University of Nebraska - Lincoln

DigitalCommons@University of Nebraska - Lincoln

CSE Conference and Workshop Papers

Computer Science and Engineering, Department of

2009

JOR: A Journal-guided Reconstruction Optimization for RAID-Structured Storage Systems

Suzhen Wu
Wuhan National Laboratory for Optoelectronics, suzhen66@gmail.com

Dan Feng
Huazhong University of Science & Technology, Wuhan, China, dfeng@hust.edu.cn

Hong Jiang
University of Nebraska-Lincoln, jiang@cse.unl.edu

Bo Mao
Wuhan National Laboratory for Optoelectronics,, maobo@gmail.com

Lingfang Zeng
Huazhong University of Science & Technology, Wuhan, China, lfzeng@hust.edu.cn

See next page for additional authors

Wu, Suzhen; Feng, Dan; Jiang, Hong; Mao, Bo; Zeng, Lingfang; and Chen, Jianxi, "JOR: A Journal-guided Reconstruction Optimization for RAID-Structured Storage Systems" (2009). *CSE Conference and Workshop Papers*. 20.
https://digitalcommons.unl.edu/cseconfwork/20

2009 15th International Conference on Parallel and Distributed Systems

JOR: A Journal-guided Reconstruction Optimization for RAID-Structured Storage Systems

Suzhen Wu*, Dan Feng*✉, Hong Jiang†, Bo Mao*, Lingfang Zeng* and Jianxi Chen*
*Wuhan National Laboratory for Optoelectronics, China
School of Computer Science & Technology, Huazhong University of Science & Technology, Wuhan, China
Email: {suzhen66, maobo.hust}@gmail.com, {dfeng, lfzeng, chenjx}@hust.edu.cn
†Department of Computer Science & Engineering, University of Nebraska-Lincoln, Lincoln, USA
Email: jiang@cse.unl.edu

***Abstract*—This paper proposes a simple and practical RAID reconstruction optimization scheme, called JOurnal-guided Reconstruction (*JOR*). JOR exploits the fact that significant portions of data blocks in typical disk arrays are unused. JOR monitors the storage space utilization status at the block level to guide the reconstruction process so that only failed data on the used stripes is recovered to the spare disk. In JOR, data consistency is ensured by the requirement that all blocks in a disk array be initialized to zero (written with value zero) during synchronization while all blocks in the spare disk also be initialized to zero in the background.**

JOR can be easily incorporated into any existing reconstruction approach to optimize it, because the former is independent of and orthogonal to the latter. Experimental results obtained from our JOR prototype implementation demonstrate that JOR reduces reconstruction times of two state-of-the-art reconstruction schemes by an amount that is approximately proportional to the percentage of unused storage space while ensuring data consistency.

***Keywords*-RAID; storage system; reliability; reconstruction; performance evaluation;**

I. Introduction

RAID provides improved levels of performance, availability and reliability for storage systems over a single-disk system [1]. When a disk fails, RAID with a hot-spare disk will automatically switch to the *reconstruction mode* at a degraded performance. The reconstruction process reads all data blocks from surviving disks, rebuilds the data blocks on a failed disk, and then writes the rebuilt data blocks onto the spare disk. After recovering all failed data blocks, RAID returns to the *normal mode*.

Recovering from disk failures is crucial to applications that require both high performance and high reliability from their storage subsystems [2]. Such systems demand not only the ability to recover from a disk failure without losing data, but also a reconstruction process that (1) minimizes the reconstruction time, and (2) has minimal impact on user and system performance, simultaneously.

As a result of the exponential growth in individual disk capacity and system scale coupled with a stagnant improvement in individual disk failure rate, reconstruction in such systems may become the common mode rather than the exception [3], [4]. To make things worse, the improvement in seek and rotation time has been much slower than the advance in individual disk capacity, thus significantly increasing the length of the reconstruction process. Moreover, latent sector errors [5] during reconstruction further increase the probability of data loss. Thus, the RAID reconstruction performance can have a serious impact on the reliability and availability of RAID-structured storage systems.

Studies on RAID reconstruction have been concentrating on how to improve the performance of RAID reconstruction algorithms. To regenerate each lost block on the failed disk, most existing reconstruction algorithms [2], [6]–[11] read all remaining blocks in its stripe group in all surviving disks, calculate to regenerate the block, and then write the newly rebuilt block onto the spare disk.

In contrast, Sivathanu *et al.* [12] proposed a live-block recovery method that recovers only the live data to the spare disk to speedup the reconstruction process. However, live-block recovery must rely on file system's semantic knowledge to determine which blocks are live and should be recovered, which thus limits its applicability since it remains largely impractical to obtain file semantic information at the block level for general file systems. Furthermore, live-block recovery is effective only for replication-based disk arrays (*e.g.*, RAID1 and RAID10), but not applicable for parity-based disk arrays (*e.g.*, RAID5 and RAID6) due to possible data inconsistency and data loss (see Section II-B for detail).

