RAID5 mapping structure has been established during the RAID initialization as described in section 2.2. Secondly, Parity Declustering can recover single disk failure, and multiple disk failures do not result in data loss in some cases. Finally, reconstruction threads in Parity Declustering have inevitable I/O conflicts while S^2-RAID5 is conflict-free.

There are also existing works on selecting appropriate data organizations stored in RAID systems according to workload characteristics or application scenarios. By applying different strategies that adapt to different workloads, I/O performance can be improved such as HP AutoRAID [25], Muti-tier RAID [13], Two-tiered Software Architecture [18], Muti-partition [22] and so on. Tsai and Lee [22] presented a new variation of RAID organization, Multi-partition RAID (mP-RAID), to improve storage efficiency and reduce performance degradation when disk failures occur. Their Multi-partition data layout is similar to what we referred to as subRAID. The key difference between mP-RAID and S^2-RAID is that S^2-RAID ensures that any two subRAIDs share no more than one disk. This property is important to allow conflict-free parallel reconstruction of RAID after a failure.

RAID Reconstruction: Realizing the importance of shortening the "window of vulnerability", there is a great deal of research reported in speeding up RAID reconstruction through exploiting workload characteristics [3], [10], [20], [21], [26], data or parity reorganization [11], [27] and system structures [19]. Wu et al. [26] proposed a surrogate RAID approach in their WorkOut that saves hot and popular data and rebuilds highly popular data units prior to rebuilding other units when a disk failed. The merit of this approach is that it not only reduces online reconstruction time but also the user average response time.

Similar in spirit, Menon and Mattson [11] proposed a technique called distributed sparing that makes use of parallel spare disks to improve RAID performance. The objective of this work is to utilize otherwise unused spare disks in the system. Spare disk is dispersed to the data disks of the RAID, which can achieve good access performance and also speed up data reconstruction. The difference is that S^2-RAID makes use of multiple subRAIDs to read in parallel and performs write operations to the spare disks that are not involved in reading during reconstruction.

7 Conclusions

In this paper, we have presented a new disk array architecture, S^2-RAID, for the purpose of fast data reconstruction in case of disk failures. The idea is to partition large disks into smaller logic disks to form sub arrays. The sub arrays are skewed among the physical disks in such a way that reconstruction can be done in parallel and conflict free. By making use of multiple spare disks that consist of either physical spares or available disk space on data disks, S^2-RAID substantially speeds up RAID reconstruction time. A prototype S^2-RAID has been built in the iSCSI target to measure the performance of the RAID architecture. Numerical results using real world I/O traces show that S^2-RAID improves RAID reconstruction time by a factor of 2 to 4, compared to the other three RAID layouts. The frontend I/O performance during online RAID reconstruction is comparable to the baseline of traditional RAID5. There is a good tradeoff for S^2-RAID between reconstruction and frontend

performance and the performance evaluations based on measurements have been carried out to demonstrate the superb advantage of S^2-RAID over existing RAID architectures.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable insights that have improved the quality of the paper greatly. This work is supported by the National Basic Research Program (973) of China (No. 2011CB302303), the National Natural Science Foundation of China (No. 60933002), and the NSF/CCF #1017177. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Guillermo A. Alvarez, Walter A. Burkhard, Larry J. Stockmeyer, and Flaviu Cristian. Declustered disk array architectures with optimal and near-optimal parallelism. In *Proceedings of the 25th annual international symposium on Computer architecture*, ISCA '98, pages 109–120, 1998.

[2] Siu-Cheung Chau and Ada Wai-Chee Fu. A gracefully degradable declustered RAID architecture. *Cluster Computing*, 5:97–105, January 2002.

[3] Mon-Song Chen, Dilip D. Kandlur, and Philip S. Yu. Optimization of the grouped sweeping scheduling (GSS) with heterogeneous multimedia streams. In *Proceedings of the first ACM international conference on Multimedia*, MULTIMEDIA '93, pages 235–242, 1993.

[4] Storage Performance Council. Available: http://www.storageperformance.org.

[5] Zoran Dimitrijević, Raju Rangaswami, and Edward Chang. Preemptive RAID scheduling. Technical Report TR-2004-19, University of California, Santa Barbara, 2004.

