RAPID-Cache—A Reliable and Inexpensive Write Cache for High Performance Stroage Systems

Yiming Hu, *Senior Member*, *IEEE*, Tycho Nightingale, and
Qing Yang, *Senior Member*, *IEEE*

Abstract—Modern high performance disk systems make extensive use of nonvolatile RAM (NVRAM) write caches. A single-copy NVRAM cache creates a single point of failure while a dual-copy NVRAM cache is very expensive because of the high cost of NVRAM. This paper presents a new cache architecture called **RAPID-Cache** for *Redundant, Asymmetrically Parallel, and Inexpensive Disk Cache*. A typical RAPID-Cache consists of two redundant write buffers on top of a disk system. One of the buffers is a primary cache made of RAM or NVRAM and the other is a backup cache containing a two-level hierarchy: a small NVRAM buffer on top of a log disk. The small NVRAM buffer combines small write data and writes them into the log disk in large sizes. By exploiting the locality property of I/O accesses and taking advantage of well-known Log-structured File Systems, the backup cache has nearly equivalent write performance as the primary RAM cache. The read performance of the backup cache is not as critical because normal read operations are performed through the primary RAM cache and reads from the backup cache happen only during error recovery periods. The RAPID-Cache presents an asymmetric architecture with a fast-write-fast-read RAM being a primary cache and a fast-write-slow-read NVRAM-disk hierarchy being a backup cache. The asymmetrically parallel architecture and an algorithm that separates actively accessed data from inactive data in the cache virtually eliminate the garbage collection overhead, which are the major problems associated with previous solutions such as Log-structured File Systems and Disk Caching Disk. The asymmetric cache allows cost-effective designs for very large write caches for high-end parallel disk systems that would otherwise have to use dual-copy, costly NVRAM caches. It also makes it possible to implement reliable write caching for low-end disk I/O systems since the RAPID-Cache makes use of inexpensive disks to perform reliable caching. Our analysis and trace-driven simulation results show that the RAPID-Cache has significant reliability/cost advantages over conventional single NVRAM write caches and has great cost advantages over dual-copy NVRAM caches. The RAPID-Cache architecture opens a new dimension for disk system designers to exercise trade-offs among performance, reliability, and cost.

Index Terms—Disks, storage systems, performance, reliability, fault-tolerance.

1 Introduction

Modern disk I/O systems make extensive use of nonvolatile RAM (NVRAM) write caches to asynchronous write [2], [3], [4], i.e., a write request is acknowledged before the write goes to disk. Such write caches significantly reduce response times of disk I/O systems seen by users, particularly in RAID systems. Large write caches can also improve system throughput by taking advantage of both temporal and spatial localities [2], [5], as data may be overwritten several times or combined together before been written to the disk. IO requests are very bursty [6], requests are often come together with long intervals of relative inactive periods in between. Large write caches also benefit from the the burstiness of write workloads since data coming from bursts can be quickly stored in the cache and written back to the disk later when the system is less busy. Treiber and Menon reported that write caches could reduce disk utilization for writes by an order of magnitude when compared to basic RAID-5 systems [3]. However, the use of write caches introduces two problems: poor reliability and high cost.

Disks are impressively reliable today, with a Mean Time To Failure (MTTF) of up to one million hours. Such a low failure rate, coupled with possible redundancy such as RAID, gives a Mean Time To Data Loss (MTTDL) of several hundreds of millions of hours in a typical RAID-5 system [7]. Adding a single cache in front of a disk system creates a single point of failure, which is vulnerable to data loss. Savage and Wilkes pointed out in [7] that because typical NVRAM technology (battery backed RAM) has a quite low MTTF of 15K hours, a single-copy NVRAM cache suffers significantly higher risk of data loss than results from disk failures. To overcome the reliability problem, some high-end RAID systems use dual-copy caches so that a failure in one cache leaves the other cache intact [2]. When a write request comes, the controller writes two copies of the data independently into the two caches, a primary cache and a backup cache. Besides the reliability problem, NVRAM is also known to be very costly [7], [8], [9] so the size of the

- *Y. Hu is with the Department of Electrical & Computer Engineering and Computer Science, University of Cincinnati, Cincinnati, OH 45221-0030. E-mail: yhu@ececs.uc.edu.*
- *T. Nightingale is with Sun Microsystems, 901 San Antonio Road, USUN03-204, Palo Alto, CA 94303. E-mail: tycho.nightingale@sun.com.*
- *Q. Yang is with the Department of Electrical and Computer Engineering, University of Rhode Island, Kingston, RI 02881. E-mail: qyang@ele.uri.edu.*

Manuscript received 4 Dec. 2000; accepted 20 Sept. 2001.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 113249.

NVRAM cache is often limited. For example, a major NVRAM manufacturer quoted the price of NVRAM with embedded lithium-cell batteries for $55/MB in quantity as of December 2000. The cost of disks, on the other hand, is about 0.5 cents/MB, which is a difference of four orders of magnitude. Moreover, the cost difference is widening (the difference was three orders of magnitudes two years ago) because prices of disks are falling very rapidly. For a disk system with a reasonably sized write cache, the NVRAM may dominate the cost of the entire system. For example, in a system with 16 disks (40 GB per disk) and an NVRAM write cache of 256 MB, at $55/MB, the NVRAM costs about $14,080, while the total cost of 16 disks is only $3,200 (assuming each 40-GB disk costs $200). If we use dual-copy caches to ease the reliability problem of the single-copy cache, the cost becomes prohibitively high, particularly for large caches. As a result, it is only suitable for the upper echelon of the market [10].

The standard dual-copy write cache system has a *symmetric structure*, where both the primary write cache and the backup write cache have the same size and the same access characteristics—fast read speed and fast write speed. However, the backup cache does not provide any performance benefit to the system during normal operations. Therefore, it is wasteful to use a backup cache identical to the primary cache. What is needed is only a backup cache that can be written to very quickly while its read operations are not as critical since reads from the backup cache occur only during error-recovering periods.

Based on these observations, we propose a new disk cache architecture called *Redundant, Asymmetrically Parallel, Inexpensive Disk Cache*, or **RAPID-Cache** for short, to provide fault-tolerant caching for disk I/O systems inexpensively. The main idea of the RAPID-Cache is to use a conventional, fast-write-fast-read *primary cache* and a nonvolatile, fast-write-slow-read *backup cache*. The primary cache is made of normal NVRAM or DRAM, while the backup cache consists of a small NVRAM cache and a log disk (cache disk). In the backup cache, small and random writes are first buffered in the small NVRAM buffer to form large logs that are written into the *cache disk* later in large transfers, similar to log structured file systems [11], [12], [13], [14]. Because large writes eliminate many expensive small writes, the buffer is quickly made available for additional requests so that the two-level cache appears to the host as a large NVRAM. As a result, the backup cache can achieve the same write speed as the primary cache. The slow-read performance of the backup cache does not affect the system performance since every data block in the backup cache has a copy in the primary cache which can be read at the speed of RAM. The dual cache system here is asymmetric since the primary cache and the backup cache have different sizes and structures. The reliability of the RAPID-Cache is expected to be high since, disk is very reliable. The system is also inexpensive because the NVRAM in the backup cache can be very small, ranging from hundreds of KB to several MB and the cost of the disk space is significantly less than that of a large NVRAM. We will show that RAPID-Caches provide much higher reliability compared to single-copy NVRAM caches and much lower cost compared to dual-copy NVRAM caches, without sacrificing performance. On the other hand, because of its low cost, with the *same budget*, RAPID-Caches can have significantly higher performance compared to conventional NVRAM cache architectures by affording much larger primary cache sizes, while still maintaining good reliability.

While the idea of RAPID-Cache can be used in any I/O system, it is particularly suitable for parallel disk systems such as RAID because RAID systems are most likely to be used in environments which require high performance and high reliability. Therefore, we concentrate our study on RAPID-Caches on top of RAID-5 systems in this paper. We have carried out trace-driven simulation experiments as well as analytical studies to evaluate the performance and reliability of the RAPID-Cache. Using real-world traces as well as synthetic traces generated based on realistic workloads [6], [15], we analyze the performance of the RAPID-Cache architecture and compare it with existing disk cache architectures. Numerical results show that the RAPID-Cache has significant performance/cost and reliability advantages over the existing architectures.