Inspired by live-block recovery [12], we propose a *JOR* (*JO*urnal-guided *R*econstruction) optimization scheme to improve RAID reconstruction performance. The main idea behind JOR is to reconstruct the blocks in used stripes to their corresponding locations on the spare disk and ignore the unused stripes, by monitoring which stripes have been used and which stripes remain unused at the block level. JOR has the potential to recover the RAID-structured storage system more quickly than existing approaches and is applicable for all RAID levels. Importantly, JOR ensures data consistency constantly and does not incur data loss during reconstruction. More significantly, JOR is orthogonal to and can be easily incorporated into most existing RAID

1521-9097/09 $26.00 © 2009 IEEE
DOI 10.1109/ICPADS.2009.69

reconstruction approaches without modifying the file system or operating system.

We implement a JOR prototype by incorporating the JOR optimization into two state-of-the-art reconstruction approaches, PR [8] and PRO [9], [13], in the Linux software RAID (MD). The off-line and trace-driven experiments show that JOR-enhanced PR and PRO outperform their unenhanced counterparts in reconstruction time consistently by an amount that is proportional to the percentage of unused storage space. Accordingly, JOR reduces the period of performance degradation.

The rest of this paper is organized as follows. Background and motivation are presented in Section II. In Section III we describe the design and implementation of the JOR optimization. Performance evaluation and discussion are presented in Section IV. Related work is presented in Section V. We conclude this paper by summarizing the main contribution of this paper in Section VI.

II. Background and Motivation

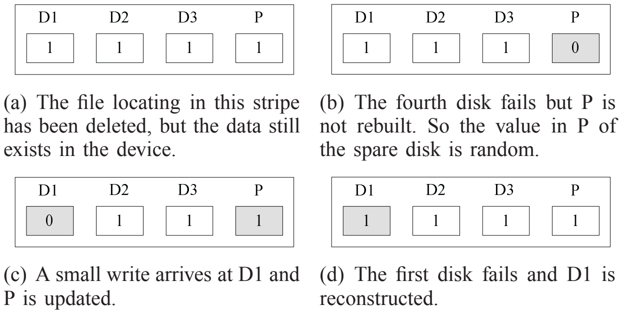
A. Unused storage space

In the real world, storage systems are deployed for long-time runs so that its capacity is always outgrowing the demand [14]. In addition, recent studies indicate that the annualized failure rates of disks are generally higher than those reported by manufactures [3], [4]. To cope with the increased disk failure rates, data centers that employ large-scale disk-based storage systems routinely deploy sufficient amount of spare disk capacity for the purpose of data reconstruction and/or absorbing peak performance demands. Thus, at the time of data reconstruction for a failed disk, there is usually a great deal of unused storage space.

Furthermore, Gray [15] reported that disks at Microsoft are only 30% full on average. Agrawal *et al.* [16] found that the median fullness of file system has decreased from 47% to 42% during the five-year study of the file system metadata. Through the 45 days of traces collected from servers, Hsu & Smith [17] found that only less than 50% of the disk space has been accessed.

It must be noted that *unused* blocks at the block level are different from *free* blocks at the file level, because the file-system deletion operation is likely to result in some used blocks to be free blocks. However, studies show that the deletions can be negligible for most file systems. Based on 36 sets of traces collected from PCs running the Microsoft Windows95 operating system, Zhou & Smith [18] concluded that the "DELETE" operation only accounts for 0.16% of the total file system calls. Recently, Traeger *et al.* [19] showed that the number of delete-inode operations is quite small during the build process.

Studies [18], [19] also show that most dead blocks, except for the unallocated blocks, are produced by being overwritten at the file level rather than being deleted or truncated. Based on a write-back cache policy, delayed writes allow programs to continue executing after writing data to the cache but not the disks.Thus, temporary files that exist for less than 30 seconds are often deleted from the cache before they must be written to the disks [20]. Furthermore, if the storage system exploits NVRAM to buffer write data, multiple overwrites can be absorbed into a single write to the hard disk [21]. In conclusion, overwrites are shown not to impact disks at the block level for the most part, especially for the cached storage systems.



(a) The file locating in this stripe has been deleted, but the data still exists in the device.

(b) The fourth disk fails but P is not rebuilt. So the value in P of the spare disk is random.

(c) A small write arrives at D1 and P is updated.

(d) The first disk fails and D1 is reconstructed.

Figure 1. An example of data loss caused by live-block recovery for parity-based RAID.

Consequently, researchers are increasingly looking for creative ways to effectively utilize the unused storage space. For example, the research conducted at Princeton explores ways to trade off capacity for performance [22]. FS ^{2} uses the free disk space to dynamically change the disk layout, thus increasing disk I/O performance [23]. PARAID [14], write off-loading [24] and Everest [25] exploit the unused storage space at the block level to store the migrated data, thus improving energy efficiency or peak performance. Therefore, we can confidently conclude that *there is significant unused capacity at the block level*, especially in the Content Addressable Storage (CAS) [26], in which the fixed contents are written once and never changed.

B. Reconstruction problem with live-block recovery in parity-based RAID

As mentioned in Section I, live-block recovery [12] is effective for replication-based RAID, but not applicable for parity-based RAID. The reason is that the stripes across all the disks that cover the un-recovered blocks will lose their data consistency during recovery. If a subsequent disk fails after completing the previous reconstruction process, it is likely to be unable to recover the failed data. Fig. 1 gives an example to illustrate the scenario in detail.

