PRO: A Popularity-based Multi-threaded Reconstruction Optimization for RAID-Structured Storage Systems

Lei Tian[†‡], Dan Feng[†‡], Hong Jiang[§], Ke Zhou[†‡],
Lingfang Zeng[†‡], Jianxi Chen[†‡], Zhikun Wang[†‡], Zhenlei Song[†]
[†]*Huazhong University of Science and Technology*
[‡]*Wuhan National Laboratory for Optoelectronics*
[§]*University of Nebraska-Lincoln*
E-mail:{dfeng, k.zhou, ltian}@hust.edu.cn, jiang@cse.unl.edu

Abstract

This paper proposes and evaluates a novel dynamic data reconstruction optimization algorithm, called *p*opularity-based multi-threaded *r*econstruction *o*ptimization (PRO), which allows the reconstruction process in a RAID-structured storage system to rebuild the frequently accessed areas prior to rebuilding infrequently accessed areas to exploit access locality. This approach has the salient advantage of simultaneously decreasing reconstruction time and alleviating user and system performance degradation. It can also be easily adopted in various conventional reconstruction approaches. In particular, we optimize the disk-oriented reconstruction (DOR) approach with PRO. The PRO-powered DOR is shown to induce a much earlier onset of response-time improvement and sustain a longer time span of such improvement than the original DOR. Our benchmark studies on read-only web workloads have shown that the PRO-powered DOR algorithm consistently outperforms the original DOR algorithm in the failure-recovery process in terms of user response time, with a 3.6%~23.9% performance improvement and up to 44.7% reconstruction time improvement simultaneously.

1. Introduction

Reliability and availability are the most common issues that administrators of storage systems are concerned with and directly affect the end users of such systems. Frequent or long downtimes and data losses are clearly intolerable to the end users. Research has shown that 50 percent of companies losing critical systems for more than 10 days never recover, 43 percent of companies experiencing a disaster never reopen, and 29 percent of the remaining close within two years [1]. For some companies that do survive long-term performance degradation, the resulting penalties can become too costly to ignore. For example, for data services at a large-scale data center, a typical Service Level Agreement specifies the percentage of transaction response times that can exceed a threshold (e.g. seconds), and the penalties for failing to comply can be very costly [2].

As a result, redundant storage systems, such as RAID [3], have been widely deployed to prevent catastrophic data losses. Various RAID schemes employ mirroring, parity-encoding, hot spare and other mechanisms [4] to tolerate the failures of disk drives. If a disk failure occurs, RAID will automatically switch to a recovery mode, often at a degraded performance level, and activate a background reconstruction process for data recovery. The reconstruction process will rebuild data units of the failed disk onto the replacement disk while RAID continues to serve I/O requests from clients. After all of the contents are rebuilt, RAID returns the system to the normal operating mode.

Advances in storage technology have significantly reduced cost and improved performance and capacity of storage devices, making it possible for system designers to build extremely large storage systems comprised of tens of thousands or more disks with high I/O performance and data capacity. However, reliability of individual disk drives has improved very slowly, compared to the improvement in their capacity and cost, resulting in a very high overall failure rate in a system with tens of thousands of disk drives. To make matters worse, the time to rebuild a single disk has lengthened as increases in disk capacity far outpace increases in disk bandwidth, lengthening the "window of vulnerability" during which a subsequent disk failure (or a series of subsequent disk failures) causes data loss [5]. Furthermore, many potential applications of data replication or migration, such as on-line backup and capacity management, are faced with similar challenges as the reconstruction process.

Therefore, the efficiency of the reconstruction algorithm affects the reliability and performance of RAID-structured storage systems directly and significantly. The goals of reconstruction algorithms are (1) to mini-

mize the reconstruction time to ensure system reliability and (2) to minimize the performance degradation to ensure user and system performance, simultaneously. Existing effective approaches, such as disk-oriented reconstruction (DOR) [6] and pipelined reconstruction (PR) [7], optimize the recovery workflow to improve the reconstruction process's parallelism. Based on stripe-oriented reconstruction (SOR) [8], both DOR and PR execute a set of reconstruction processes to rebuild data units by exploiting parallelism in processes and pipelining. While they both exploit reconstruction parallelism, the problem of optimizing reconstruction sequence remains unsolved. In a typical recovery mode, the reconstruction process rebuilds data units while RAID continues to serve I/O requests from the clients, where clients' accesses can adversely and severely affect the reconstruction efficiency because serving clients' I/O requests and reconstruction I/O requests simultaneously leads to frequent long seeks to and from the multiple separate data areas. Optimizing the reconstruction sequence with users' workload characteristics, we believe, is a key in improving existing and widely-used reconstruction approaches such as those mentioned above.

In this paper, we propose a Popularity-based multi-threaded Reconstruction Optimization (PRO) algorithm to optimize the existing reconstruction approaches, which exploits I/O workload characteristics to guide the reconstruction process. The main idea behind our PRO algorithm is to reconstruct high-popularity data units of a failed disk, which are the most frequently accessed units in terms of the workload characteristics, prior to reconstructing other units. Furthermore, by fully exploring the access patterns, the PRO algorithm has the potential to recover many units ahead of users' accesses with high probability to avoid performance degradation caused by recovery.

To the best of our knowledge, little research effort has been directed towards integrating workload characteristics into the reconstruction algorithm. Most of the existing reconstruction algorithms perform a sequential reconstruction on the failed disk regardless of the workload characteristics. Taking into account the workload distribution (such as spatial locality, temporal locality, etc.), the PRO approach attains a flexible and pertinent reconstruction strategy that attempts to reduce reconstruction time while alleviating the performance degradation. PRO achieves this by inducing a much earlier onset of response-time performance improvement and sustaining a longer time span of such improvement while reducing reconstruction time during recovery.

We implement the PRO algorithm based on DOR, and our benchmark studies on read-only web workloads have shown that the PRO algorithm outperforms DOR by 3.6%~ 23.9% in user response time and by up to 44.7% in reconstruction time, simultaneously. Because the PRO algorithm is specifically designed to only generate the optimal reconstruction sequence based on workload characteristics without any modification to the reconstruction workflow or data layout, it can be easily incorporated into most existing reconstruction approaches of RAID and achieve significant improvement on system reliability and response time performance.

The rest of this paper is organized as follows. Related work and motivation are presented in Section 2. In Section 3, we describe the algorithm and implementation of the PRO approach. Performance evaluation and discussions are presented in Section 4. We conclude the paper in section 5 by summarizing the main contributions of this paper and pointing out directions for future research.