The paper is organized as follows: The next section presents the detailed architecture and operations of the RAPID-Cache. Section 3 presents our experimental methodology. Simulation results will be presented in Section 4, followed by an approximate reliability and cost analysis in Section 5. We discuss related work in Section 6 and conclude the paper in Section 7.

2 Architecture and Operations

Fig. 1 shows the basic structure of a RAPID-Cache. It consists of a conventional primary RAM cache and a backup cache. The backup cache is a two-level hierarchy with a small NVRAM on top of a cache disk, similar to DCD [16]. In a RAPID-Cache, every I/O write operation is sent to both the primary cache and the backup cache while read operations are performed using the primary cache only.

For very high overall reliability, the primary cache can be NVRAM to provide redundant protection during a power failure. On the other hand, for low cost systems, the primary cache can be DRAM. During normal operations, the DRAM primary cache and the backup cache contain redundant data. If any one of the two caches fails, data can be reconstructed from the other. During a power failure, data are retained in the backup NVRAM and the cache disk. If both the read cache and the primary write cache are made of DRAM, we can use a unified read/write cache structure, as shown in Fig. 2a, for better cache utilization. A RAPID-Cache with a large unified DRAM primary cache has higher throughput, lower cost, and better reliability than that of a single-copy conventional NVRAM cache. For many applications that require redundant protection during a power failure, a *Triple RAPID-Cache* (shown in Fig. 2b) can be used to build a highly reliable, very large cache system. The idea is to use two low-cost backup caches to support one large primary DRAM cache. During normal operations, the primary cache and the two backup caches provide triple redundancy protection. The two backup caches provide dual-redundancy protection during a power failure. Triple RAPID-Caches are especially suitable for high-end systems

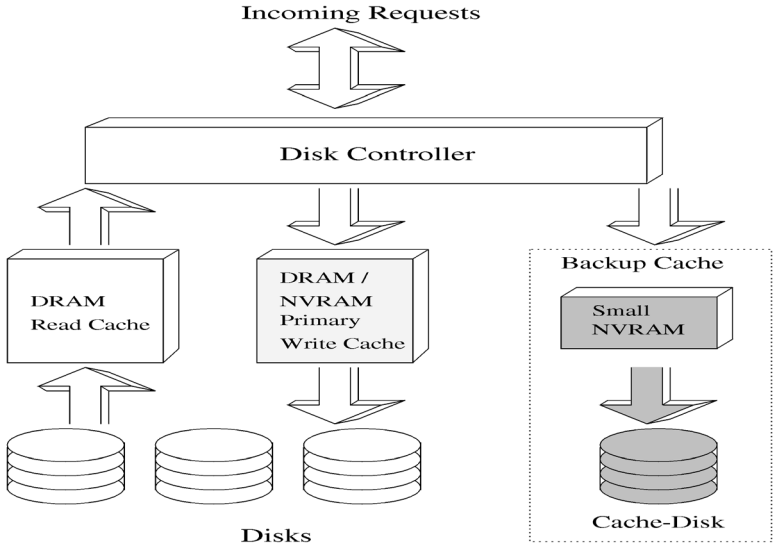

Fig. 1. RAPID-Cache on top of a disk system.

that need very large and very reliable write caches. For large cache sizes, a Triple RAPID-Cache has lower cost and better reliability than both a RAPID-Cache with a large NVRAM primary cache and a conventional dual-copy NVRAM cache.

2.1 Structures of the Backup Cache

Fig. 3 shows the detailed structures of the backup NVRAM cache and the cache disk. The NVRAM cache consists of an *LRU Cache*, two to four *Segment Buffers*, and a *Hash Table*. Another related data structure, called the *Disk Segment Table,* is located in a DRAM buffer.

The actively-accessed data in the backup cache reside in the LRU cache. The less actively-accessed data are kept in the cache disk. Data in the cache disk are organized in the format of *Segments* similar to that in a Log-structured File System such as the Sprite LFS and the BSD LFS [12], [13]. A segment contains a number of *slots* each of which can hold one data block. Data blocks stored in segments are addressed by their *Segment IDs* and *Slot IDs*. Data blocks stored in the LRU cache are addressed by their *Logical Block Addresses (LBAs)*. The Hash Table contains location information for each of the valid data blocks in the backup cache. It describes whether a block is in the NVRAM LRU cache or in the cache disk, as well as the data address in the LRU cache or the cache disk. In our current design, the size of the hash entry is 16 bytes. Since data in the backup cache is the exact image of the data in the primary write cache, the total number of valid data blocks in the backup cache is the same as in the primary write cache, regardless of the sizes of the backup NVRAM and the cache disk. If the data block size is 8 KB, then, for a 32 MB write cache, there are 4,096 blocks in total. Since each valid block has a corresponding hash entry of 16 bytes, the total hash table size is 64 KB, which is compact enough to be placed in the NVRAM.

For the purpose of speeding up garbage collection, we also keep track of a data structure called the *Disk Segment Table*. Since this table contains redundant information that can be quickly and completely reconstructed from the Hash Table in case of a crash, it is stored in the DRAM. We will

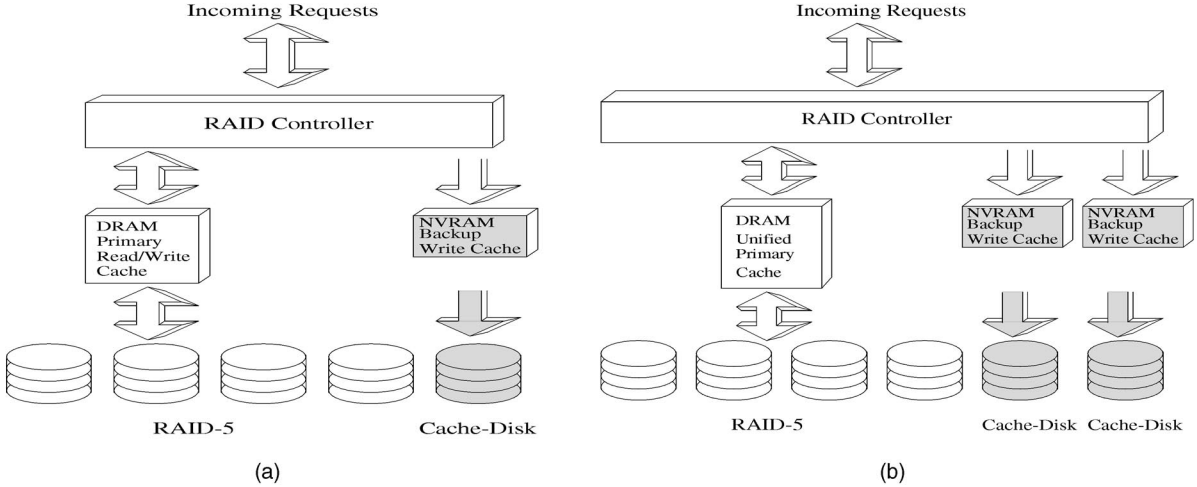

Fig. 2. Unified RAPID-Cache and triple RAPID-Cache. (a) A RAPID-Cache with a unified read/write cache. (b) A triple RAPID-Cache with two cache

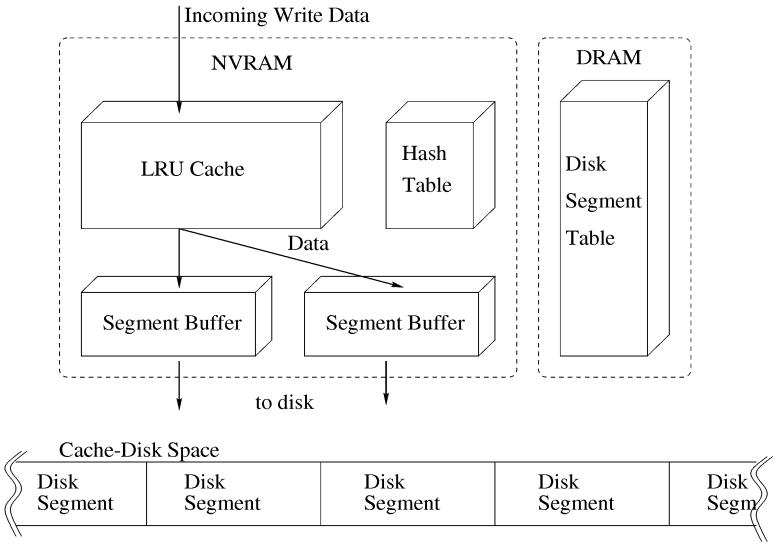

Fig. 3. The detailed structure of the backup cache and the cache disk.