[6] Garth Gibson. Reflections on failure in post-terascale parallel computing(Keynote). In *Proceedings of the 2007 International Conference on Parallel Processing*, ICPP '07, 2007.

[7] Mark Holland. On-line data reconstruction in redundant disk arrays. *Ph.D. thesis, Carnegie Mellon University*, 1994.

[8] Mark Holland and Garth Gibson. Parity declustering for continuous operation in redundant disk arrays. In *Proceedings of the 5th international conference on Architectural support for programming languages and operating systems*, ASPLOS '92, 1992.

[9] Edward K. Lee. Software and performance issues in the implementation of a RAID. Technical Report 894351, University of California at Berkeley, 1990.

[10] Jack Y. B. Lee and John C. S. Lui. Automatic recovery from disk failure in continuous-media servers. *IEEE Transactions on Parallel and Distributed Systems*, 13(5):499–515, 2002.

[11] Jai Menon and Dick Mattson. Distributed sparing in disk arrays. In *Proceedings of the 37th international conference on COMPCON*, pages 410–421, 1992.

[12] Richard R. Muntz and John C. S. Lui. Performance analysis of disk arrays under failure. In *Proceedings of the 16th International Conference on Very Large Data Bases*, VLDB '90, 1990.

[13] Nitin Muppalaneni and K. Gopinath. A multi-tier RAID storage system with RAID1 and RAID5. In *Proceedings of the 14th International Parallel and Distributed Processing Symposium*, IPDPS '00, pages 663–671, 2000.

[14] David Nagle, Denis Serenyi, and Abbie Matthews. The panasas activescale storage cluster: Delivering scalable high bandwidth storage. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, SC '04, 2004.

[15] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write off-loading: Practical power management for enterprise storage. *ACM Transactions on Storage*, 4(3):10:1–10:23, November 2008.

[16] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, SIGMOD '88, pages 109–116, 1988.

[17] UMass Trace Repository. Available: http://traces.cs.umass.edu/index.php/storage/storage.

[18] Brandon Salmon, Eno Thereska, Craig A. N. Soules, and Gregory R. Ganger. A two-tiered software architecture for automated tuning of disk layouts. In *Proceedings of the 1st Workshop on Algorithms and Architectures for Self-Managing Systems*, pages 13–18, 2003.

[19] Muthian Sivathanu, Vijayan Prabhakaran, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Improving storage system availability with D-GRAID. In *Proceedings of the 3rd USENIX conference on File and storage technologies*, FAST '04, 2004.

[20] Lei Tian, Dan Feng, Hong Jiang, Ke Zhou, Lingfang Zeng, Jianxi Chen, Zhikun Wang, and Zhenlei Song. PRO: a popularity-based multi-threaded reconstruction optimization for RAID-structured storage systems. In *Proceedings of the 5th USENIX conference on File and Storage Technologies*, FAST '07, 2007.

[21] Lei Tian, Hong Jiang, Dan Feng, Qin Xin, and Xing Shu. Implementation and evaluation of a popularity-based reconstruction optimization algorithm in availability-oriented disk arrays. In *Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies*, MSST '07, pages 233–238, 2007.

[22] Wen-jiin Tsai and Suh-yin Lee. Multi-partition RAID: a new method for improving perform of disk arrays under failure. *The Computer Journal*, 40(1):30–42, 1997.

[23] Jiguang Wan, Jibin Wang, Qing Yang, and Changsheng Xie. S^2-raid: A new raid architecture for fast data recovery. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies*, MSST '10, pages 1–9, 2010.

[24] Brent Welch, Marc Unangst, Zainul Abbasi, Garth Gibson, Brian Mueller, Jason Small, Jim Zelenka, and Bin Zhou. Scalable performance of the panasas parallel file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST '08, 2008.

[25] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems*, 14(1):108–136, 1996.

[26] Suzhen Wu, Hong Jiang, Dan Feng, Lei Tian, and Bo Mao. WorkOut: I/O workload outsourcing for boosting RAID reconstruction performance. In *Proccedings of the 7th conference on File and storage technologies*, FAST '09, 2009.