(1) Suppose that a RAID5 disk array consists of four disks. Fig. 1(a) shows the data layout of one stripe in the disk array before a disk failure. D1, D2 and D3 denote the data blocks while P denotes the parity block. "1" indicates that the data in the block is all "1" while "0" indicates that the data in the block is all "0". When the data in the stripe has been deleted in the file system, the data blocks and parity block ("1")

still remain in the disk drives. *Note: D1 xor D2 xor D3 = P.*

(2) The fourth disk fails. In this stripe, the failed block is the parity block. If the deleted stripe is detected and identified as dead, the parity block will not be recovered to the spare disk. The value in the P location of the spare disk will be random, *i.e.*, "0" or "1". In order to describe the problem clearly, we assume it to be "0", as shown in Fig. 1(b). *Note: D1 xor D2 xor D3 ≠ P.*

(3) After the reconstruction process completes, a small write arrives at the D1 location. Suppose that the write data is "0". The parity should be recalculated and written to the P location. *New P = new D1("0") xor old D1("1") xor old P("0") = "1".* Currently, D1 is a live block and the exact data value is "0", as shown in Fig. 1(c). *Note: D1 xor D2 xor D3 ≠ P.*

(4) Unfortunately, the first disk fails and a new spare disk is added to reconstruct the data of the failed disk. Now, since the stripe contains a live block, the stripe should be recovered, as shown in Fig. 1(d). *D1 = D2("1") xor D3("1") xor P("1") = "1". Note: D1 is incorrect now and we will be unable to obtain the correct data beyond this point.*

From the above example, we can see that if the dead blocks at the file level are not recovered, data loss is likely to occur, implying that live-block recovery is not feasible for parity-based RAID. Due to this shortcoming of live-block recovery, we propose a simple and practical block-level JOR optimization scheme that speeds up the reconstruction process and avoids data loss simultaneously.

III. Journal-guided Reconstruction(JOR)

In this section, we first define what "used" and "unused" blocks are at the block level and describe the system requirement. Then we present a description of the data structure and algorithm of the JOR optimization. The data consistency issue and the prototype implementation are discussed at the end of this section.

A. Definition of used and unused blocks

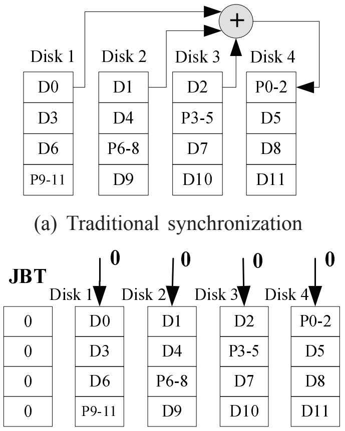
Generally, each stripe in a disk array is in one of the following three states:

(1) *Allocated*, it is currently allocated by the higher-level system (*e.g.*, file system or database system) to store data;

(2) *Released*, it was previously allocated but subsequently released as the data was deleted or moved elsewhere;

(3) *Unallocated*, it has never been allocated.

Sivathanu *et al.* [21] concluded that the block liveness information, which describes whether a block holds valid data currently, cannot be tracked accurately in traditional storage systems since the storage system is unaware of which blocks are thought to be live by the file system. In this paper, we adopt a weaker form of this liveness. That is, we refer to the state *Allocated* as *used* and the state *Unallocated* as *unused*. Also, we refer to the state *Released* as *used*, implying that failed blocks in this state should be rebuilt to ensure data consistency in JOR, for reasons shown in Section II-B. Fortunately, Section II-A demonstrates that the Released blocks accounts for only a small part.



(a) Traditional synchronization

(b) Modified synchronization

Figure 2. Traditional and modified synchronization process.

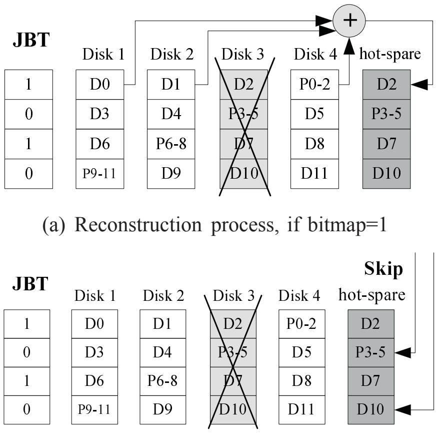
B. Modified synchronization for JOR

The goal of synchronization, when creating and initializing a disk array, is to ensure data consistency at the block level. Based on the traditional synchronization approaches, as shown in Fig. 2(a), the reconstruction process must be applied to all data blocks on the surviving disks, which is not applicable for JOR. Thus, we modify the synchronization process. For any RAID level, all blocks in a disk array are initialized to zero (written with value zero) during synchronization, as shown in Fig. 2(b). Noticeably, the overhead of the modified synchronization process is comparable with or even better than that of the traditional counterpart.

In addition, all blocks in the spare disk are also initialized to zero in the background prior to reconstruction. Thus, data consistency is ensured in JOR that reconstructs only the used stripes, since all blocks in the unused stripes in the surviving disks and the spare disk are all zero blocks.

C. Data structure and algorithm

JOR relies on one key data structure, namely, *JBT* (Journal Bitmap Table). JBT stores the storage space utilization status at the block level. A journal bit per stripe (short for *bitmap*) is used to indicate whether the stripe has been used or not. "1" indicates that the corresponding stripe has been used while "0" indicates that the corresponding stripe remains unused. When a disk array is created, all bits of JBT are initialized to "0", indicating that all stripes in this disk array are unused. When a stripe receives a write request, the corresponding bitmap is set to be "1", indicating that the stripe is now used.