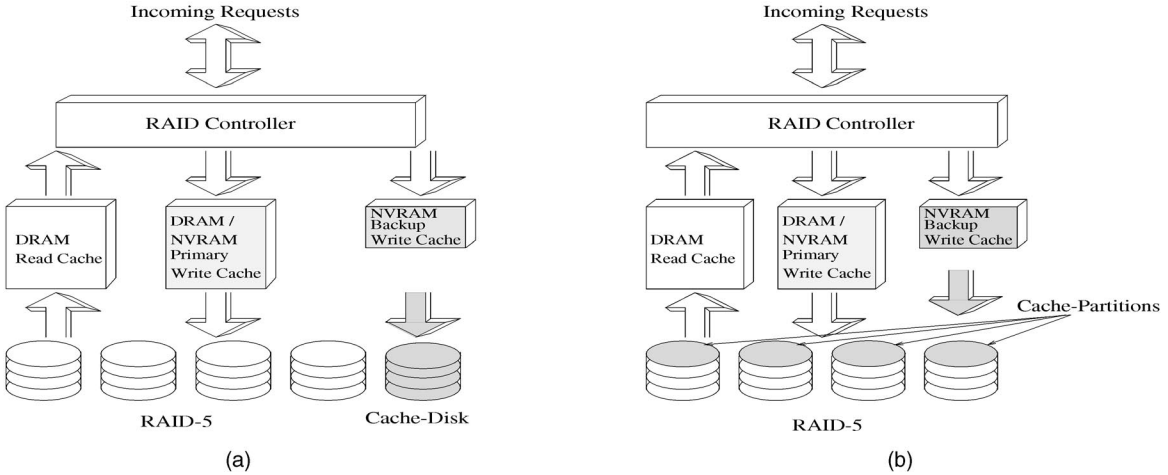

Fig. 4. Physical RAPID-Cache and logical RAPID-Cache. (a) A RAPID-Cache with a dedicated physical cache disk. (b) A RAPID-Cache with a distributed logical cache disk.

discuss the structure of the Disk Segment Table in detail later in this section.

The cache disk in the backup cache can be a *dedicated physical disk*, as shown in Fig. 4a. It can also be distributed among the data disks of the RAID system, each data disk having a small partition acting as a part of a large *distributed logical cache disk*, as shown in Fig. 4b. In modern RAID systems, the physical cache disk can often be implemented without extra cost since many modern RAID systems include one or several spare disks that can be put into service when an active disk fails [17]. However, as pointed out by Wilkes et al. [17], during normal operations, the spare disks are not used in many systems[1] and contribute nothing to the performance of the system. It is also hard to tell if the spare disks are still working since they are not in use. Such a spare disk can therefore be used as a physical cache disk of a RAPID-Cache. A secondary benefit here is that now we are aware of whether the spare disk is in working condition or not, and we are able to replace the failed one before it is too late. When a spare disk becomes an active disk to replace a failed one, the RAPID-Cache can be degraded to the logical cache disk mode by using a partition residing on the spare disk until the failed disk is replaced and a new spare disk is put into the system. In the case of a logical cache disk, the data written into the logical cache partitions on the RAID disks do not involve in parity operations. In other words, the logical cache partitions act as "Just a Bunch of Logical Disks." The backup cache provides a reliable, nonvolatile backup of the cached data.

1. In a system using *distributed sparing*, the spare disk is utilized during normal operation.

2.2 Write

When a write request comes, the controller first invalidates any data copy in the read cache. It then sends the data simultaneously to the primary cache and the LRU cache of the backup cache. If there is space available in the caches, the data are then copied to the caches immediately. A hash entry in the backup cache is also created to indicate that the data block is located in the backup LRU cache. Once the data is written into both the primary cache and the NVRAM buffer of the backup cache, the controller sends an

acknowledgment to the host signaling that the request is complete.

If there is no space left in the primary cache, the controller first tries to discard a clean block from the cache to make room for the new request. However, if it cannot find a clean block, the controller chooses the Least-Recently-Used (LRU) data block and writes it to the RAID. When the LRU block is safely written into the RAID, the space in the primary cache is freed for the incoming request. Meanwhile, the copy of the replaced data in the secondary cache, whether in the LRU cache or in the cache disk, is also invalidated.

If the LRU cache in the backup cache is full, the RAPID controller picks an empty segment buffer and sets it as the "current" segment buffer. An LRU data block is then copied to the segment buffer and the corresponding entries in the Hash Table and the Disk Segment Table are modified to reflect the fact that the data block is now in the current segment buffer instead of the LRU cache. Since the segment buffer is also in the NVRAM, the cache space used by the LRU data block can now be safely freed to accept the incoming request. The following write requests may continue to evict LRU blocks to the segment buffer until the segment buffer is full. The controller then writes the contents of the segment buffer into a cache disk segment in one large write. At this point, the controller switches to another empty segment buffer as the current segment buffer and continues operation. Since the entire segment buffer is written in one large write instead of many small writes, the segment buffer is very quickly made available again when the write finishes. Therefore, the small NVRAM cache and the large cache disk appear to the controller as a large NVRAM write cache.

The dedicated segment buffers allow the data to be transferred to the cache disk in a single large and continuous transfer. If the I/O systems can support scatter/gather I/O transferring (a hardware technique to assemble data from noncontiguous memory locations), then the dedicated segment buffers are not needed.

The segment buffer size directly affects the write efficiency. For a RAPID-Cache with a dedicated cache disk, the larger the segment size, the smaller the write overheads caused by disk seeking and rotational latencies. On the other hand, a larger segment size results in a smaller LRU cache size for a given NVRAM size. Therefore, there is a trade-off between large segment sizes and large backup LRU cache sizes. During our simulation experiments, we found that for the workload we used, two to four 256 KB segment buffers give the best overall performance. For a RAPID-Cache with a logical cache disk, the segment size cannot be too large because segment writes must compete with normal RAID reads in data disks. Large segment sizes may result in lower read performance. We found that using four 128 KB segment buffers can achieve good performance in this case.

2.3 Read

Reading is straightforward in RAPID-Cache. When a request comes, the read cache and the primary write cache are searched. If there is a cache hit, data can be returned immediately. In case of a cache miss, the Least Recently Used (LRU) block in the read cache is discarded and its buffer space is freed. The requested data is then read from the RAID system into the freed LRU block before the data is returned. The backup cache is not involved in read operations.

2.4 Destage

In a traditional RAID system with an NVRAM write cache, dirty data in the write cache is written into the RAID system in a process called *destage* [5], which normally happens in the background. A RAID system with a RAPID-Cache also requires destaging. In our current design, one or several *destaging threads* are initiated when the controller detects an idle period, or when the number of dirty blocks in the primary write cache exceeds a high water-mark, say 70 percent of the cache capacity. The destaging threads find a dirty LRU block in the primary cache, read the old data and parity of that block from disks or the read cache, compute the new parity, and write the new data and parity to disks. After the new data and parity are written, the dirty block in the primary cache is marked as "clean," and the same data block in the backup cache, whether it is in the NVRAM LRU cache or in the cache disk, is invalidated. The invalidation of the backup cache block involves releasing the LRU buffer if the block is in the NVRAM LRU cache, marking the corresponding segment slot as "invalid" if the data is in a disk segment or a segment buffer, and deleting the hash entry from the hash table. The destaging threads run continuously until the idle period is over, or until the dirty block count in the primary cache falls below a low water-mark, say 30 percent of the cache capacity.