2. Related Work and Motivation

In this section, we first present the background and related work on the reconstruction issues. Second, we reveal the ubiquitous properties of popularity and locality of typical workloads and thus elucidate the motivations for our research.

2.1. Existing Reconstruction Approaches

There has been substantial research reported in the literature on reliability and recovery mechanisms in RAID or RAID-structured storage systems. Since the advent of disk arrays and RAID, researchers have been working to improve reliability and availability of such systems by devising efficient reconstruction algorithms for the recovery process, with several notable and effective outcomes.

There are two general approaches to disk array reconstruction depending on the source of improvement. The first general approach improves performance by reorganizing the data layout of spare or parity data units during recovery.

Hou et al. [9] considered an approach to improving reconstruction time, called distributed sparing. Instead of using a dedicated spare disk which is unused in the normal mode, they distribute the spare space evenly across the disk array. Their approach can result in improved response times and reconstruction times because

one extra disk is available for normal use and less data needs to be reconstructed on a failure. As an extension to the distributed sparing approach in large-scale distributed storage systems, Xin et al. [10] presented a fast recovery mechanism (FARM) to dramatically lower the probability of data loss in large-scale storage systems.

The second general approach improves performance by optimizing the reconstruction workflow. Because this approach needs not modify the data organization of RAID, it is more prevalent in RAID implementations. Therefore, it is also the approach that our proposed algorithm will be based on.

Disk-oriented reconstruction (DOR) and pipelined reconstruction (PR) are two representative and widely-used reconstruction approaches. The DOR algorithm was proposed by Holland [11] to address the deficiencies of both the single-stripe and parallel-stripe reconstruction algorithms (SOR). Instead of creating a set of reconstruction processes associated with stripes, the array controller creates a number of processes, with each associated with one disk. The advantage of this approach is that it is able to maintain one low-priority request in the queue for each disk at all times, thus being able to absorb all of the array's bandwidth that is not absorbed by the users. Although DOR outperforms SOR in reconstruction time, the improvement in reliability comes at the expense of performance in user response times.

To address the reliability issue of continuous-media servers, Lee and Lui [7] presented a track-based reconstruction algorithm to rebuild lost data in tracks. In addition, they presented a pipelined reconstruction (PR) algorithm to reduce the extra buffers required by the track-based reconstruction algorithm. Muntz and Lui [12] conducted a performance study on reconstruction algorithms using an analytical model. Their first enhancement, reconstruction with redirection of reads, uses the spare disk to service disk requests to the failed disk if the requested data has already been rebuilt to the spare disk. They also proposed to piggyback rebuild requests on a normal workload. If a data block on the failed disk is accessed and has not yet been rebuilt to the spare disk, the data block is regenerated by reading the corresponding surviving disks. It is then a simple matter of also writing this data block to the spare disk.

The basic head-following algorithm [11] attempts to minimize head positioning time by reconstructing data and parity in the region of the array currently being accessed by the users. The problem with this approach is that it leads to almost immediate deadlock of the reconstruction process. Since the workload causes the disk heads to be uncorrelated with respect to each other, head following causes each reconstruction process to fetch a reconstruction unit from a different parity stripe. The buffers are filled with data units from different parity stripes and hard to be freed, and so reconstruction deadlocks.

Assuming that the probability of a second disk failure (while the first failed disk is under repair) is very low, Kari et al. [13] presented a delayed repair method to satisfy the response time requirement, which introduced a short delay between repair requests to limit the number of repair read (or write) requests that any user disk request might need to wait. With regard to RAID-structured storage systems, it may not be practical to adopt such a delayed repair method directly because of the relatively high probability of a second disk failure.

To address the problem of performance degradation during recovery, Vin et al. [14] integrated the recovery process with the decompression of video streams, thereby distributing the reconstruction process across the clients to utilize the inherent redundancy in video streams. From the viewpoint of a file system's semantic knowledge, Sivathanu et al. [15] proposed a live-block recovery approach in their D-GRAID, which changes the recovery process to first recover those blocks which are live (i.e., belong to allocated files or directories).

Of the above related work on disk array reconstruction or recovery based on reconstruction workflow optimizations, the DOR algorithm still dominates in its applications and implementations; whereas, the others can be arguably considered either rooted at DOR or DOR's extensions or variations. In general, while DOR absorbs the disk array's bandwidth, its variations attempt to take advantage of user accesses. On the other hand, our proposed PRO algorithm improves over these approaches by not only preserving the inherent sequentiality of the reconstruction process, but also dynamically scheduling multiple reconstruction points and considering the reconstruction priority. What's more important, the PRO algorithm makes use of the access locality in I/O workloads, which is ignored by the above reconstruction algorithms, and improves the response-time performance and the reliability performance simultaneously.

2.2. Popularity and Locality of Workloads

A good understanding of the I/O workload characteristics can provide a useful insight into the reason behind DOR's inability to minimize application's performance

degradation, thus helping us improve the DOR algorithm and other related algorithms to address this problem. In particular, workload characteristics such as temporal locality and spatial locality can be exploited to alleviate performance degradation while reducing reconstruction time. Experience with traditional computer system workloads has shown the importance of temporal locality to computer design [16].

In many application environments, 80% accesses are always directed to 20% of the data, a phenomenon that has long been known as Pareto's Principle or "The 80/20 Rule" [17]. Pareto's Principle points to the existence of temporal locality and spatial locality in various kinds of I/O workloads. The presence of access locality in I/O workloads has been well known in the literature. Gomez & Santonja [17] studied three sets of I/O traces provided by Hewlett-Packard Labs, and showed that some of the blocks are extremely hot and popular while other blocks are rarely or never accessed. In the media workload, Cherkasova and Gupta [18] found that 14%~30% of the files accessed on the server accounted for 90% of the media sessions and 92%~94% of the bytes transferred, and were viewed by 96%~97% of the unique clients.

Studies have further indicated that access locality is more pronounced in workload characteristics of web servers [19], and identified three distinct types of web access locality, namely, static, spatial and temporal locality. Static locality refers to the observation that 10% of files accessed on a web server typically account for 90% of the server requests and 90% of the bytes transferred [20]. Roselli et al. [21] found that for all workloads, file access patterns are bimodal in that most files tend to be mostly-read or mostly-written. The web workload has far more read bandwidth than any other workload but has relatively little write bandwidth. Sikalinda et al. [22] found that in the web search workload almost all the operations are reads (i.e., 99.98% of the total number of operations). Since typical on-line transaction processing type (OLTP-like) workloads are read-dominated [13], the load on the surviving disks increased by typically between 50-100% in the presence of a disk failure. This severely degrades the performance as observed by the users, and dramatically lengthens the period of time required to recover the lost data and store it on a replacement disk.