[27] Qin Xin, Ethan L. Miller, and Thomas Schwarz. Evaluation of distributed recovery in large-scale storage systems. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, HPDC '04, 2004.

[28] Qin Xin, Ethan L. Miller, Thomas Schwarz, Darrell D. E. Long, Scott A. Brandt, and Witold Litwin. Reliability mechanisms for very large storage systems. In *Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*, MSST '03, 2003.

Jiguang Wan received the bachelor's degree in computer science from Zhengzhou University, China, in 1996, the MS and PhD degrees in computer science from Huazhong Univerisity of Science and Technology, China, in 2003 and 2007, respectively. He is currently an associate professor at Wuhan National Laboratory For Optoelectronics,Huazhong University of Science and Technology. His research interests include computer architecture, networked storage system, I/O and data storage architectures, parallel and distributed system.

Jibin Wang received the bachelor's degree in computer science from Chengdu University of Information Technology, China, in 2007, the M-S degree in computer science from Huazhong University of Science and Technology, China, in 2010. He is currently working towards the PhD degree in computer science department at Huazhong University of Science and Technology. His research interests include computer architecture networked storage system, parallel and distributed system.

Changsheng Xie received the BS and MS degrees in computer science from Huazhong University of Science and Technology, China, in 1982 and 1988, respectively. Presently, he is a professor in the Department of Computer Engineering at Huazhong University of Science and Technology (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, disk I/O system, networked data storage system, and digital media technology. He is the vice chair of the expert committee of Storage Networking Industry Association (SNIA), China.

Qing Yang received the BS degree in computer science from Huazhong University of Science and Technology, Wuhan, China, in 1982, the MS degree in electrical engineering from the University of Toronto, Canada, in 1985, and the PhD degree in computer engineering from the Center for Advanced Computer Studies, University of Louisiana, Lafayette, in 1988. Presently, he is a distinguished engineering professor in the Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, where he has been a faculty member since 1988. His research interests include computer architectures, memory systems, I/O and data storage architectures, storage networking (SAN, NAS, and LAN), parallel and distributed computing (software and hardware), embedded computer systems and applications, computer applications in biomedical systems. He is a Fellow of the IEEE and the IEEE Computer Society and a member of the ACM SIGARCH.

Supplemental File of S^2-RAID: Parallel RAID Architecture for Fast Data Recovery

Jiguang Wan, Jibin Wang, Changsheng Xie, and Qing Yang, *Fellow, IEEE*

APPENDIX A
DATA LAYOUT OF S^2-RAID

A.1 Analysis of Conflict-Free Parallelism

In this section, we show mathematically that S^2-RAID can reconstruct data on a failed disk by reading each stripe on good disks in parallel without conflict.

Let $\mu_{i,0}$, $\mu_{i,1}$, ..., and $\mu_{i,G-1}$ be column sets representing elements in the 1^{st}, 2^{nd}, ..., and G^{th} columns of the sub-matrix m_i respectively. Let U_i be a vector set representing all column vectors of m_i. That is, $U_0 = (\mu_{0,0} \quad \mu_{0,1} \quad \mu_{0,G-1})$, $U_1 = (\mu_{1,0} \quad \mu_{1,1} \quad \mu_{1,G-1})$, ..., and $U_{G-1} = (\mu_{G-1,0} \quad \mu_{G-1,1} \quad \mu_{G-1,G-1})$. Clearly, to realize the conflict-free parallel reconstruction the following two conditions must be satisfied simultaneously,

$$\begin{cases} \mu_{i,j} \cap \mu_{j,i} = \{e\}, \mu_{i,j} \in U_i, \mu_{j,i} \in U_j \ and \ i \neq j \\ \mu_{i,j} \cap \mu_{i,k} = \varnothing, \mu_{i,j}, \mu_{i,k} \in U_i \ and \ j \neq k \end{cases},$$

where $0 \leq i, j, k \leq G-1$ and {e} is an identity set with non-null element. In other words, there is only one common element when intersection column sets from different groups and no intersection among column sets in a group. This is the key to the S^2-RAID layout.