Notice that data in the backup cache are never read or written during a destaging process. Therefore, the slow-speed of the cache disk will not affect the destaging performance.

2.5 Garbage Collection

We have shown that in a RAPID-Cache, data in the cache disk are organized in segments, similar to an LFS system. In an LFS system, after the system is running for a while, many disk segments become only partially full because of data overwrites and invalidations. As a result, LFS must frequently call the *garbage collector* that reads several partially full disk segments into RAM, compacts the data, and writes the data back to the disk in a new segment. As mentioned previously, such garbage collection can cause great performance loss in some cases.

In a RAPID-Cache system, segments in the cache disk may also become fragmented and require garbage collection. However, because of the asymmetrically parallel architecture of the RAPID-Cache, all data in the cache disk are also in the primary write cache which can be read quickly. There is no need to read data from the cache disk. To do garbage collection, the RAPID controller simply searches the Disk Segment Table to find several fragmented segments. It then copies the corresponding data from the primary cache to a segment buffer in RAM. Finally, the controller writes the whole contents of the segment buffer to a new disk segment and invalidates the old segments. The

garbage collection overhead of a RAPID-Cache is only a small fraction of that of LFS.

To perform garbage collection, the controller must be able to to quickly identify which disk segments contain valid data blocks. It also must be able to quickly find the Logical Block Address (LBA) of a data block cached in a disk segment slot when given its Segment ID and Slot ID. The *Disk Segment Table* has an entry for each segment in the cache disk. Each entry contains a counter of valid blocks cached in the segment, a flag indicating if the segment is cached in a segment buffer or not, a lock for concurrency control, and a *Slot Mapping Table* that describes which slots in the segment contain valid data. The Slot Mapping Table is an array of integers. Each slot in the segment has an entry (an integer) in the table. If the slot does not contain a valid data block, its entry in the Slot Mapping Table is set to -1. Otherwise, the entry is set to the LBA of the cached data block. Since each slot in the cache disk has an entry in its Slot Mapping Table, the total size of the Disk Segment Table is mainly determined by the number of slots in the cache disk. If the cache disk size is 256 MB and the cache block size is 8 KB, then there are 32K slots requiring 32K integers (128 KB) in the Disk Segment Table. The information in the Disk Segment Table can be quickly and completely reconstructed from the Hash Table in case of a crash.

If garbage collection is needed (when the number of available empty disk segments falls below a threshold), the garbage collector is activated. The collector searches the Disk Segment Table for a set of disk segments with the smallest valid block counters (which have the maximum amount of garbage). The collector then reads these segments into RAM in large I/O requests, finds remaining valid blocks (using the slot mapping table), and merges them to form a new segment. The new segment is then written back to the disk. The old disk segments can now be marked as blank.

In addition to the low-cost garbage collection algorithm, in simulation experiments, we found that, for the workload we used, the RAPID controller almost never had to call the garbage collector, meaning that the garbage collection overhead has virtually no impact on the overall system performance. This is due to the following two reasons: First, because disk spaces are so inexpensive now (about 0.5 cents per MB as of this writing), we normally choose a cache space that is 5-10 times larger than the primary write cache size. For example, for a primary cache size of 32 MB, we can use a disk space of 160 MB as the cache space,[2] which costs only about \$0.8. Since the 32 MB of data are spread over a space of 160 MB, much of the disk space is empty. Second, unlike an LFS system which writes both actively-accessed data and inactive data into a segment, in a RAPID-Cache system, active data and inactive data are separated. Most active data are kept in the LRU cache, while data in the disk segments of the cache disk are relatively inactive. Therefore, entire segments are often invalidated because the background destage constantly threads destage inactive data to disk arrays. As a result, most of time, the controller can find an empty disk segment without the need for garbage collection.

2. Almost all modern disks have a minimal capacity of 20 GB or more as of this writing. Therefore, for a physical cache disk we may be able to use a quite large cache-space. As addressed before, a physical cache disk may not introduce extra cost to the system. For a logical cache disk, however, we may want to use a smaller cache-space size than the one used here, so we can have more disk space for data disks.

2.6 Error Handling and Availability

A RAPID-Cache system has excellent reliability because of the data redundancy provided by the primary cache and the backup cache. If data in any one of the caches is lost for any reason, the other cache is read to rebuild the data. During a system crash or a power failure, data is retained in the NVRAM or the cache disk of the backup cache. If the primary cache is also made of NVRAM, it can provide additional protection. It takes only several seconds (tens of seconds at most) to recover all the data from the backup cache because reading from the cache disk is done in the large segment size thus is very efficient.

In fact, during a power failure period, data cached in the cache disk is much safer than in an NVRAM. Disks can retain their data for a long period of time without doing anything. On the other hand, data stored in active devices such as NVRAM or UPS (Uninterrupted Power Supply) backed DRAM are not as safe as data on disks because NVRAM batteries may leak and UPS may run out of power or fail.

Compared to a single NVRAM write cache, a RAPID-Cache system has excellent availability. If one cache partition of a logical cache disk crashes, the whole system can operate continuously since either a spare disk will swap in to replace the failed disk, or the controller can simply skip the failed disk without affecting the system performance significantly. If a dedicated cache disk crashed, the system can borrow a small partition from each data disk and operate in a logical cache disk mode. If the NVRAM of the backup cache fails, a small portion of the primary cache can be borrowed so the system can continue its operations until the failed NVRAM is replaced. If the primary write cache fails, the read cache can be switched to a unified read/write cache mode to accept write data. In the case where a RAPID-Cache system uses a unified read/write primary cache, if the entire unified primary cache fails, the system may still operate in a degraded mode with lower performance because of the slow read speed of the backup cache.

3 Simulation Models

We use trace-driven simulations to evaluate the effectiveness of RAPID-Cache. In this section, we describe the details of our simulation and workload models.

3.1 The Simulators

The RAPID-Cache simulator is built on top of a general cached-RAID5 simulator developed by us. The RAID mapping function is borrowed from the Berkeley *raidsim* simulator. The disk model used in our simulator is developed by Kotz et al. [18] that models an HP 97560 disk drive described in [19]. HP 97560 is a 5.25-inch, 1.26 GB disk with an average access time of 23 ms for an 8 KB data block. The disk simulator provides detailed simulation, including SCSI bus contention, built-in cache read-ahead and write-behind, head-skewing, etc. The simulator is quite accurate and is

used by several other large scale simulation systems such as Stanford SimOS and Dartmouth STARFISH. However, HP 97560 is slightly out-dated. We have made the following changes to make it closer to the performance ranges of current disks: increasing the rotation speed from 4,002 rpm to 7,200 rpm, increasing its capacity by increasing the average linear density from 72 sectors/track to 288 sectors/track, increasing the interface bus speed from 10 MB/sec to 40 MB/sec; and decreasing its platter number from 10 to 3. We also assume that the RAID controller has a high-speed 80 MB/sec fibre-channel bus connected to the host. Requests must reserve the bus before starting data transfer. The bus speed also limit, the response time of NVRAM/RAM cache hits since data hit in the RAM cache also need to be read through the bus. The controller can handle up to 32 pending requests. Additional requests must wait in a FIFO queue. Requests are processed in a First-Come-First-Serve basis, but they may complete out-of-order. The cache block size is 8 KB. The cache is fully associative and uses a LRU algorithm, which is easy and efficient to implememnt with software for such a relatively small cache. For the RAID-5 simulator, the number of RAID-5 columns is set to be 8 or 16 disks and the number of RAID-5 rows to be 1. The stripe unit is 32 KB and the data layout is Left-Symmetric. All results were obtained with the garbage collector running.