Motivated by insightful observations made by other researchers and by our own research, we propose to integrate temporal locality and spatial locality into the reconstruction algorithms to improve their effectiveness. If the frequently accessed data units could be reconstructed prior to reconstructing all others, the effect of performance degradation can be potentially hidden from the users in their subsequent accesses to the same data units, especially during the recovery process.

3. Design and Implementation of PRO

Due to the aforementioned shortcomings of existing parallel reconstruction approaches (such as DOR and PR), a solution must be sought to optimize the reconstruction sequence. Based on prior research by other researchers and by us, we believe that the key to our solution is to exploit the workload characteristics into the reconstruction algorithm. By making an appropriate connection between the external workload and the internal reconstruction I/O activities, we propose a Popularity-based multi-threaded Reconstruction Optimization algorithm (PRO) to combine the locality of the workloads with the sequentiality of the reconstruction process.

The main idea behind the PRO algorithm is to monitor and keep track of the dynamic popularity changes of data areas, thus directing the reconstruction process to rebuild highly popular data units of a failed disk, prior to rebuilding other units. More specifically, PRO divides data units on the replacement disk into multiple non-overlapping but consecutive data areas, called "hot zones". Correspondingly, PRO employs multiple independent reconstruction threads, where each reconstruction thread is responsible for one of the hot zones and the priority of each thread is determined by the latest frequency of users' accesses to its hot zone. The PRO algorithm adopts a priority-based time-sharing scheduling algorithm to schedule the reconstruction threads. The scheduler activates the thread with the highest priority periodically by allocating a time slice to the thread that begins rebuilding the remaining data units of its hot zones until the time slice is used up.

Different from the strictly sequential sequence of most existing reconstruction approaches, PRO generates a workload-following reconstruction sequence to exploit the locality of workload characteristics.

In this section, we first outline the design principles behind PRO, which is followed by a detailed description of the PRO algorithm via an example, as well as an overview of PRO's implementation.

3.1. Design Principles

As Holland concluded [11], a reconstruction algorithm must preserve the inherent sequentiality of the

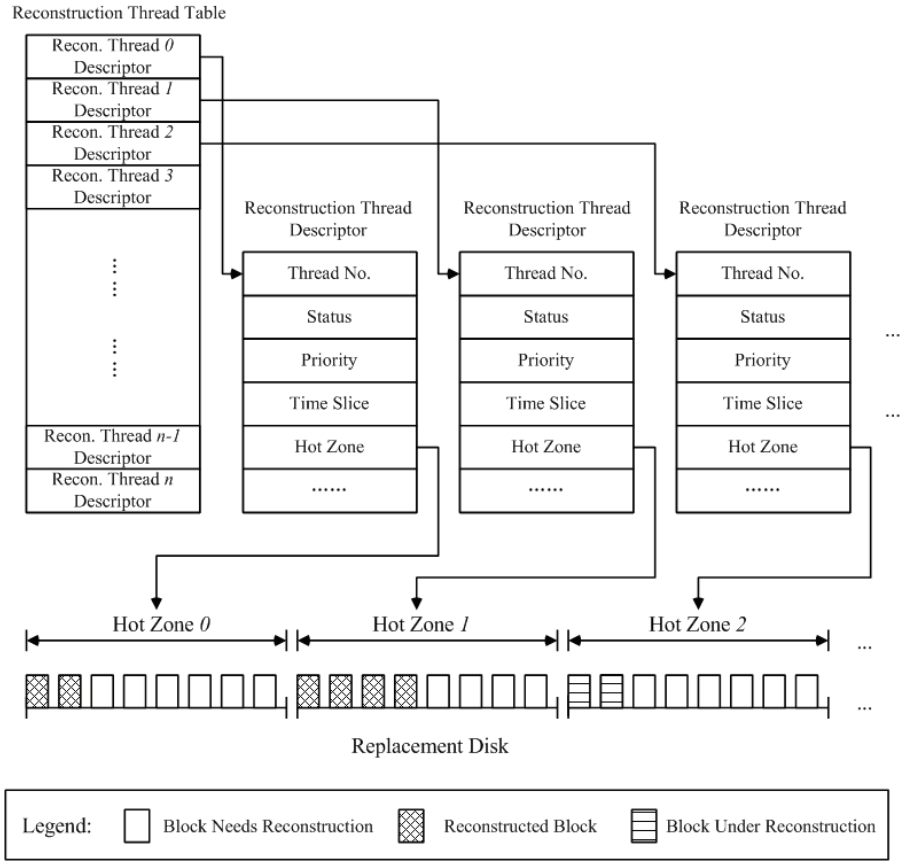


Figure 1: Reconstruction thread descriptor.

reconstruction process, since a disk drive is able to service sequential accesses at many times the bandwidth of random accesses. This leads to the development of the disk-oriented reconstruction (DOR) algorithm and the pipelined reconstruction (PR) algorithm, and to the rejection of the head-following approaches. On the other hand, to take full advantage of locality of access in the I/O workload, we believe that a reconstruction algorithm should rebuild data units with high-popularity prior to rebuilding low-popularity or no-popularity data units on the failed disk. Consequently, frequent long seeks to and from the multiple separate popular data areas result in seek and rotation penalties for multiple reconstruction points.

To strike a good balance between the above two seemingly conflicting goals of maintaining sequentiality and exploiting access locality, the PRO algorithm effectively combines "global decentralization" with "local sequentiality" in the reconstruction process. Inspired by the design principles of classical time-sharing operating systems, PRO adopts the ideas of "divide and conquer" and "time-sharing scheduling" to achieve the above goals.

3.2. The Reconstruction Algorithm

To obtain an optimal reconstruction sequence, PRO tracks the popularity changes of data areas and generates a workload-following reconstruction sequence to combine locality with sequentiality.

A description of the PRO algorithm is given as follows.