Now consider column vectors of matrix matrix m_i and m_j. We use $C_{i,p}$ and $C_{j,q}$ to represent $(p+1)^{th}$ column vector of the sub-matrix m_i and the $(q+1)^{th}$ column vector of the sub-matrix m_j respectively. Suppose the vectors $C_{i,p}$ and $C_{j,q}$ are known and given by

$$C_{i,p} = \begin{pmatrix} a_{0,p} \\ a_{1,p} \\ a_{2,p} \\ \cdots \\ a_{K-1,p} \end{pmatrix} and \ C_{j,q} = \begin{pmatrix} b_{0,q} \\ b_{1,q} \\ b_{2,q} \\ \cdots \\ b_{K-1,q} \end{pmatrix},$$

where $a_{i,j}$ and $b_{i,j}$ denote elements at $(j+1)^{th}$ column $(i+1)^{th}$ row of the $C_{i,p}$ and $C_{j,q}$, respectively. From Equation (1) we know that the elements of the $C_{j,q}$ can be obtained from sub-matrix m_i though cyclic shift operations, we have

$$\begin{cases} b_{0,q} = a_{0,q \bmod G} \\ b_{1,q} = a_{1,[q+(j-i)] \bmod G} \\ b_{2,q} = a_{2,[q+2\times(j-i)] \bmod G} \\ \quad \cdots \\ b_{K-1,q} = a_{K-1,[q+(K-1)\times(j-i)] \bmod G} \end{cases},$$

then $C_{j,q}$ can be represented by

$$C_{j,q} = \begin{pmatrix} b_{0,q} \\ b_{1,q} \\ b_{2,q} \\ \cdots \\ b_{K-1,q} \end{pmatrix} = \begin{pmatrix} a_{0,q \bmod G} \\ a_{1,[q+(j-i)] \bmod G} \\ a_{2,[q+2\times(j-i)] \bmod G} \\ \cdots \\ a_{K-1,[q+(K-1)\times(j-i)] \bmod G} \end{pmatrix},$$

$1 \leq K \leq G$. What we want to show next is that the column subscripts of $a_{0,q \bmod G}$, $a_{1,[q+(j-i) \bmod G]}$, $a_{2,[q+2\times(j-i)] \bmod G}$, ..., and $a_{K-1,[q+(K-1)\times(j-i)] \bmod G}$ are different from each other.

Suppose that there were two identical subscripts on row n1 and n2 of that column, the following equation would hold

$$[q + n1 \times (j-i)] \bmod G = [q + n2 \times (j-i)] \bmod G,$$

which implies

$$[(n1 - n2) \times (j-i)] \bmod G = 0. \quad (2)$$

However, there are some limitations as follows:

$$\begin{cases} 0<n1, n2, i, j<G \\ n1 \neq n2, i \neq j \\ G \ is \ a \ prime \ number \end{cases},$$

which mean the true of the following inequalities

$$\begin{cases} (n1 - n2) \neq 1, (j-i) \neq 1 \\ (n1 - n2)<G, (j-i)<G \end{cases}.$$

Finally, we can deduce the conclusion, $(n1 - n2) \times (j - i) \ mod \ G \neq 0$. Then, Equation (2) can never hold. Therefore, their column subscripts are all different from each other indicating that they span all columns of matrix m_i and there is one and only one of the same element in any column of m_i. Consequently, the intersection between $C_{i,p}$ and $C_{j,q}$ is always one. That is, the system can carry out conflict-free parallel reconstruction.

A.2 Design of S^2-RAID10

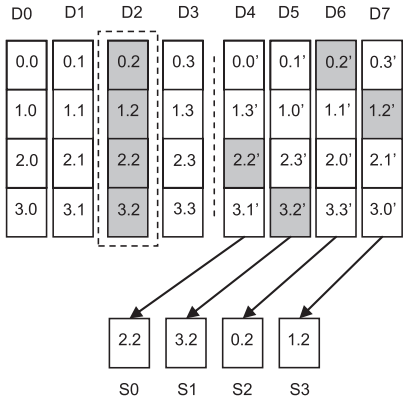

Fig. 10. An overview of S^2-RAID10 data layout.