3.2 Workload Models

The purpose of our performance evaluation is to show that the RAPID-Cache can deliver the same performance as very expensive NVRAM caches under various workload environments. In order to provide a fair and unbiased evaluation, we paid special attention in selecting the workload that drives our simulators since it plays a critical role in performance evaluation. Our main objective in choosing the workload models is to make it as close to realistic workloads as possible and to cover as wide a range of parameters as possible. With this objective in mind, two sets of trace files have been selected as discussed below.

3.2.1 Real-World Traces

The first set of traces are real-world traces obtained from EMC Corporation and HP Laboratories. The EMC trace (referred to as *EMC-tel* trace hereafter) was collected by an EMC Symmetrix disk array system installed at a telecommunication customer site. The trace file contains more than 200,000 requests, with a fixed request size of 2 KB. The trace is write-dominated with a write ratio of 89 percent. The average request rate is about 333 requests/second.

The HP traces were collected from HP-UX systems during a 4-month period and are described in detail in [6]. The trace is called *cello-news*. It is a single disk holding the Usenet news database. The news database was updated constantly throughout the day. The trace has been used by Savage and Wilkes to evaluate their AFRAID RAID system [7]. We have chosen 10 days of traces starting from May 1, 1992. Each day has several hundreds of thousand requests.

Careful examination of the HP trace files reveals that I/O requests are very bursty [6]. The request rate in each request burst is very high while there usually exists a very long idle period (up to 30 seconds) between two consecutive bursts of requests. As a result, the average request rate is very low at about several requests per second. With such a low request rate and highly bursty request pattern, the RAPID-Cache will obviously perform very well since it will have enough time to move all data in the NVRAM buffer collected in a burst into the cache disk and to do destaging and garbage collection during an idle period. In order to present a conservative evaluation for the RAPID-Cache, we artificially reduce the idle period to increase the average request rate. We searched the traces and shortened any idle period longer than 50 milliseconds to 50 milliseconds to make the average request rate about 40 requests/second. To further increase the I/O rate, we also overlaid several days of *cello-news* traces into a single trace. The same approach has been used by Varma and Jacobson in [5] in which up to six days of *cello* traces were overlaid to study the performance of RAID-5 caches. In this study, we overlaid up to 10 days of traces, giving rise to a request rate of 400 requests/second. The numbers of requests in the resulting traces vary from about 200,000 requests to about one million. The request size is about 8 KB.

3.2.2 Synthetic Traces

While real-world traces give a realistic evaluation of the performance of the systems, they have a limited view of system performance considering the fast changing computer world [20]. In order to observe how the RAPID-Cache performs under a variety of workloads, we generated a set of synthetic traces. Our synthetic traces were generated based on I/O access characteristics of the *cello-news* traces and the traces presented by Zivkov and Smith [15]. We carefully fine-tuned the trace generation parameters such as *request interval times*, *data access patterns*, *working-set sizes*, and *read/write ratios* in such a way that the characteristics of generated traces are similar to these real-world traces. Furthermore, in order to provide a fair and comprehensive evaluation, we also vary the workloads over a wide spectrum to cover as many as possible of all possible workload situations.

The *request interval time* in the traces is modeled using exponentially distributed random variables. We chose the exponential distribution function after our analyses on *cello-news* and other traces from HP Laboratories. An I/O request may come in a burst which is called a "bursty" request or outside of a burst which is called a "background" request. The generator repeatedly inserts clusters of bursty requests into the "background" requests. The mean request rate in a burst is 10 times larger than the mean request rate of background requests, though both have exponentially distributed interval times. The burst length and the interval length between two consecutive bursts in terms of the number of requests are exponentially distributed based on the analysis of real-world traces, with the mean being λ and μ, respectively.

The *data access pattern* is modeled using an approach similar to the one used by Varma and Jacobson in [5]. Two separate history tables are maintained, one for writes, and the other for reads. A request may be selected either from a history table or selected uniformly among all possible disk addresses. If the request is to be selected from a history table, an entry in the table is selected using a random variable representing the distance of the entry from the top of the table. While our analysis of the real-world traces

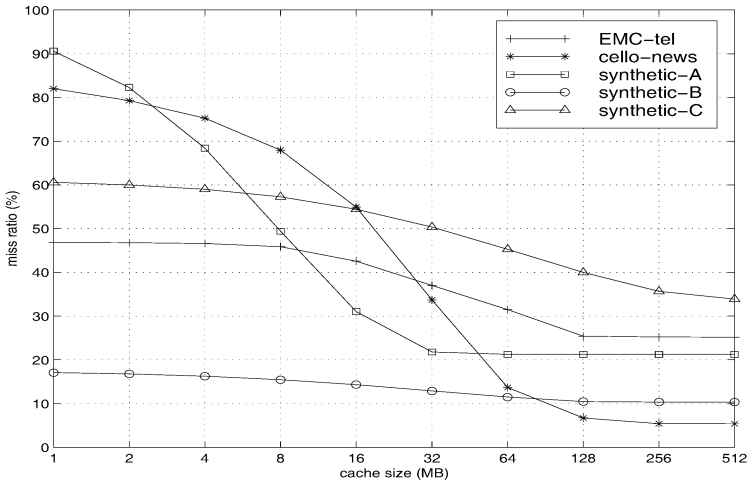

Fig. 5. Miss ratios of traces with various cache sizes.

shows that this distance can be approximated by an exponential distribution most of the time, we intentionally chose three different distribution functions, namely exponential distribution, normal distribution, and uniform distribution, to generate traces with different access patterns. The history tables are in the form of LRU stacks to simulate the temporal locality of I/O accesses. When an entry is selected from a table, it is removed from the stack and pushed back on top of the stack.

The *working-set size* is mainly controlled by the sizes of the history tables. Larger history tables result in larger working-sets. In [15], Zivkov and Smith performed extensive research on disk caching in large databases and timesharing systems. They found that traces from different systems show significantly different characteristics. The working-set sizes of our synthetic traces are modeled after the *cello-news* traces and traces studied by Zivkov and Smith. Fig. 5 shows the cache miss ratios as a function of cache sizes for *EMC-tel*, *cello-news*, and our three synthetic traces on an LRU cache with an 8 KB block size. *Synthetic-A* trace is modeled after *cello-news* with a slightly higher initial miss ratio and a smaller working-set of 32 MB. *Synthetic-B* is modeled after the "telecom" trace in [15], which has a very low initial miss ratio of 17 percent (hence, a very small first working-set). Its miss ratio decreases very slowly when the cache size increases. *EMC-tel* also has a low initial miss ratio. Finally, *Synthetic-C* is modeled after the "bank" trace in [15] with a moderate initial miss ratio of 60 percent and a very large working-set of 512 MB. Table 1 lists the characteristics of all five traces used in this study. The request sizes of these traces are 8 KB except for *EMC-tel*, which has a request size of 2 KB.

We have generated 112 different trace files with the mean I/O request rates ranging from 100 to 8,000 requests/second. Each trace file contains at least 200,000 requests. To verify whether 200,000 requests are sufficient to generate reasonably accurate results in each simulation run, we selected several trace files and increased their request numbers to over one million. We then ran simulations under several different configurations using these long traces. We found that the results of a trace with 200,000 requests are within 3-10 percent of the same trace with one million requests. As a result of the cold-start effect, the average response times of 200,000 requests are always biased 3-10 percent slower than those of one million requests. However, the *relative performance* of different configurations using the same trace length does not change. For example, a RAPID-Cache and a standard dual-copy cache have almost identical performance for traces with 200,000 requests. The two still have almost identical performance for traces with one million requests, although their average response times are slightly faster with the longer trace. Given the time limit, we ran most our simulations using 200,000 requests.

TABLE 1
Characteristics of Traces

Trace Name	Read Ratio	Access Pattern	Working set(MB)	Burstiness of synthetic traces	Note
EMC-tel	11%	N/A	128	N/A	Write-dominated
Cello-news	40%	N/A	128	N/A	Write-dominated, very bursty
Synthetic-A	40%	normal	32	$\lambda = 32$, $\mu = 1024$	Write-dominated, moderate working-set
Synthetic-B	40%	uniform	1	$\lambda = 128$, $\mu = 1024$	Write-dominated, very small working-set
Synthetic-C	80%	exponential	512	$\lambda = 64$, $\mu = 1024$	Read-dominated, very large working-set

"Access Pattern" refers to the random function that controls the distance from the top of history table to a selected entry in the table. λ is the mean burst length, while μ is the mean interval length; both are in terms of the number of requests.