(a) Reconstruction process, if bitmap=1

(b) Reconstruction process, if bitmap=0

Figure 3. The JOR process for RAID5.

Fig. 3 illustrates the JOR-enhanced RAID5 reconstruction process. In the reconstruction thread of JOR, each stripe is first checked for its utilization status in *JBT*. If the stripe is used (*i.e.*, bitmap=1), all surviving blocks in the stripe are read from surviving disks to rebuild the failed block to the spare disk, as shown in Fig. 3(a). Otherwise (*i.e.*, bitmap=0), JOR skips the unused stripe, as shown in Fig. 3(b). For other RAID levels, unused stripes are skipped by the JOR reconstruction process in exactly the same way as in RAID5.

Since all blocks in unused stripes are zero blocks, for each write request to an unused stripe during reconstruction, the new parity is simply the parity-calculation of the newly written data blocks in this stripe. Thus, each write request to an unused stripe only entails writing the new data and new parity to the disks, as opposed to reading the old data and old parity first for a used stripe. This means that the small-write penalty well known in parity-based RAID can be alleviated in JOR.

When receiving a write request in the normal mode, if the corresponding bitmap of JBT is “0”, it is set to “1” before writing the data to the device and the new parity can be calculated with the write data without reading the old data and the old parity. Otherwise, the write request is processed in the same way as in the original approach. JOR does not change how read requests are handled.

D. Data consistency

Data consistency in JOR dictates that: (1) the key data structure must be safely stored without loss and (2) skipping unused stripes in reconstruction must not incur data loss.

First, JBT cannot be lost after it is created and initialized, otherwise the data in the disk array will become inconsistent. Accordingly, before a disk array is powered down, JBT must be flushed to the *superblock* in all disks of the disk array or a dedicated NVRAM. Then when the disk array is powered up again, JBT is read back to the memory. Moreover, to prevent data loss in the event of a power supply failure, JBT must be stored in non-volatile memory (*i.e.*, battery-backed RAM) during the entire period when the corresponding RAID is activated. In a word, the life cycle of JBT is consistent with that of its attached disk array.

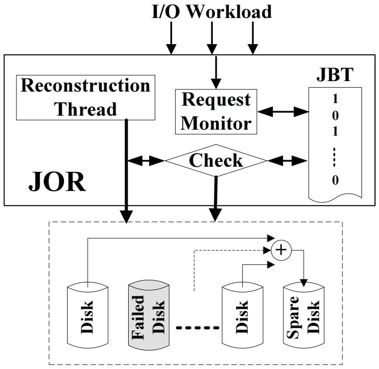


Figure 4. JOR organization.

Second, skipping the dead blocks by live-block recovery will incur possible data inconsistency and data loss, as described in Section II-B. For JOR, since all blocks in unused stripes of the surviving disks and spare disk are zero blocks, even if the unused stripes are not reconstructed, the data in these stripes are already consistent.

E. Prototype implementation

JOR can be embedded into any RAID software and can be easily incorporated into any platforms based on block-level devices. In this paper, we implement a JOR prototype by embedding it into the Linux Software RAID (MD).

Fig. 4 shows the organization of our current JOR prototype implementation, including three key modules. *Request Monitor* is responsible for keeping track of the storage space utilization status at the block level and managing JBT. *Reconstruction Thread* is responsible for reconstructing the failed data blocks to the spare disk. *Check* helps the Reconstruction Thread to decide whether a stripe should be reconstructed. Once a disk fails, the JOR Reconstruction Thread will be initiated. The reconstruction process has been described in Section III-C.

IV. Performance Evaluations

In this section, we evaluate the performance of JOR through extensive off-line and trace-driven experiments of our JOR prototype implementation, and analyze the memory overhead of JOR.

A. Experimental setup and methodology

We have incorporated JOR into MD’s default reconstruction algorithm PR [8] and PRO-powered PR (PRO for short) [9], [13]. In our experiments, we compare the performance of JOR-enhanced PR (JOR+PR for short) with that of PR and compare the performance of JOR-enhanced PRO (JOR+PRO for short) with that of PRO in terms of reconstruction time and average response time.

Table I
EXPERIMENTAL SETUP

Machine	Intel Xeon 3.0GHz, 1GB DDR RAM
Device Adapter	Highpoint RocketRAID 2220 SATA cards
Disks	WD2500YD SATA disk Average Rotation Time=4.2ms Average Seek Time=8.9ms
Trace Characteristic	Financial1.spc: Read Ratio=32.82%, Average IOPS=69 Financial2.spc: Read Ratio=82.39%, Average IOPS=125 WebSearch.spc: Read Ratio=100%, Average IOPS=113
Trace Replay	Raidmeter [9]
OS	Linux 2.6.11
RAID Software	MD & mdadm 2.5.3

Table I lists the experimental configuration. In addition, we use a separate IDE disk to house the operating system and other software, and use the main memory to substitute a battery-backed RAM for simplicity [10]. Since the workloads use only a fraction of the whole storage space, we limit the capacity of each disk to 10GB in our experiments. Performance evaluations use RAIDmeter [9], a block-level trace-replay tool, to replay traces and evaluate the I/O response time of the storage device.