First, PRO divides data units on the replacement disk into multiple non-overlapping but consecutive data areas called "*hot zones*" (the algorithm for defining the appropriate number of hot zones is described in Section 3.4), with each being associated with a popularity measure determined by the latest frequency of users' accesses to it. Correspondingly, the entire reconstruction process is divided into multiple independent reconstruction threads with each being responsible for rebuilding one of the hot zones. The priority of a reconstruction thread is dynamically adjusted according to the current popularity of its hot zone. Similar to a real thread in operating systems, each reconstruction thread has its reconstruction thread descriptor (see Figure 1

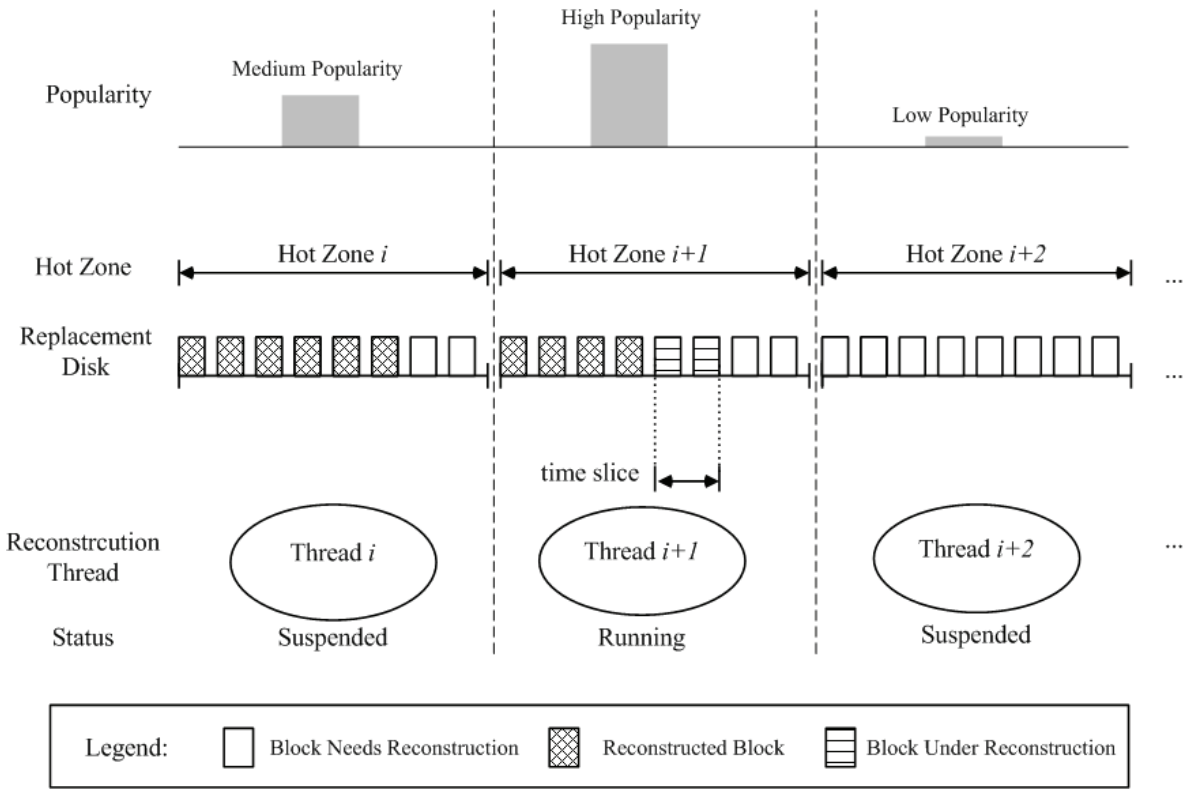


Figure 2: A snapshot of the PRO algorithm at work.

that shows its data structure and content) that includes properties such as status, priority, time slice, etc.

Second, once all of the reconstruction threads are initialized, a reconstruction scheduler selects the reconstruction thread with the highest priority and allocates a time slice to it, which activates this thread to rebuild the remaining data units of its hot zone until the time slice is used up. If the thread runs out of its time slice, the reconstruction scheduler suspends it, re-selects the reconstruction thread that has the current highest priority, and allocates one time slice to it. This process repeats until all the data units have been rebuilt. Figure 2 is a snapshot of the PRO algorithm at work. From Figure 2, one can see three reconstruction threads and their respective hot zones with different popularity. Because the hot zone of the middle thread has the highest popularity, the thread is switched to the running state and allocated a time slice by the reconstruction scheduler. At the same time, the other two threads are suspended due to their low popularity.

The PRO approach uses the reconstruction threads to track the popularity changes of the hot zones and always picks up the most popular data area to rebuild prior to rebuilding others, thus achieving the goal of fully exploiting access locality based on popularity (via global decentralization) during the reconstruction process.

A time slice in PRO is always associated with a number of the consecutive data units. In our implementation, a time slice is set to be 64, that is, 64 consecutive data units. With this approach, PRO achieves the goal of maintaining local sequentiality of the reconstruction process.

After the above step, the data units in the reconstruction queue are workload-following and locally sequential.

3.3. The PRO Structure and Procedures

The PRO architecture consists of three key components: Access Monitor (AM), Reconstruction Scheduler (RS) and Reconstruction Executor (RE). AM is responsible for capturing and analyzing clients' access locality and adjusting popularities of the corresponding hot zones. The responsibility of RS is to select data units of the hot zone with the highest popularity, or the most frequently-accessed data units, generate the corresponding tasks and put them into a FIFO reconstruction task queue. The main function of RE is to fetch jobs from

the reconstruction task queue and rebuild the corresponding units on the replacement disk. The operations of each of the three PRO components are detailed below.

Access Monitor (AM):

Repeat
1. AM receives the I/O request and determines its type and address information.
2. If the type is read and the address is located on the failed disk, increase the popularity of the corresponding hot zone (including this address).
Until (the failed disk has been reconstructed)

Reconstruction Scheduler (RS):

Repeat
1. Check the time slice of the reconstruction thread whose status is running.
2. If the thread has no time slice left, select the reconstruction thread with the highest popularity from all reconstruction threads, and allocate a time slice to it. At the same time, reset the popularity of all reconstruction threads to zero.
3. Set the status of the chosen reconstruction thread to the running state, and set the status of other threads to the suspended state. Meanwhile, reduce the remaining time slice of the chosen reconstruction thread by one.
4. The chosen reconstruction thread begins rebuilding a remaining data units in its hot zone, generating a corresponding reconstruction job to obtain this reconstruction unit, and queuing it to the tail of the reconstruction job queue.
5. If all data units in this hot zone are reconstructed, reclaim the chosen reconstruction and its hot zone.
Until (the failed disk has been reconstructed)

Reconstruction Executor (RE):