Fig. 10 shows one example of S^2-RAID10 design with 8 disks and subRAID size of 8. Disks D0, D1, D2, and D3 are data disks and D4, D5, D6, and D7 are mirror disks, respectively. Instead of directly mirroring data stripes in traditional RAID10, S^2-RAID10 skews data partitions in mirror disks by shifting each subsequent partition by one position as shown in the figure. In this way, data reconstruction can be done in parallel in case of a disk failure. As an example, if disk D2 fails in shaded area as shown in Fig. 10 and the data partitions in D2 had been mirrored from D4 to D7 (see the partitions 0.2′, 1.2′, 2.2′, and 3.2′), then they can be read out in parallel. In other words, the data reconstruction can be done 4 ($\beta = 4$) times faster than the original RAID10. The data read from the 4 mirror disks then can be written in parallel to spare disks. Note that the 4 spare disks labeled S0 through S3 shown in Fig. 10 can be either physical spare disks if they are available or available disk space on the 8 data/mirror disks, as the similar descriptions on S^2-RAID5. Of course, we also give the general layout of S^2-RAID10 in the next section.

A.3 S^2-RAID10 Expansion Strategy

Similar to S^2-RAID5, S^2-RAID10 is also easily expandable. The condition for generalizing S^2-RAID10 is that disk number R must be an even number greater than 4. We separate R disks into two types of groups evenly, primary and secondary, and the notation N is a constant ($N = 2$). We then divide every disk into $K(1<K \leq G)$ partitions and select one partition from each disk to form primary and secondary groups to form subRAIDs. The subRAID size G is equal to $R/2$, and the degree of parallelism, β, is equivalent to the number of disk partitions at the best ($\beta \leq K$).

The compositional procedure of S^2-RAID10 is shown below. First, we give the initial matrix table $m_{primary}$,

$$m_{primary} = \begin{pmatrix} P_{0,0} \\ P_{0,1} \\ \cdots \\ P_{0,K-1} \end{pmatrix},$$

where $P_{0,j}$ $(0 \leq j \leq K-1)$ is a G elements vector representing subRAIDs of partition j on disks of group 0 (also called primary group). Then the layout matrix of secondary group can be created by shift operations,

$$m_{secondary} = \begin{pmatrix} P'_{0,0} \\ P'_{0,1} \\ \cdots \\ P'_{0,K-1} \end{pmatrix} = \begin{pmatrix} SH_r^0(P_{0,0}) \\ SH_r^1(P_{0,1}) \\ \cdots \\ SH_r^{K-1}(P_{0,K-1}) \end{pmatrix}, \quad (3)$$

where $P'_{0,j}$ denotes the $(j+1)^{th}$ line of the matrix table in secondary group, equaling to the result of $SH_r^b(P_{0,j})$. The function of $SH_r^b(P_{0,j})$ is similar to the Equation (1).

Given the condition $R = 8$, we can obtain the following initial layout matrix $m_{primary}$,

$$m_{primary} = \begin{pmatrix} 0.0 & 0.1 & 0.2 & 0.3 \\ 1.0 & 1.1 & 1.2 & 1.3 \\ 2.0 & 2.1 & 2.2 & 2.3 \\ 3.0 & 3.1 & 3.2 & 3.3 \end{pmatrix}.$$

Recall that notation $S.L$ represents logic disk number L in subRAID number S. The identifier 0.0, for example, represents the first logic partition of the subRAID 0. Then the final mapping table M with $R = 8$ based on Equation (3) is:

$$M = \begin{pmatrix} 0.0 & 0.1 & 0.2 & 0.3 & 0.0' & 0.1' & 0.2' & 0.3' \\ 1.0 & 1.1 & 1.2 & 1.3 & 1.3' & 1.0' & 1.1' & 1.2' \\ 2.0 & 2.1 & 2.2 & 2.3 & 2.2' & 2.3' & 2.0' & 2.1' \\ 3.0 & 3.1 & 3.2 & 3.3 & 3.1' & 3.2' & 3.3' & 3.0' \end{pmatrix},$$

where notation $S.L'$ is an image of $S.L$. Obviously, this mapping table can achieve parallel reconstruction at 4 times speedup without access conflicts.

APPENDIX B
PROTOTYPE IMPLEMENTATION

For the purpose of proof-of-concept and performance evaluation, we have built two prototypes of S^2-RAID and Parity Declustering RAID in the same environment. Our implementations are based on the source code of the Linux software RAID (MD) and the following two sections give the details of them.