TABLE 2
Performance of Trace *EMC-tel*

Cache sizes Read/Write	Response Time (ms)											
	Split R/W Cache						Unified R/W Cache					
	baseline		P-RAPID		L-RAPID		baseline		P-RAPID		L-RAPID	
(MB)	read	write	read	write	read	write	read	write	read	write	read	write
8/4 (12)	3.44	0.15	3.44	0.16	3.55	0.16	3.03	0.15	3.03	0.16	3.16	0.16
8/8 (16)	2.71	0.15	2.71	0.16	2.84	0.16	2.56	0.15	2.56	0.16	2.69	0.16
8/16 (24)	1.75	0.15	1.75	0.16	1.87	0.16	1.96	0.15	1.96	0.16	2.08	0.16

The size of the NVRAM buffer of the RAPID-Cache is 1 MB. L-RAPID means Logical-RAPID-Cache and P-RAPID means Physical-RAPID-Cache.

4 Simulation Results

Our objective here is to show that the RAPID-Cache has the same or similar performance as the conventional cache, with much lower cost and much higher reliability. On the other hand, with the same budget, the RAPID-Cache architecture allows a much larger primary cache because of its low cost to obtain much higher performance compared to a conventional cache. For this purpose, we define a baseline system as a conventional read/write cache system with an optional backup NVRAM cache that is of the same size as the primary write cache. The backup NVRAM size of RAPID-Caches is chosen to be 2 MB. We found through our experiments that using a size larger than 2 MB does not result in significant performance improvement and using a size smaller than 1 MB may cause performance degradation for some traces. Section 4.2.1 discusses the impact of the backup cache size in more detail. The size of cache disks of RAPID-Caches is set to 256 MB for all simulation runs. This number was chosen because it is large enough to avoid garbage collection most of time.

4.1 I/O Performance

In this section, we will compare the performance of RAPID-Caches with those of conventional caches in terms of throughput and response time using the simulation results. It should be emphasized, that for the same primary cache size, a RAPID-Cache system will not perform better than a conventional cache because the system performance is mainly limited by the primary cache size. Rather, we will show that the RAPID-Cache has the same or similar performance as the conventional cache with much lower cost and much higher reliability. On the other hand, with the same budget, the RAPID-Cache architecture allows a much larger primary cache because of its low cost to obtain much higher performance compared to a conventional cache.

4.1.1 Performance of Real-World Traces

Because the real-world traces (*EMC-tel* and *cello-news*) have only a moderate load, we used a relatively small system for simulation. The number of disks in the RAID-5 system is set to 8. The read cache size is 8 MB for the *EMC-tel* trace and 32 MB for the *cello-news* trace. We varied the primary write cache sizes from 4 to 16 MB for the *EMC-tel* trace and 16 to 32 MB for the *cello-news* trace.

Table 2 lists the read and write response time of the baseline systems and RAPID caches under the *EMC-tel* workload. Because of the use of immediate report and the low cache miss ratio of this trace, the average response times are quite low, especially for write requests. The table shows that the baseline system and the RAPID-Caches have very similar or almost identical performance.

Fig. 6 compares the performance of RAPID-Caches with those of conventional caches under the *Cello-news* trace. In the figure, the average I/O response times (of both read and

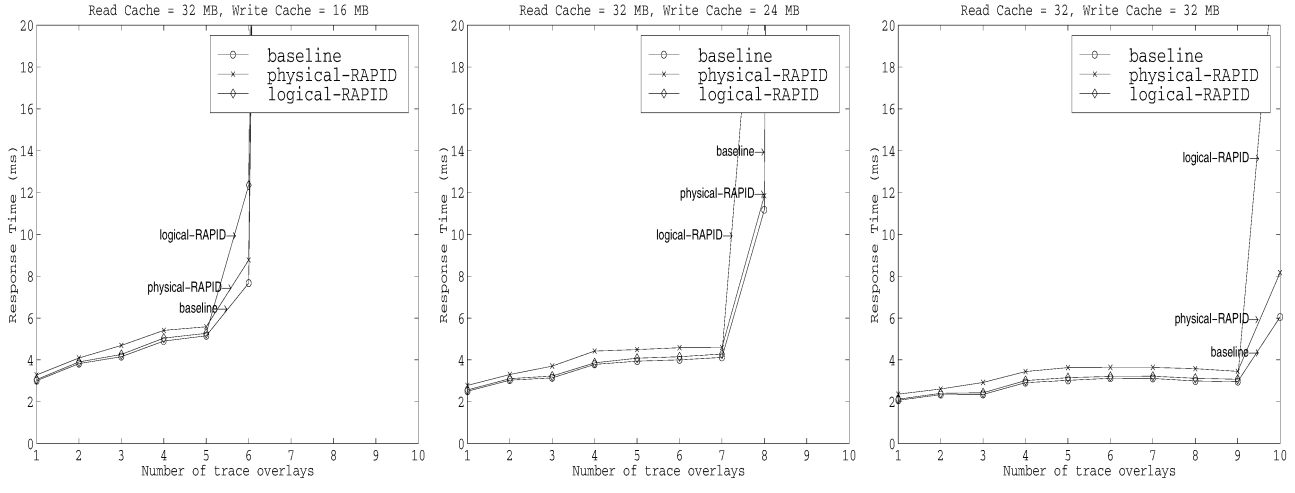

Fig. 6. Performance of trace *cello-news. Note: A single trace has a request rate of about 40 requests/sec. Overlaying 10 traces results in a request rate of about 400 requests/sec.*

write requests) are plotted as functions of numbers of overlaid traces. Because of the use of large read caches and the fast write technique, the average response times of all systems are very low at low throughput. The average response times steadily increase as the number of overlaid traces (hence, the I/O request rate) goes high because of the increased disk traffic. Eventually, the I/O request rate increases to a point where the system is saturated and the response times increase sharply. At this point the system cannot handle any higher workload. We define the throughput at this point as the *maximum system throughput*.

It is clear from the results that the performance of RAPID-Caches is very close to that of baseline systems in terms of maximum system throughput and response time most of time. This confirms that the backup cache of RAPID-Caches performs very well. The small NVRAM and the cache disk do achieve the similar write performance as that of the large primary write cache.

For a Logical-RAPID-Cache, log writes onto disks have to compete with normal data reads and destages, which may cause performance degradation. However it is interesting to note that the Logical-RAPID-Cache performs quite well most of the time. The reason is that the logical cache disk space is distributed among many data disks; therefore, the traffic caused by log writing, seen by each individual data disk is relatively low. In addition, we can use large read caches in our systems because DRAM is inexpensive now. The large read caches significantly reduce the disk read traffic seen by the data disks, thus further reducing the possibility of bandwidth conflicts.

For *EMC-tel*, a Physical-RAPID-Cache always performs better than a Logical-RAPID-Cache. For *cello-news*, a Physical-RAPID-Cache performs better than a Logical-RAPID-Cache at the high workload near the saturation point. However, for this particular trace, the Physical-RAPID-Cache shows slightly higher response times than those of the Logical-RAPID-Cache and the baseline system at low workloads. This can be attributed to the extreme burstiness of the *cello-news* traces as well as the artificially reduced interburst periods. Since we have artificially reduced the length of all long interburst idle periods to 50 ms to stress the RAPID-Cache system, many large bursts arrive closely, overflowing the small (2 MB) backup NVRAM in the RAPID-Cache. As a result, the contents of the backup NVRAM in the RAPID-Cache have to be log-written into the cache disk several times during a large burst. This may create a waiting queue in front of the physical cache disk, resulting in a slightly increased write response time therefore a slightly increased average response time. For Logical-RAPID-Caches, the log-write traffic is distributed among multiple disks, so the queuing effect of log writing, is negligible when the workload is low.