In fact, the functions of RE are the same as those of the DOR algorithm except for the following subtle difference: RE chooses the next job from the queue, not the sequentially next data unit one by one as the DOR algorithm. The disk array controller creates *C* processes, each associated with one disk. Each of the *C-1* processes associated with a surviving disk executes the following loop:

Repeat
1. Fetch the next job from the reconstruction job queue on this disk that is needed for reconstruction.
2. Issue a low-priority request to read the indicated unit into a buffer.
3. Wait for the read to complete.
4. Submit the unit's data to a centralized buffer manager for XOR, or Block the process if the buffer is full.
Until (all necessary units have been read)

The process associated with the replacement disk executes the following operations:

Repeat
1. Request sequentially the next full buffer from the buffer manager.
2. Issue a low-priority write of the buffer to the replacement disk.
3. Wait for the write to complete.
Until (the failed disk has been reconstructed)

3.4. Implementation Issues

How is the popularity data collected, stored, updated?
Whenever the reconstruction process starts, the Access Monitor begins to track and evaluate the statistics of clients' accesses to the failed disk. The popularity data of each hot zone is stored in the "popularity" counter of the hot zone. It can keep better track of the dynamic changes of popular data areas if the popularity data is collected, stored and updated even after the reconstruction process starts. This, however, leads to a 2% to 4% system performance loss (similar to the impact of the I/O trace tools) while the probability of a disk failure is relatively low.

Once a client accesses a hot zone, the value of its popularity counter is increased by one. Each time the reconstruction scheduler finishes a thread scheduling procedure, all the popularity counters of hot zones are reset to zeros to be ready for a new round of monitoring. This implies that short-term, rather than long-term, popularity history is captured by the PRO implementation. This is based on our belief that the former is more important than the latter during recovery because current or recent past popularity change tends to point to newly and rapidly accessed data areas that should be reconstructed more urgently.

How many zones are used?
In PRO, hot zones are fully dynamically created, updated and reused, including the corresponding

Component	Description
CPU	Intel Pentium4 3GHz
Memory	512M DDR SDRAM
SCSI HBA	LSIlogic 22320R
SCSI Disk	Seagate ST3146807LC
OS	NetBSD 2.1
RAID software	RAIDframe [26]

Table 1: The platform for performance evaluation.

reconstruction threads. At the beginning of the recovery process, there is no reconstruction thread or hot zone.

When a client accesses a data unit, say, numbered N, on the failed disk, a thread will be created and initiated, along with its corresponding hot zone of default length L and range N to $N+L$. After initialization, the reconstruction thread switches to the *alive* status and waits for scheduling. If a client accesses a data unit belonging to this hot zone, its popularity will be increased by one. But if a client accesses a data unit not belonging to any of the existing hot zones, a new thread, along with its new hot zone, will be initialized and created in a manner described above. Assuming that the data unit is numbered M, if $M<N$ and $N-M<L$, the range of the newly created hot zone is set to be M to $N-M$ to ensure non-overlapping between two adjacent hot zones, else the range is set to be M to $M+L$. In the other words, the default length L is the maximum threshold. When all data units in a hot zone are rebuilt, the hot zone and the corresponding reconstruction thread will be reclaimed for the reuse purpose.

Clearly, the number and length of hot zones are not fixed but dynamically adjusted by the workload. In the implementation, we set the maximum length of a hot zone to 1024 and the maximum number of hot zones (thus of the reconstruction threads) to 128, that is, the PRO algorithm can track the popularity changes of 128 data areas simultaneously.

How are threads scheduled?
The PRO algorithm adopts a priority-based time-sharing scheduling algorithm to schedule multiple reconstruction threads. It must be noted that the PRO algorithm's scheduler bases its selection decision mostly on a thread's priority (or its corresponding zone's popularity).

4. Performance Evaluations

This section presents results of a comprehensive experimental study comparing the proposed PRO-powered DOR algorithm (PRO for short) with the original DOR algorithm. To the best of our knowledge, the DOR algorithm is arguably the most effective among existing reconstruction algorithms in part because it is implemented in many software and hardware RAID products [23-25] and most widely studied in the literature. This performance study analyzes reconstruction performance in terms of user response time and reconstruction time, and algorithm complexity.

4.1. Experimental Setup

We conducted our performance evaluation of the two algorithms above on a platform of server-class hardware and software (see Table 1). The speed of the Seagate ST3146807LC disks is 10000 rpm, its average seek time is 4.7ms, and its capacity is 147GB. We use 2 SCSI channels and each channel attaches 5 disks.

Because the NetBSD platform has no appropriate benchmark to replay the I/O trace, we implemented a block-level replay tool, RAIDmeter, to evaluate performances of the two reconstruction algorithms since most databases run not on a file system but on the raw devices directly. The main function of RAIDmeter is to replay the traces and evaluate the I/O response time, that is, to open the block device, send the designated read/write requests to the device in terms of the time-stamp and request type of each I/O event in the trace, wait for the completion of the I/O requests and gather the corresponding I/O results, such as the response time, throughput and so on. Since the NetBSD 2.1 operating system cannot support read or write operations to files larger than 2GB, it is very inconvenient to benchmark the performance of disk arrays consisting of hard drives with hundreds of GB capacity each. As a result, we had to limit the capacity of every disk to 5GB. In addition, RAIDmeter adopts the approach of randread [27] to overcome the capacity limitation to support I/O accesses to files of tens of GBs. We believe that RAIDmeter is the best block-level trace-replay tool available for the current NetBSD platform.

4.2. Methodology and Workload

We evaluated our design by running trace-driven evaluations over the web I/O traces identified from the Storage Performance Council [28] because the web workloads tend to most pronouncedly reveal the nature of locality. These traces were collected from a system running a web search engine, and they are all read-dominant and with high locality. Because of the absolute read-domination (99.98%) in all of the web-search-engine I/O traces, we filtered out the write requests and feed only read events to our RAIDmeter replay tool.

Trace Name	Number of Requests	Popularity (20% most active area by size handles so much of total requests)	Average Inter-Arrival Time (ms)
Web Trace 1	842,535	69.99%	3.74
Web Trace 2	999,999	70.12%	3.73
Web Trace 3	999,999	69.89%	5.71

Table 2: The characteristics of three web traces.