B.1 Software Structure of S^2-RAID

S^2-RAID was implemented in the Linux operating system inside the kernel and embedded into the iSCSI target [8]. In the iSCSI target below, we realized the S^2-RAID functionalities. Fig. 11 shows the software structure of

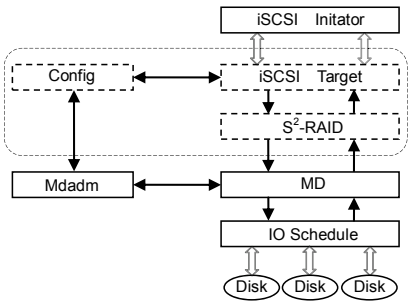

Fig. 11. Software structure of S^2-RAID implementation.

our prototype. It mainly includes three modules, iSCSI target module, S^2-RAID function module and configuration module.

The iSCSI target module modifies the SCSI command handling and disk I/O process. The disk I/Os of the iSCSI target call upon interfaces of the S^2-RAID module.

The S^2-RAID module implements the basic functions of S^2-RAID5 and S^2-RAID10 including RAID rebuilder based on MD. MD itself provides RAID rebuilder that allows parallel reconstruction of multiple RAIDs for a disk failure provided that these RAIDs do not share physical disks. When multiple RAIDs share physical disks, MD's rebuilder reconstructs data sequentially with no parallelism.

Considering the mapping of S^2-RAID, we employ two small mapping tables to manage data layouts as shown in Fig. 1 (see the figure in the paper) and Fig. 10. And the mapping mode within each subRAID is the same as traditional MD. In this way, upon a disk failure, we are able to use MD's rebuilder to reconstruct data in parallel. In addition, some extra information should be recorded including spare disk locations. While the addressing of S^2-RAID can be done by computation as described in Fig. 2 in the paper.

For S^2-RAID10 shown in Fig. 10, we built 16 subRAIDs with RAID1 level using 2 disk partitions, then the S^2-RAID module forms a uniform addressing space with RAID0 level, as called S^2-RAID10. From point of view of RAID reconstruction, there are 4 independent conventional RAID1s without sharing physical disks allowing parallel reconstruction in case of one disk failure.

The S^2-RAID module finally maps the multiple subRAIDs to one unified LUN. This LUN presents to the iSCSI target module as one logical disk for read and write I/Os.

The configuration module provides RAID setup and configuration functions using mdadm commands to realize different S^2-RAID functions. It also allows users to configure iSCSI target by means of iSCSI configuration functions.

B.2 Implementation of Parity Declustering

This section gives the implementation details of Parity Declustering. Parity Declustering is one of the existing works that is closely related to S^2-RAID5. In order to show the difference both of them, we have implemented Parity Declustering based on MD. Our implementation is based on a data construction algorithm, which is derived from a balanced incomplete block design [3].

Through the concept we know, a block design table is an arrangement of ν distinct objects into b tuples, each of which contains k objects, such that each object appears in exactly γ tuples, and any pair of objects in tuples just appears λ tuples. There are also two notations C and G, where C is the number of disks in the RAID and G is the span of the stripe units with which parity unit can protect some smaller number of data units instead of $C-1$ and $C=\nu, G=k$. We have implemented a 9-disk layout of Parity Declustering, with the stripe width of 3. The detail of distributing parameters are $\nu=9$, $b=12$, $k=3$, and $\gamma=4$ as shown in Table 3.

TABLE 3
A 9-Disk Parity Declustering Layout Based on Balanced Incomplete Block Design.

Stripe Number	Tuple	Stripe Number	Tuple
0	0, 1, 2	6	0, 4, 8
1	3, 4, 5	7	1, 5, 6
2	6, 7, 8	8	2, 3, 7
3	0, 3, 6	9	0, 5, 7
4	1, 4, 7	10	1, 3, 8
5	2, 5, 8	11	2, 4, 6

APPENDIX C
ADDITIONAL EVALUATION OF S^2-RAID

C.1 Details of Testbed Settings

In this section, we introduce the details of software and hardware configurations used in our evaluations as listed in Tables 4 and 5.