We believe that the slightly increased write response time of Physical-RAPID-Caches under such a highly bursty workload is not a major performance concern. The main performance metric here is throughput. As long as the cache disk has sufficient bandwidth to keep up with the write traffic in the long run (we will discuss the performance of the cache disk later in this section), the throughput of the entire system in the equilibrium state is limited by the primary cache, not the backup cache. Moreover, because of its low cost, with the *same budget* a RAPID-Cache can use a much larger primary cache to achieve a much higher throughput and lower response time than a baseline system. For example, as shown in Fig. 6, for the same backup cache configuration, increasing the size of primary write cache of the RAPID-Cache from 16 MB to 32 MB almost doubles its throughput and reduces the response time.

4.1.2 Performance of Synthetic Traces

Since the synthetic traces have higher I/O rates, we scaled up the systems by increasing the number of disks in the RAID-5 systems from 8 to 16 and the read cache size from 32 MB to 64 MB. Figs. 7, 8, and 9 compare the performance of RAPID-Caches with those of conventional caches. In each of these figures, the top three subplots show the average I/O response times plotted as functions of I/O request rates (throughput).

It is clear from the simulation results that the maximum throughputs of Physical-RAPID-Caches are almost identical to that of baseline systems. Logical-RAPID-Caches also perform very well. Their performance is similar or close to that of baseline systems.

The response-time versus throughput curves of RAPID-Caches and baseline systems are almost completely overlapped together most of time. To make it easier to compare, in the botton subplots of Figs. 7, 8, and 9, we show the relative differences of response times vesus throughput before saturation points (data after saturation points are meaningless, as the systems are not in a workable state). These subplots show that the response times of Physical-RAPID-Caches are very similar to that of the baseline system (within 1.5 percent most of time). Even in the worst case, which occurs for *Synthetic-C* trace under very high load, the Physical-RAPID-Caches have response-times only 2-4 percent higher than the baseline systems. The Logical-RAPID-Caches have slightly higher response times (1-10 percent higher) than the baseline systems. As discussed in the last section, such sightly increased response times should not be a major concern since the most important performance metrics here is throughput.

It is interesting to note that, for read-dominated traces such as *synthetic-C* (Fig. 9), the performance degrades gracefully when the workload increases, while for write-dominated traces such as *synthetic-A* and *synthetic-B* (Figs. 7 and 8) the performance changes relatively abruptly. When the workload increases, the read response times increase rapidly because of the increased disk traffic, while the write response times increase only gradually because of the use of the fast write technique. In write-dominated traces, the average response time is largely determined by the write response time; therefore, the average response times increase only gradually until the system write cache saturates. At this point, the average response times increase abruptly.

4.2 Performance Impacts of the Backup Cache

To obtain a better picture of how the backup cache affects the overall system performance, in this section, we study the effects of the backup NVRAM size and the maximum throughput of backup caches.

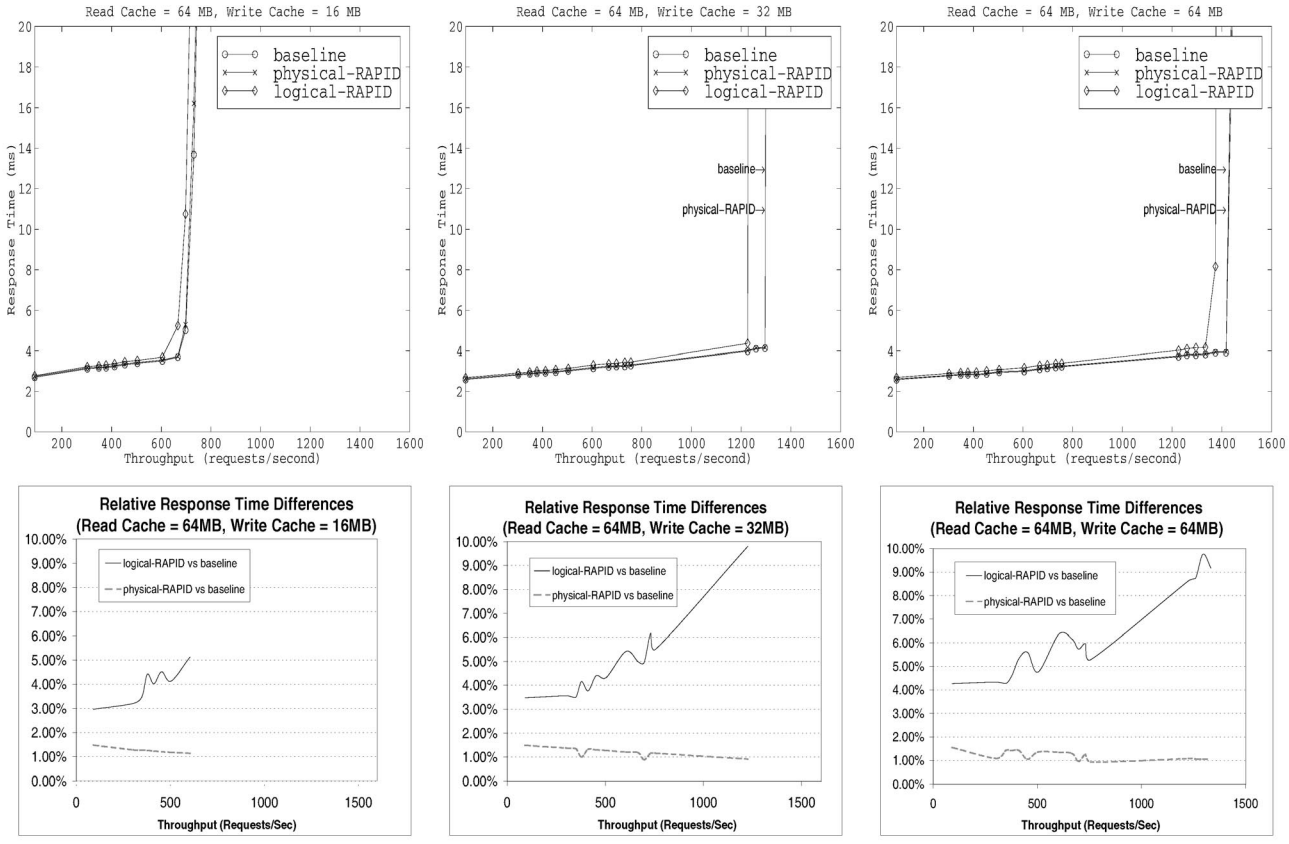

Fig. 7. Performance and relative response time difference of trace *Synthetic-A*.