The workloads used in our experiments are three traces obtained from the Storage Performance Council [27]. The first two, *Financial1* and *Financial2*, with different read/write ratios and I/O intensity, were collected from OLTP applications running at a large financial institution. The last one, *Websearch*, was collected from a system running a popular web search engine. Due to the relatively short reconstruction time in our current experimental setup, we use the beginning part of these three traces with lengths appropriate for our current reconstruction experiments. Moreover, we only use one part of the Websearch trace that is attributed to device zero as its IOPS (*i.e., I/O Per Second*) is too high for our RAID set, similar to [10].

B. I/O performance in the normal mode

As described in Section III-C, the JBT setting process spends some CPU cycles computing which stripe the current write request is involved. Thus, it may impact the I/O performance, specifically the write performance. *Is this impact acceptable to the higher-level systems and end users?* This subsection attempts to answer this question.

We conduct experiments to compare the I/O performance of original MD and JOR-enhanced MD in the normal mode on a platform of an 8-disk RAID5 disk array with a stripe unit size of 64KB, driven by the two Financial traces, respectively. Each test lasts one hour. In our experiments, the JBT setting process leads to less than 1.5% I/O performance loss in terms of average user response time (*i.e.*, 1.47% and 0.88% for the Financial1 and Financial2 traces, respectively). This level of performance cost is the same as that induced by the Journal-guided Resynchronization for Software RAID [28] and is arguably acceptable to the users.

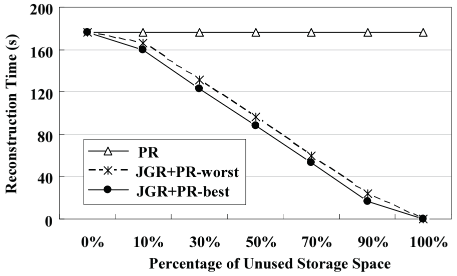


Figure 5. A comparison of PR and JOR-enhanced PR in off-line reconstruction time for a RAID5 disk array.

Given that the JBT setting process is carried out exclusively in the main memory and only affects write requests, along with the fact that processing capacity in storage systems is increasingly abundant, the I/O performance of JOR-enhanced MD in the normal mode is likely to approach that of the original MD.

C. Off-line reconstruction performance

We then conduct experiments on an 8-disk RAID5 disk array with a stripe unit size of 64KB and compare the off-line reconstruction time of PR and JOR+PR with respect to the percentage of unused storage space, as shown in Fig. 5. For JOR, we measure two cases [12]: (1) *the worst case*, where unused space is spread throughout the disk array, and (2) *the best case*, where unused space is compacted into a single portion of the disk array. Since the off-line reconstruction workflow of PRO is exactly the same as that of PR, we do not evaluate PRO in the off-line experiments.

Fig. 5 shows that, regardless of how much storage space has been used, the reconstruction time of PR is 176 second, which is restricted by the capacity and positioning time of the disks used in our experiments. Thus, PR is completely oblivious of the space utilization status. On the other hand, JOR+PR reduces the reconstruction time by an amount that is proportional to the percentage of unused storage space. Note that the results of JOR+PR are almost the same for both the best and worst case scenarios. Therefore, for simplicity, we only deploy the best case in the following experiments, that is, the unused space is sequential.

D. On-line reconstruction performance

In addition, we also conduct on-line reconstruction experiments on an 8-disk RAID5 disk array with a stripe unit size of 64KB. With respect to the percentage of unused storage space, we compare the on-line reconstruction time of PR, JOR+PR, PRO and JOR+PRO, as shown in Table II. The reconstruction bandwidth range is set be 10MB/s–200MB/s, that is, RAID favors user I/O requests while ensuring that the reconstruction speed is at least 10MB/s [10].

Table II shows that JOR+PR and JOR+PRO reduce the reconstruction time from their unenhanced counterparts by an amount that is approximately proportional to the percentage of unused storage space. The results are consistent to those

Table II
THE RECONSTRUCTION TIME RESULTS FOR A RAID5 DISK ARRAY.

Workload	Percentage of Unused Space	Reconstruction Time (second)					
		PR	JOR+PR	improved	PRO	JOR+PRO	improved
Financial1	10%	809.9	708.3	12.54%	778.2	677.5	12.95%
	30%		502.3	37.98%		484.4	37.75%
	50%		310.7	61.63%		281.4	63.84%
	70%		133.6	83.50%		128.1	83.54%
	90%		59.3	92.67%		55.4	92.89%
Financial2	10%	717.5	672.3	6.31%	677.7	642.9	5.13%
	30%		557.4	22.31%		531.9	21.50%
	50%		440.1	38.66%		432.8	36.13%
	70%		293.3	59.12%		291.4	57.00%
	90%		99.8	86.09%		99.5	85.32%
Websearch	10%	1000.2	900.1	10.01%	1000.5	900.7	9.97%
	30%		699.3	30.08%		700.3	30.00%
	50%		498.6	50.05%		500.6	49.96%
	70%		299.2	70.08%		299.3	70.08%
	90%		99.7	90.03%		99.9	90.02%

in the off-line reconstruction experiments. For the Financial1 trace, JOR improves the reconstruction performance of PR and PRO by an amount that is noticeably larger than the percentage of the unused storage space. For the Financial2 trace, the result is exactly the opposite. The reason is that the I/O intensity of the Financial1 trace is significantly lower in its early portion than in its later portion while the opposite is true for the Financail2 trace. For the Websearch trace, I/O requests are distributed uniformly and so JOR reduces the reconstruction time also uniformly.