TABLE 4
Hardware Details of The Storage Client and Server. Note that all hardwares in either client or server are the same except for the disk and HBA.

OS	Fedora 14(kernel-2.6.35)
Disks (server)	1 Seagate ST3320418AS, 320GB, 7200RPM 15 Seagate ST3500410AS, 500GB, 7200RPM
Disk (client)	1 Seagate ST3320418AS, 320GB, 7200RPM
Mainboard	Supermicro X8DT6
CPU	Intel(R) Xeon(R) CPU 5650 @2.67GHz
NIC	Intel 82574L
Memory	8GB DDR3
HBA (server)	Symbios Logic SAS2008

C.2 Benchmark Settings

Besides the traces, we also choose a set of standard benchmarks as test cases that are widely used in the research community and industry.

The first benchmark we selected is PostMark [4] that is widely used as file system benchmark tool written

TABLE 5
Parameters Used in Evaluation and Software Settings of The Storage Server and Client.

Application softwares	
blktrace	blktrace 1.0
iSCSI Target	iSCSI target-1.4.20.2
iSCSI initator	iSCSI-initiator-util 6.2.0865
PostgreSql	PostgreSQL-8.1.15
MySQL	MySQL 5.1.40
TPC-C	TPCC-UVA-1.2.3
PostMark	PostMark-1.5
Sysbench	Sysbench-0.4.12
Important parameters	
chunk size	64KB (default)
speed_limit_min	8MB/s

by NetApp, Inc. In our experiments, we set PostMark workload to include 100, 000 files of size 4KB to 500KB and to perform 200, 000 transactions. Read and write block sizes are set to 4KB.

Sysbench is a system evaluation benchmark designed for identifying system parameters that are important for a system running a database under intensive load [5]. Sysbench runs at file-IO test mode with 20 threads and 20GB file size. The file test mode is sequential reading.

TPC-C is a well-known benchmark used to model the operational end of businesses where real-time transactions are processed. We set up the PostgreSQL database based on the implementation from TPCC-UVA [6]. 30 warehouses with 10 terminals per warehouse are built on PostgreSQL database with measurement interval of 120 minutes and 20 minutes ramp-up time. Details regarding TPC-C workloads specification can be found in [2].

C.3 Additional Results Driven by Benchmarks

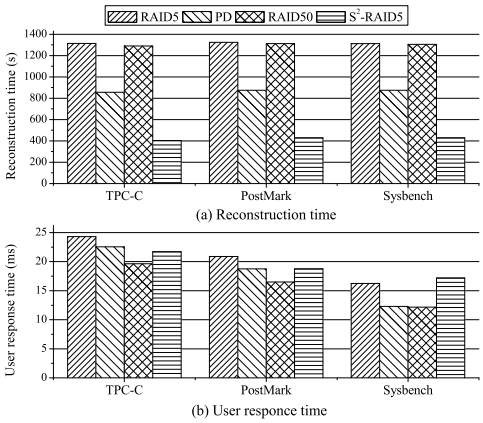

Fig. 12. Comparisons of reconstruction performance driven by three benchmarks.

In addition to trace-driven experiments, experiments driven by three standard benchmarks have been carried out with the same RAID settings described above. The results are plotted in Fig. 12. One can see from Fig. 12(a) that, S^2-RAID5 outperforms other RAID layouts significantly in terms of reconstruction time for all three benchmarks. We have observed over a factor of two to three performance improvements over RAID5 and RAID50 for all three benchmarks. Even compared with PD, dramatic performance improvement was also observed in terms of reconstruction time.

For the frontend performance, we observed that S^2-RAID5 shows comparable performance as RAID5 and PD for benchmark TPC-C and Postmark even though the reconstruction time of S^2-RAID5 is substantially better than their counter parts. This fact is shown in Fig. 12(b). As we all know, TPCC-UVA is open source implementation of the standard TPC-C benchmark. TPCC-UVA has the read to write ratio of 1:1 [7], which is different form standard TPC-C that has read to write ratio of 1.9:1 [1]. And this read to write ratio of TPCC-UVA is similar to the Usr trace. As a result, its performance shown in Fig. 12(b) is similar to that shown in Fig. 8. For PostMark, we also observed a 1:1 read to write ratio in our experiments. The performance difference among the four RAID layouts is not that significant similar to TPC-C.