RAID Level	Number of Disks	Reconstruction Time (second)								
		Web Trace 1			Web Trace 2			Web Trace 3		
		DOR	PRO	improved	DOR	PRO	improved	DOR	PRO	improved
RAID5	3	1123.8	666.5	40.7%	1058.3	585.2	44.7%	452.3	351.6	22.3%
	5	457.4	353.1	22.8%	374.5	304.1	18.8%	242.8	203.9	16.0%
	7	344.0	319.1	7.2%	325.7	278.5	14.5%	238.2	210.8	11.5%
	9	243.5	240.3	1.3%	231.5	215.3	7.0%	192.4	184.5	4.1%
RAID1	2	1208.1	1157.7	4.2%	938.0	870.1	7.3%	424.2	409.5	3.7%

Table 3: A comparison of PRO and DOR reconstruction times as a function of the number of disks.

RAID Level	Number of Disks	Average User Response Time during recovery (millisecond)								
		Web Trace 1			Web Trace 2			Web Trace 3		
		DOR	PRO	improved	DOR	PRO	improved	DOR	PRO	improved
RAID5	3	31.8	24.2	23.9%	28.5	23.9	16.0%	27.4	23.1	15.6%
	5	21.7	19.3	11.1%	21.0	18.7	11.0%	20.0	17.8	11.3%
	7	25.0	23.8	5.1%	22.5	21.4	4.5%	22.6	20.0	11.8%
	9	19.1	18.2	4.5%	19.6	17.3	11.5%	19.5	18.8	3.6%
RAID1	2	29.5	28.2	11.1%	21.4	20.6	11.0%	18.8	17.8	11.3%

Table 4: A comparison of PRO and DOR user response times as a function of the number of disks.

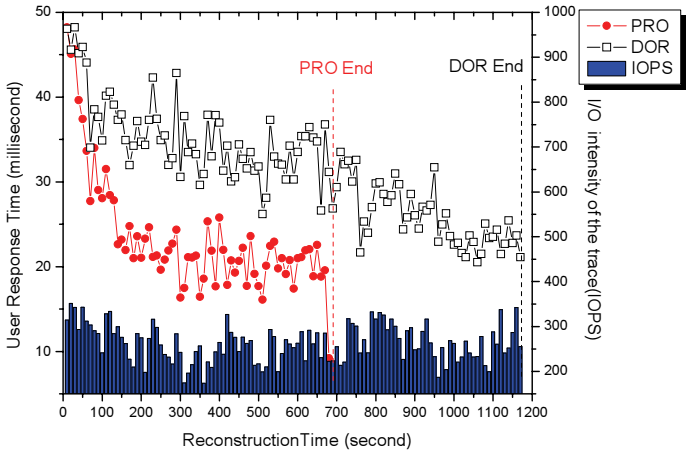
Owing to the relatively short reconstruction times in our current experimental setup, it may not be necessary to use the full daily-level traces. Thus, we only reserved the beginning part of the traces with lengths appropriate for our current reconstruction experiments. Table 2 shows the relevant information of the web-search-engine I/O traces.

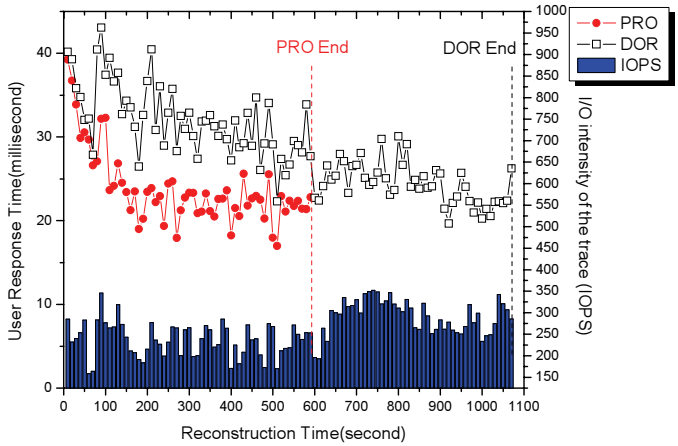
4.3. Reconstruction Performance

We first conducted our performance evaluation of the two reconstruction algorithms on a platform of a RAID-5 disk array consisting of variable number of disks and 1 hot-spare disk, with a stripe unit size of 64KB and a RAID-1 disk array consisting of 2 disks and 1 hot-spare disk, with a stripe unit of 64KB.

Tables 3 and 4 show the measured reconstruction times and user response times of DOR and PRO, respectively, and reveal the efficacy of the PRO algorithm in improving the array's reconstruction time and user response time simultaneously during recovery. Across all given workloads, RAID levels and disk numbers in our experiments, the PRO algorithm almost consistently outperforms the DOR algorithm in reconstruction time and user response time by up to 44.7% and 23.9%, respectively.

It is not uncommon that a second disk drive can fail shortly after a first disk failure in a very large-scale RAID-structured storage system, and thus it is very important to shorten the reconstruction time to avoid a





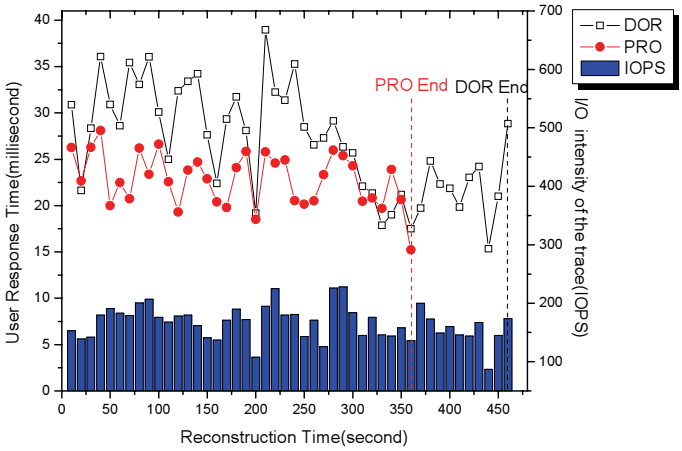


Figure 3 (a), (b), (c): A comparison of PRO and DOR user response time as a function of the respective traces: web trace 1, 2 and 3. In all of the figures, the bottom columns mean the I/O intensity of the trace and the two curves mean the PRO and DOR user response time trend during recovery. The RAID-5 disk array consists of 3 disks and 1 hot-spare disk, with a stripe unit size of 64KB.

In summary, by exploiting access locality the PRO algorithm consistently outperforms the DOR algorithm both in reliability and in performance during recovery.

4.4. Complexity Analysis

Space Complexity
Because the PRO approach is based on the implementation of the DOR algorithm, the memory requirements are almost the same as that for the latter. Compared with DOR, PRO requires extra memory only for the storage of reconstruction thread descriptors. In our actual implementation, each thread descriptor consumes 1KB memory, and if we set the maximum threshold of the number of threads to 128 (as is in our implementation), the extra memory needed for the PRO approach is about 128KB.