Notice also that JOR does not directly optimize the I/O performance (*i.e.*, user response time) during reconstruction. In our experiments, the average user response time during reconstruction for the JOR-enhanced schemes is almost the same as that during the same period for the unenhanced counterparts. Nevertheless, JOR enables the disk array to return to the normal mode much more quickly and thus reduces the period of performance degradation significantly. As a result, we will not focus on the I/O performance during reconstruction in the following evaluations.

To examine the performance impact on JOR by the different RAID levels and different reconstruction bandwidths, we conduct experiments on an 8-disk disk array with variable RAID levels of RAID5, RAID6 and RAID10. We set the reconstruction bandwidth range to be the default (*i.e.*, 1MB/s–200MB/s), making the RAID reconstruction process yield to user I/O requests. For RAID6, we measure the reconstruction time when two disks fail simultaneously.

Fig. 6 shows that JOR reduces the reconstruction time by an amount that is roughly proportional to the increasing percentage of unused storage space. For the Financial1 trace, JOR does not reduce the reconstruction time by an amount that is absolutely proportional to the percentage of unused storage space . The Websearch trace is so intense that the reconstruction speed remains at approximately 1MB/s, the set minimal bandwidth, for the three RAID levels. Although the Financial2 trace is less intense than the Websearch trace, it is relatively stable so that the reconstruction speed remains also stable. The performance improvement by JOR for intense or stable traces is similar to that during off-line reconstruction for the most part.

In addition, we scale up the minimal reconstruction bandwidth and conduct experiments on different stripe unit sizes and different numbers of disks in a disk array to assess their impact on reconstruction performance. The results show the same trend as that mentioned above. In other words, *the reconstruction performance of JOR is only sensitive to the percentage of unused storage space.*

E. Memory overhead

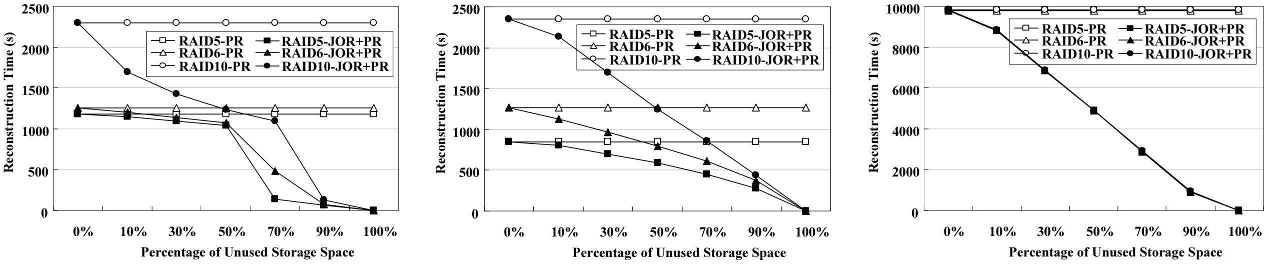
This part of the evaluation answers the question of **"how much extra memory does JOR require to implement JBT?"**. Suppose that there are n disks in the disk array, the individual disk capacity is cKB, and the stripe unit size is sKB. One bit denotes one stripe. Equation (1) shows the extra memory overhead to maintain JBT. Noticeably, the memory overhead is independent of the number of disks in a disk array.

$$JBT = \frac{c}{s \times 8} Byte \tag{1}$$

In our current implementation, if the stripe unit size is the same as the Linux kernel page size (*i.e.*, 4KB), representing all stripe status of a disk array consisting of n 500GB disks in JBT will consume 15.625MB memory. Since the cost of memory continues to decline and the capacity continues to increase [29], the non-volatile memory used for storing JBT is arguably no longer an issue.

V. RELATED WORK

Since the performance of reconstruction affects the reliability and availability of RAID-structured storage systems, the storage research community has put in a great deal of effort to the development of effective reconstruction schemes to minimize the reconstruction time and the performance



(a) Financial1 (b) Financial2 (c) Websearch

Figure 6. A comparison of PR and JOR-enhanced PR in on-line reconstruction time.

degradation in recent yeas. There are three general approaches to addressing the problem by focusing on different aspects of the reconstruction process.

The first general RAID reconstruction approach reorganizes the data layout to improve reconstruction performance. Distributed sparing [7] leverages the I/O response time and disk rebuild time by decreasing the parity group size. FARM [11] reduces the data recovery time by exploiting the excess disk capacity in large-scale distributed storage systems. The client-driven rebuild approach [30] based on a per-file RAID layout achieves good recovery performance by allowing the clients to build files in parallel.

The second general approach improves reconstruction performance by optimizing the reconstruction workflow [2], [6], [8]. SOR (Stripe-Oriented Reconstruction) [6] creates a number of reconstruction processes associated with stripes, while DOR (Disk-Oriented Reconstruction) [2] generates a group of processes associated with each corresponding disk. Although DOR outperforms SOR in reconstruction time, the improvement in reliability comes at the expense of performance degradation in user response time. PR (Pipelined Reconstruction) [8] takes advantage of the sequential property of track retrievals to pipeline the reading and writing processes.