Different from TPC-C and PostMark, we use the Sysbench benchmark under sequential read access mode to evaluate the frontend performance once again. The results in Fig. 12(b) also support previous statements (as explained in section 5.1) of poor sequential read performance for S^2-RAID5 due to the heavy reconstruction workloads of S^2-RAID5 comparing with other three RAID layouts.

C.4 S^2-RAID10 Performance

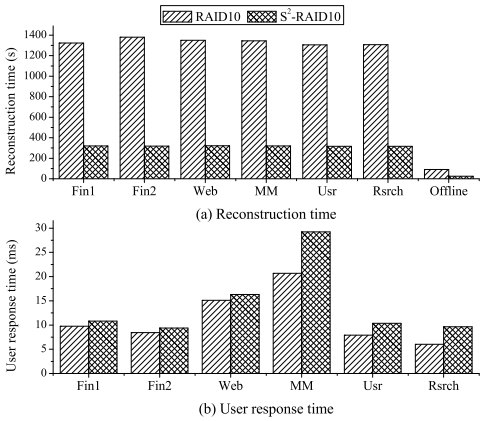

Fig. 13. Comparisons of reconstruction performance for RAID10 and S^2-RAID10 under traces.

To evaluate the reconstruction performance of RAID10, we conduct experiments for the two RAID layouts, RAID10 and S^2-RAID10, with 8 disks. As shown in Fig. 13(a), S^2-RAID10 outperforms RAID10 by a factor of 4.13 on average, with the parallelism degree β equaling 4. Most performance gains of S^2-RAID10 come from the disperse disk partitions in secondary group, and four parallel reconstruction threads can migrate tiny patches to their reserved disks instead of

the whole disk. There are 8 disks in total (including 1 spare disk) involved in reconstruction process for 8-disk S^2-RAID10, with 3 disks and 1 spare disk in primary group playing the role of spare disks and 4 disks in secondary group for reconstruction. While traditional RAID10 is based on a one-to-one reconstruction mode, with only two disks taking reconstruction by data migration.

Better disk utilizations and high degree of parallelism in S^2-RAID10 during reconstruction process will also adversely affect frontend I/O performance while reconstruction is going on. The results of frontend performance while reconstruction is in progress in background are shown in Fig. 13(b). S^2-RAID10 has an average 27.13% performance degradation in terms of user response time compared with RAID10. For multimedia applications such as MM, this performance degradation is more pronounced. The fact is that all disks in S^2-RAID10 are used for either reconstruction or responding frontend I/O requests. It is understandable, why frontend performance suffers a little bit - the heavy workloads of each disk.

REFERENCES

[1] Shimin Chen, Anastasia Ailamaki, Manos Athanassoulis, Phillip B. Gibbons, Ryan Johnson, Ippokratis Pandis, and Radu Stoica. TPC-E vs. TPC-C: characterizing the new TPC-E benchmark via an I/O comparison study. *SIGMOD Record*, 39(3):5–10, February 2011.

[2] Transaction Processing Performance Council. TPC benchmark^{tm} C standard specification, 2005.

[3] Marshall Hall. *Combinatorial Theory (2nd Edition)*. Wiley Interscience, 1986.

[4] Jeffrey Katcher. Postmark: a new file system benchmark. Technical Report TR3022, Network Appliances, 1997.

[5] Alexey Kopytov. SysBench, a system performance benchmark, Available: http://sysbench.sourceforge.net/index.html, 2004.

[6] Diego R. Llanos. TPCC-UVa: an open-source TPC-C implementation for global performance measurement of computer systems. *SIGMOD Record*, 35(4):6–15, December 2006.

[7] Jin Ren and Qing Yang. A new buffer cache design exploiting both temporal and content localities. In *Proceedings of the 30th IEEE International Conference on Distributed Computing Systems*, ICDCS '10, pages 273–282, 2010.

[8] iSCSI Enterprise Target. Available: http://sourceforge.net/projects/iscsitarget/files/, 2012.