Time Complexity
Since the time for each thread-scheduling event is mostly consumed in the selection of the highest-priority thread from all of the candidate threads, we can estimate approximately that the computation overhead of the PRO algorithm is *O(n)*, where n is the total number of the existing threads. If we use a priority queue, the computation complexity can be reduced to *O(logn)*. However, the computation overhead will be negligible on a modern processor compared with the enormous I/O latency of a disk.

Implementation Complexity
We briefly quantify the implementation complexity of PRO. Table 5 lists the lines of code, counted by the number of semicolons and braces, which are modified or added to the RAIDframe. From the table, one can see that very few additions and modifications are needed to add to the reconstruction module. It is clear that most of the complexity is found in the Reconstruction Scheduler, because this module is in fact the bridge between the Access Monitor and the Reconstruction Executor, and its functions are shared with these modules. Compared with the 39691 lines of the RAIDframe implementation, the modifications for PRO occupy only 1.7 % of the total lines.

Component	Mod. Lines	Add. Lines	Total Lines
AM	0	84	84
RS	0	590	590
RE	2	12	14
Total	2	686	688

Table 5: Code complexity for the DOR modifications in the RAIDframe.

5. Conclusion

The recovery mechanism of RAID becomes increasingly more critical to the reliability and availability of storage systems. System administrators demand a solution that can reconstruct the entire content of a failed disk as quickly as possible and, at the same time, alleviate performance degradation during recovery as much as possible. However, it is very difficult to achieve these two goals simultaneously.

In this paper, we developed and evaluated a novel dynamic reconstruction optimization algorithm for redun dant disk arrays, called a Popularity-based multithreaded Reconstruction Optimization algorithm (PRO). The PRO algorithm exploits the access locality of I/O workload characteristics, which is ubiquitous in real workloads, especially in the web server environment. The PRO algorithm allows the reconstruction process to rebuild the frequently-accessed areas prior to building infrequently-accessed areas. Our experimental analysis shows that the PRO algorithm results in a 3.6% ~ 23.9% user performance improvements during recovery over the DOR algorithm. Further, PRO leads to a much earlier onset of performance improvement and longer time span of such improvement than DOR during recovery. More importantly, PRO can reduce the reconstruction time of DOR by up to 44.7%. By effectively exploiting access locality, the PRO algorithm accomplishes the goal of simultaneous improvement of reliability and user and system performance.

However, the PRO algorithm in its current form may not be suitable for all types of workloads, for example, it will not likely work well under write-dominated workload. A good solution for write-dominated workload remains one of our directions for future research on PRO. In addition, current PRO only integrates the access locality into the reconstruction algorithm, without distinguishing or predicting access patterns. We believe that PRO's effectiveness can be increased if the access patterns are discovered, predicted, and incorporated into the reconstruction process, which is another direction of our future research.

Acknowledgments

We would like to thank our shepherd, Andrea C. Arpaci-Dusseau, and the anonymous reviewers for their helpful comments in reviewing this paper. We also thank Greg Oster (the RAIDframe maintainer in NetBSD operating system) at the University of Sas-

katchewan for his insightful suggestions, and thank Chao Jin, Xiongzi Ge and Qiang Zou for their help in writing this paper.

This work is sponsored by the National Basic Research Program of China (973 Program) under Grant No. 2004CB318201, the Key Basic Research Project under Grant No.2004CCA07400, Project CNGI-04-5-1D, Program for New Century Excellent Talents in University NCET-04-0693, the China NSF under Grant No.60273074, No.60503059, No. 60303032, No. 60603048, and the US NSF under Grant No. CCF-0621526.

References

[1]. D. Wenk. Is 'Good Enough' Storage Good Enough for Compliance? *Disaster Recovery Journal*. 2004.

[2]. Q. Zhu and Y. Zhou. Chameleon: A Self-Adaptive Energy-Efficient Performance-Aware RAID. In *Proceedings of the fourth annual Austin Conference on Energy-Efficient Design(ACEED 2005)*, Poster Session, Austin, Texas, March 1-3, 2005.

[3]. D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the Conference on Management of Data*, 1988.

[4]. G. Gibson, Redundant Disk Arrays: Reliable, Parallel Secondary Storage. MIT Press, 1992.

[5]. Q. Xin, E. L. Miller, T. J. Schwarz, D. D. E. Long, S. A.Brandt, and W. Litwin. Reliability mechanisms for very large storage systems. In *Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 146–156, April 2003.

[6]. M. Holland, G. Gibson, and D. Siewiorek. Fast, On-Line Failure Recovery in Redundant Disk Arrays. In *Proceedings of the International Symposium on Fault-Tolerant Computing*, pages 422-43, 1993.

[7]. Jack Y.B. Lee and John C.S. Lui. Automatic Recovery from Disk Failure in Continuous-Media Servers. IEEE Transaction On Parallel And Distributed Systems, Vol. 13, No. 5, May 2002.

[8]. M. Holland, G.A. 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, pages 295-335, July 1994.

[9]. R. Hou, J. Menon, and Y. Patt. Balancing I/O Response Time and Disk Rebuild Time in a RAID5 Disk Array. In *Proceedings of the Hawaii International Conference on Systems Sciences*, pages 70-79, 1993.

[10]. Q. Xin, E. L. Miller and T. J. Schwarz. Evaluation of Distributed Recovery in Large-Scale Storage Systems. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, pages 172-181, June 2004.

[11]. M. Holland. On-Line Data Reconstruction in Redundant Disk Arrays. Carnegie Mellon Ph.D. Dissertation CMU-CS-94-164, April 1994.

[12]. R. Muntz and J. Lui, Performance Analysis of Disk Arrays Under Failure. In *Proceedings of the 16th Conference on Very Large Data Bases*, 1990.

[13]. H. H. Kari, H. K. Saikkonen, N. Park and F. Lombardi. Analysis of repair algorithms for mirrored-disk systems. IEEE Transactions on Reliability, Vol 46, No. 2, pages 193-200, 1997.

[14]. H.M Vin, P.J. Shenoy and S. Rao. Efficient Failure Recovery in Multi-Disk Multimedia Servers. In *Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*, pages 12–21, 1995.

[15]. M. Sivathanu, V. Prabhakaran, F. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Improving Storage System Availability with D-GRAID. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST'03)*, San Francisco, CA, March 2003.