The third general approach focuses on optimizing the reconstruction sequence [9], [31], [32]. The head-following reconstruction algorithm [32] tracks the movement of the disk heads to minimize the head positioning time by reconstructing data and parity in the region of the array currently being accessed by the users, but leads to almost immediate deadlock of the reconstruction process. The greedy algorithm [31] first reconstructs the tracks near the current head position, but keeps a large table of the tracks reconstructed. PRO (Popularity-based multi-threaded Reconstruction Optimization) [9] makes the reconstruction process rebuild the frequently accessed areas prior to other areas by exploiting the access locality, thus improving the system performance and reliability simultaneously.

On the other hand, Wu *et al.* [10] proposed WorkOut that outsources all writes and popular reads away from the degraded RAID set to a surrogate RAID set during reconstruction to significantly improve on-line reconstruction performance. JOR is complementary to WorkOut and can further improve the reconstruction performance.

In all above reconstruction approaches, all data blocks in surviving disks should be read and calculated during reconstruction. Live-block recovery [12] that reconstructs only the live data to the spare disk is the most related work to JOR, but it is impractical and not applicable for parity-based disk arrays. JOR not only ensures data consistency during reconstruction for all RAID levels, but also can be easily incorporated into most existing reconstruction algorithms without modifying the file system or operating system.

VI. Conclusion

The performance of the reconstruction process becomes increasing more crucial to the reliability and availability of RAID-structured storage systems. In this paper, we propose and evaluate a simple and practical reconstruction optimization scheme, called *JO*urnal-guided *R*econstruction (JOR), to significantly reduce the reconstruction time. JOR monitors the storage space utilization status at the block level to guide the reconstruction process so that only failed data on the used stripes is rebuilt to the spare disk. JOR is applicable for all RAID levels and can be easily adopted in various conventional reconstruction algorithms.

Experimental results demonstrate that both the JOR-enhanced PR and PRO reduce reconstruction times of their unenhanced counterparts by an amount that is proportional to the percentage of unused storage space. In addition, the performance overhead due to the JBT setting process is less than 1.5% of the system performance in the normal mode in our experiments, which is a reasonable and acceptable cost for the significantly reduced reconstruction time from the user's point of view.

Acknowledgment

This work is supported by the National Basic Research 973 Program of China under Grant No. 2004CB318201, 863 project 2008AA01A402, Changjiang innovative group of Education of China No. IRT0725, and the US NSF under Grant No. CCF-0621526.

References

[1] D. Patterson, G. Gibson, and R. Katz, “A Case for Redundant Arrays of Inexpensive Disks (RAID),” in *Proc. International Conference on Management of Data (SIGMOD'88)*, Chicago IL, USA, Jun. 1988, pp. 109–116.

[2] M. Holland, G. Gibson, and D. Siewiorek, "Fast, On-Line Failure Recovery in Redundant Disk Arrays," in *Proc. 23rd Annual International Symposium on Fault-Tolerant Computing (FTCS'93)*, Toulouse, France, Jun. 1993, pp. 421–433.

[3] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure Trends in a Large Disk Drive Population," in *Proc. 5th USENIX Conference on File and Storage Technologies (FAST'07)*, San Jose, CA, USA, Feb. 2007, pp. 17–28.

[4] 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 Conference on File and Storage Technologies (FAST'07)*, San Jose, CA, USA, Feb. 2007, pp. 1–16.

[5] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An Analysis of Latent Sector Errors in Disk Drives," in *Proc. 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (SIGMETRICS'07)*, San Diego, CA, USA, Jun. 2007, pp. 289–300.

[6] M. Holland, G. Gibson, and D. Siewiorek, "Architectures and Algorithms for On-Line Failure Recovery in Redundant Disk Arrays," *Journal of Distributed and Parallel Databases*, vol. 2, no. 3, pp. 295–335, 1994.

[7] R. Hou, J. Menon, and Y. Patt, "Balancing I/O Response Time and Disk Rebuild Time in a RAID5 Disk Array," in *Proc. Hawaii International Conference on Systems Sciences (HICSS'93)*, Hawaii, USA, Jan. 1993, pp. 70–79.

[8] J. Y. B. Lee and J. C. S. Lui, "Automatic Recovery from Disk Failure in Continuous-Media Servers," *IEEE Transaction On Parallel and Distributed Systems*, vol. 13, no. 5, pp. 499–515, May. 2002.

[9] L. Tian, D. Feng, H. Jiang, K. Zhou, L. Zeng, J. Chen, Z. Wang, and Z. Song, "PRO: A Popularity-based Multi-threaded Reconstruction Optimization for RAID-Structured Storage Systems," in *Proc. 5th USENIX Conference on File and Storage Technologies (FAST'07)*, San Jose, CA, USA, Feb. 2007, pp. 33–46.

[10] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao, "WorkOut: I/O Workload Outsourcing for Boosting RAID Reconstruction Performance," in *Proc. 7th USENIX Conference on File and Storage Technologies (FAST'09)*, San Francisco, CA, USA, Feb. 2009, pp. 239–252.

[11] Q. Xin, E. L. Miller, and T. J. E. Schwarz, "Evaluation of Distributed Recovery in Large-Scale Storage Systems," in *Proc. 13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04)*, Honolulu, HI, USA, Jun. 2004, pp. 172–181.

[12] M. Sivathanu, V. Prabhakaran, F. I. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Improving Storage System Availability with D-GRAID," in *Proc. 3rd USENIX Conference on File and Storage Technologies (FAST'04)*, San Francisco, CA, USA, Mar. 2004, pp. 15–30.

[13] L. Tian, H. Jiang, D. Feng, Q. Xin, and X. Shu, "Implementation and Evaluation of a Popularity-Based Reconstruction Optimization Algorithm in Availability-Oriented Disk Arrays," in *Proc. 24th IEEE Conference on Mass Storage Systems and Technologies (MSST'07)*, San Diego, CA, USA, Sep. 2007, pp. 101–106.

[14] C. Weddle, M. Oldham, J. Qian, A. A. Wang, P. Reiher, and G. Kuenning, "PARAID: The Gear-Shifting Power-Aware RAID," in *Proc. 5th USENIX Conference on File and Storage Technologies (FAST'07)*, San Jose, CA, USA, Feb. 2007, pp. 245–260.

[15] J. Gray, "Greetings! From a File System User," in *4th USENIX Conference on File and Storage Technologies (FAST'05). Keynote Address*, San Francisco, CA, USA, Feb. 2005.

[16] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, "A Five-Year Study of File-System Metadata," in *Proc. 5th USENIX Conference on File and Storage Technologies (FAST'07)*, San Jose, CA, USA, Feb. 2007, pp. 31–45.

[17] W. W. S. Hsu and A. J. Smith, "The performance effect of I/O optimizations and disk improvements," *IBM Journal Research and Development*, vol. 48, no. 2, pp. 255–289, 2004.

[18] M. Zhou and A. J. Smith, "Analysis of Personal Computer Workloads," in *Proc. 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'99)*, College Park, MD, USA, Oct. 1999, pp. 208–217.

[19] A. Traeger, E. Zadok, N. Joukov, and C. P. Wright, "A Nine Year Study of File System and Storage Benchmarking," *ACM Transactions on Storage*, vol. 4, no. 2, pp. 1–56, May. 2008.

[20] J. Ousterhout, H. D. Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson, "A Trace-Driven analysis of the UNIX 4.2 BSD File System," in *Proc. 15th ACM Symposimu on Operating Systems Princiles (SOSP'85)*, Orcas Island, WA, USA, Dec. 1985, pp. 15–24.

[21] M. Sivathanu, L. N. Bairavasundaram, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Life or Death at Block-Level," in *Proc. 6th Symposium on Operating Systems Design and Implementation (OSDI'04)*, San Francisco, CA, USA, Dec. 2004, pp. 379–394.

[22] X. Yu, B. Gum, Y. Chen, R. Y. Wang, K. Li, A. Krishnamurth, and T. E. Anderson, "Trading Capacity for Performance in a Disk Array," in *Proc. 4th Symposium on Operating Systems Design and Implementation (OSDI'00)*, San Diego, CA, USA, Oct. 2000, pp. 17–32.

[23] H. Huang, W. Hung, and K. G. Shin, "FS2: Dynamic Data Replication in Free Disk Space for Improving Disk Performance and Energy Consumption," in *Proc. 20th ACM Symposium on Operating Systems Principles (SOSP'05)*, Brighton, United Kingdom, Oct. 2005, pp. 263–276.

[24] D. Narayanan, A. Donnelly, and A. Rowstron, "Write Off-Loading: Practical Power Management for Enterprise Storage," in *Proc. 6th USENIX Conference on File and Storage Technologies (FAST'08)*, San Jose, CA, USA, Feb. 2008, pp. 253–267.

[25] D. Narayanan, A. Donnelly, E. Thereska, S. Elnikety, and A. Rowstron, "Everest: Scaling Down Peak Loads Through I/O Off-loading," in *Proc. 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI'08)*, San Diego, CA, USA, Dec. 2008, pp. 15–28.

[26] S. Rhea, R. Cox, and A. Pesterev, "Fast, Inexpensive Content-Addressed Storage in Foundation," in *Proc. 2008 USENIX Annual Technical Conference (USENIX'08)*, Boston, MA, USA, Jun. 2008, pp. 143–156.

[27] OLTP Application I/O and Search Engine I/O. UMass Trace Repository, "http://traces.cs.umass.edu/index.php/Storage/Storage."

[28] T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Journal-guided Resynchronization for Software RAID," in *Proc. 4th USENIX Conference on File and Storage Technologies (FAST'05)*, San Francisco, CA, USA, Dec. 2005, pp. 87–100.

[29] J. Gray, "Tape is Dead, Disk is Tape, Flash is Disk, RAM Locality is King," in *3rd Biennial Conference on Innovative Data Systems Research (CIDR'07). Keynote Address*, San Francisco, CA, USA, Jan. 2007.

[30] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable Performance of the Panasas Parallel File System," in *Proc. 6th USENIX Conference on File and Storage Technologies (FAST'08)*, San Jose, CA, USA, Feb. 2008, pp. 17–33.

[31] E. Bachmat and J. Schindler, "Analysis of Methods for Scheduling Low Priority Disk Drive Tasks," in *Proc. 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (SIGMETRICS'02)*, Marina del Rey, CA, USA, Jun. 2002, pp. 55–65.

[32] M. Holland, "On-Line Data Reconstruction in Redundant Disk Arrays," Ph.D Dissertation, Carnegie Mellon University, 1994.