[16]. A. Smith. Cache Memories. Computing Surveys, Vol 14, No. 3, pages 473–480, 1982

[17]. M. E. Gomez and V. Santonja. Characterizing Temporal Locality in I/O Workload. In *Proceedings of the 2002 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'02)*. San Diego, USA, July 2002

[18]. L. Cherkasova and M. Gupta. Analysis of Enterprise Media Server Workloads: Access Patterns, Locality, Content Evolution, and Rates of Change. IEEE/ACM Transactions on Networking, Vol, 12, No. 5, October 2004

[19]. L. Cherkasova, G Ciardo. Characterizing Temporal Locality and its Impact on Web Server Performance. Technical Report HPL-2000-82, Hewlett Packard Laboratories, July 2000.

[20]. M. Arlitt and C. Williamson. Web server workload characterization: the search for invariants. In *Proceedings of the ACM SIGMETRICS '96 Conference*, Philadelphia, PA, May 1996.

katchewan for his insightful suggestions, and thank Chao Jin, Xiongzi Ge and Qiang Zou for their help in writing this paper.

This work is sponsored by the National Basic Research Program of China (973 Program) under Grant No. 2004CB318201, the Key Basic Research Project under Grant No.2004CCA07400, Project CNGI-04-5-1D, Program for New Century Excellent Talents in University NCET-04-0693, the China NSF under Grant No.60273074, No.60503059, No. 60303032, No. 60603048, and the US NSF under Grant No. CCF-0621526.

References

[1]. D. Wenk. Is 'Good Enough' Storage Good Enough for Compliance? *Disaster Recovery Journal*. 2004.

[2]. Q. Zhu and Y. Zhou. Chameleon: A Self-Adaptive Energy-Efficient Performance-Aware RAID. In *Proceedings of the fourth annual Austin Conference on Energy-Efficient Design(ACEED 2005)*, Poster Session, Austin, Texas, March 1-3, 2005.

[3]. D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the Conference on Management of Data*, 1988.

[4]. G. Gibson, Redundant Disk Arrays: Reliable, Parallel Secondary Storage. MIT Press, 1992.

[5]. Q. Xin, E. L. Miller, T. J. Schwarz, D. D. E. Long, S. A.Brandt, and W. Litwin. Reliability mechanisms for very large storage systems. In *Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 146–156, April 2003.

[6]. M. Holland, G. Gibson, and D. Siewiorek. Fast, On-Line Failure Recovery in Redundant Disk Arrays. In *Proceedings of the International Symposium on Fault-Tolerant Computing*, pages 422-43, 1993.

[7]. Jack Y.B. Lee and John C.S. Lui. Automatic Recovery from Disk Failure in Continuous-Media Servers. IEEE Transaction On Parallel And Distributed Systems, Vol. 13, No. 5, May 2002.

[8]. M. Holland, G.A. 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, pages 295-335, July 1994.

[9]. R. Hou, J. Menon, and Y. Patt. Balancing I/O Response Time and Disk Rebuild Time in a RAID5 Disk Array. In *Proceedings of the Hawaii International Conference on Systems Sciences*, pages 70-79, 1993.

[10]. Q. Xin, E. L. Miller and T. J. Schwarz. Evaluation of Distributed Recovery in Large-Scale Storage Systems. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, pages 172-181, June 2004.

[11]. M. Holland. On-Line Data Reconstruction in Redundant Disk Arrays. Carnegie Mellon Ph.D. Dissertation CMU-CS-94-164, April 1994.

[12]. R. Muntz and J. Lui, Performance Analysis of Disk Arrays Under Failure. In *Proceedings of the 16th Conference on Very Large Data Bases*, 1990.

[13]. H. H. Kari, H. K. Saikkonen, N. Park and F. Lombardi. Analysis of repair algorithms for mirrored-disk systems. IEEE Transactions on Reliability, Vol 46, No. 2, pages 193-200, 1997.

[14]. H.M Vin, P.J. Shenoy and S. Rao. Efficient Failure Recovery in Multi-Disk Multimedia Servers. In *Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*, pages 12–21, 1995.

[15]. M. Sivathanu, V. Prabhakaran, F. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Improving Storage System Availability with D-GRAID. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST'03)*, San Francisco, CA, March 2003.

[16]. A. Smith. Cache Memories. Computing Surveys, Vol 14, No. 3, pages 473–480, 1982

[17]. M. E. Gomez and V. Santonja. Characterizing Temporal Locality in I/O Workload. In *Proceedings of the 2002 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'02)*. San Diego, USA, July 2002

[18]. L. Cherkasova and M. Gupta. Analysis of Enterprise Media Server Workloads: Access Patterns, Locality, Content Evolution, and Rates of Change. IEEE/ACM Transactions on Networking, Vol, 12, No. 5, October 2004

[19]. L. Cherkasova, G Ciardo. Characterizing Temporal Locality and its Impact on Web Server Performance. Technical Report HPL-2000-82, Hewlett Packard Laboratories, July 2000.

[20]. M. Arlitt and C. Williamson. Web server workload characterization: the search for invariants. In *Proceedings of the ACM SIGMETRICS '96 Conference*, Philadelphia, PA, May 1996.

[21]. D. Roselli, J. R. Lorch, and T. E. Anderson. A Comparison of File System Workloads. In *Proceedings of 2000 USENIX Annual Technical Conference*. San Diego, California, USA, June 2000.

[22]. P. G. Sikalinda, L. Walters and P. S. Kritzinger. A Storage System Workload Analyzer. Technical Report CS06-02-00, University of Cape Town, 2006.

[23]. The NetBSD Foundation. The NetBSD Guide. http://www.netbsd.org/guide/en/chap-rf.html.

[24]. The OpenBSD Foundation. The OpenBSD FAQ. http://www.se.openbsd.org/faq/faq11.html#raid.

[25]. The FreeBSD Foundation. The FreeBSD/i386 5.3-RELEASE Release Notes. http://www.freebsd.org/releases/5.3R/relnotes-i386.html.

[26]. W.V. Courtright II, G.A. Gibson, M. Holland and J. Zelenka. RAIDframe: Rapid Prototyping for Disk Arrays. In *Proceedings of the 1996 ACM SIGMETRICS international Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '96)*, Vol. 24 No. 1, pages 268-269, May 1996.

[27]. The randread pkg. http://pkgsrc.se/benchmarks/randread

[28]. SPC Web Search Engine I/O Trace. http://traces.cs.umass.edu/storage/