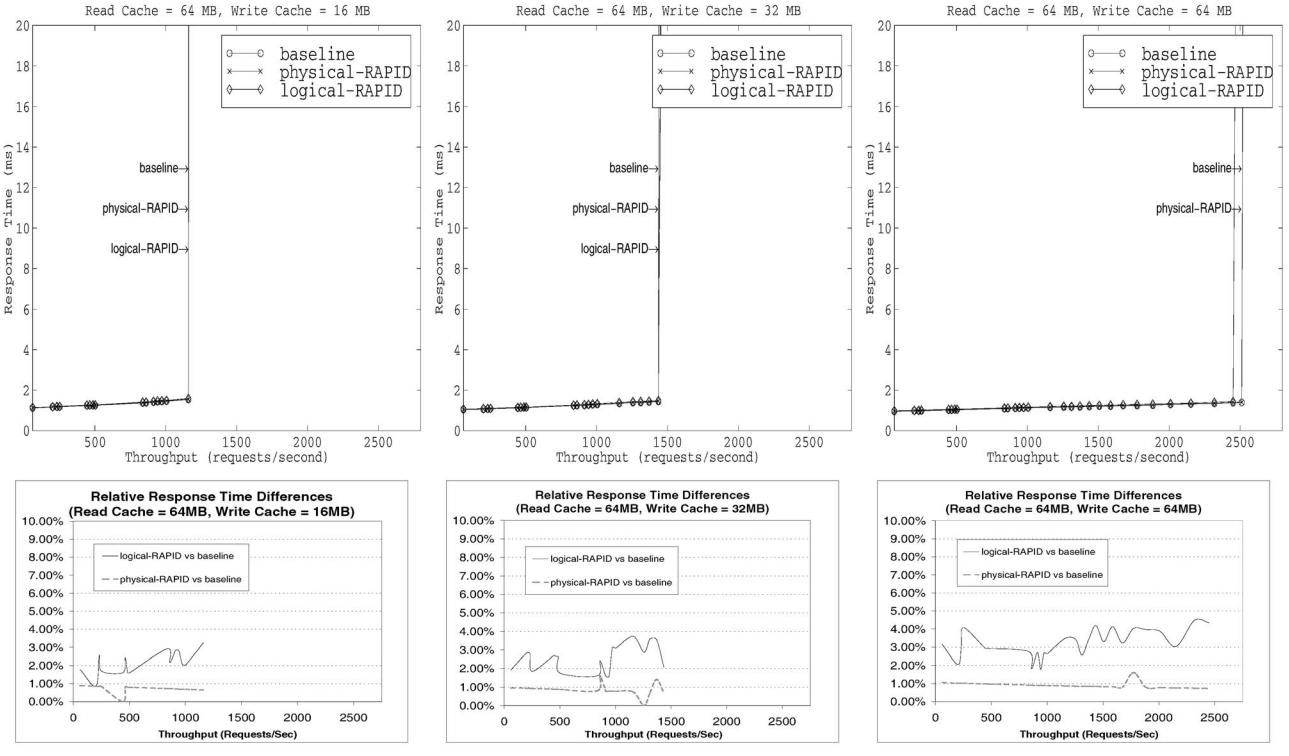

Fig. 8. Performance and relative response time difference of trace *Synthetic-B*.

4.2.1 Effects of the Backup NVRAM Size

To study the effect of the backup NVRAM size on the system performance, we varied the backup NVRAM size and reran some simulation experiments. Fig. 10 shows the performance of Physical-RAPID-Caches with three different backup NVRAM sizes, 0.5 MB, 1 MB, and 2 MB and

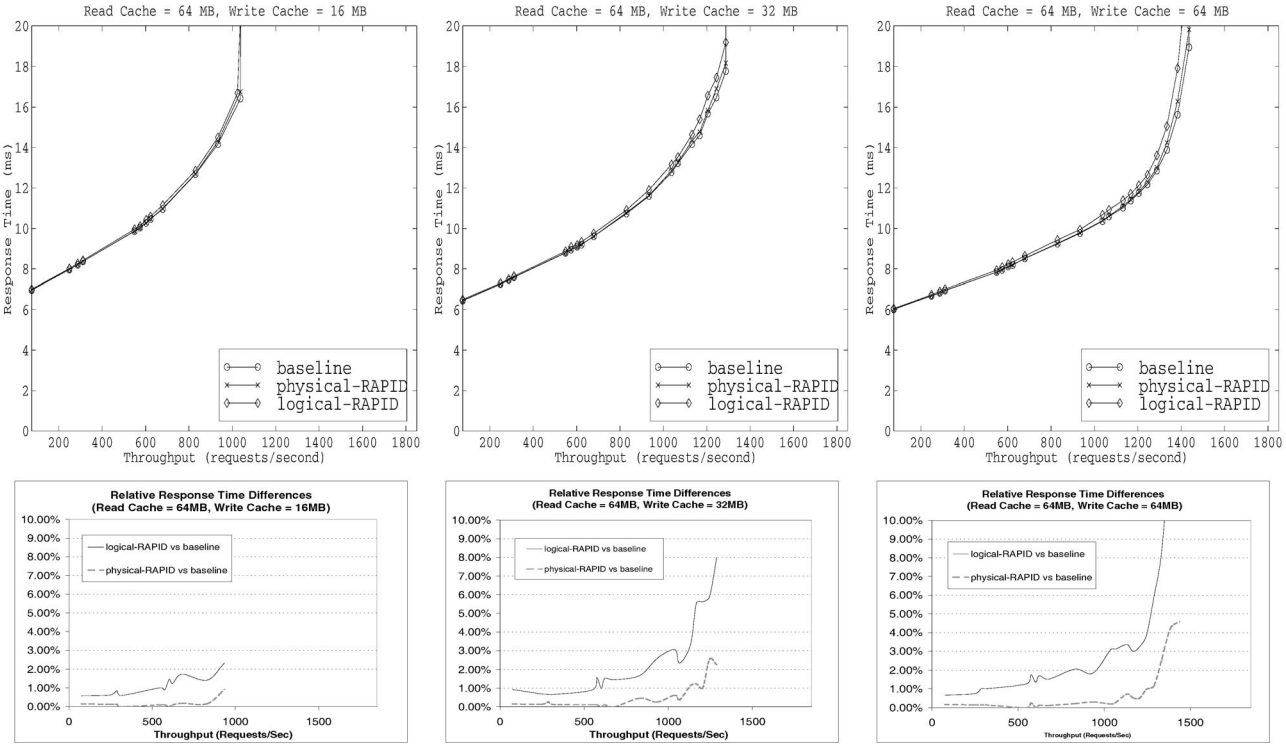

Fig. 9. Performance and relative response time difference of trace *Synthetic-C*.

compares them with that of a baseline system. We can see that, when the request rates are low, all three sizes perform equally well. However, the performance of the 0.5 MB case degrades at high workloads. Because of the limited backup NVRAM size in this case, the system may run out of segment buffers while all previous log writes are pending. As a result, if a following write request must evict a block from the backup LRU cache to a segment buffer, it must wait until a log write finishes so a segment buffer can be freed. This causes high response times and low throughput. When we increase the backup cache size to 1 or 2 MB, the system almost never runs out of segment buffers. Therefore, we are able to obtain nearly equivalent write performance as the baseline system. This is demonstrated by the fact that the response time versus throughout curve of the Physical-RAPID-Cache and that of the baseline system are almost

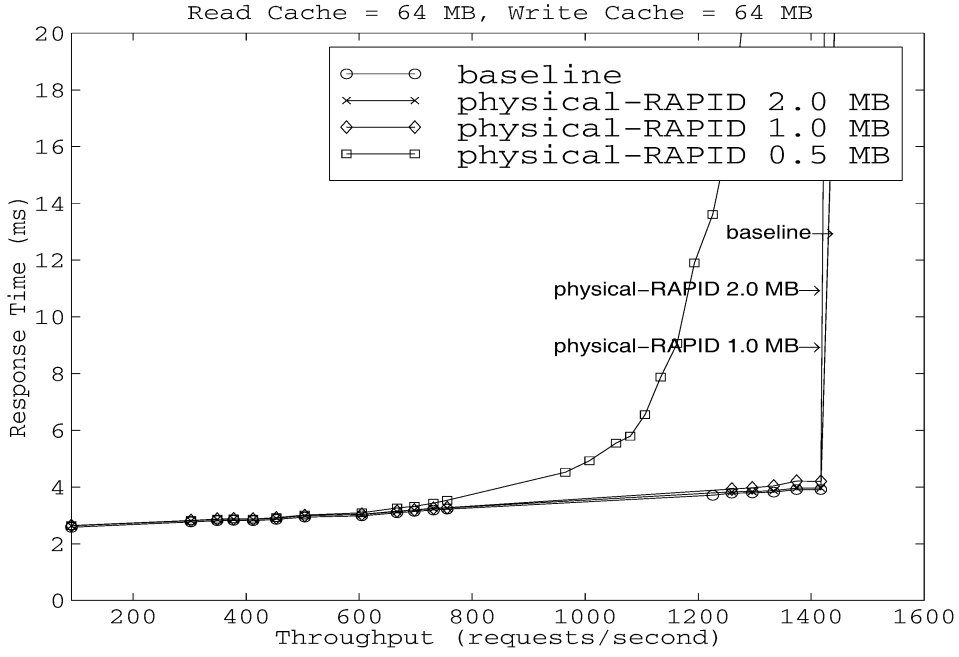

Fig. 10. Effects of Backup NVRAM Sizes on trace *Synthetic-A*.

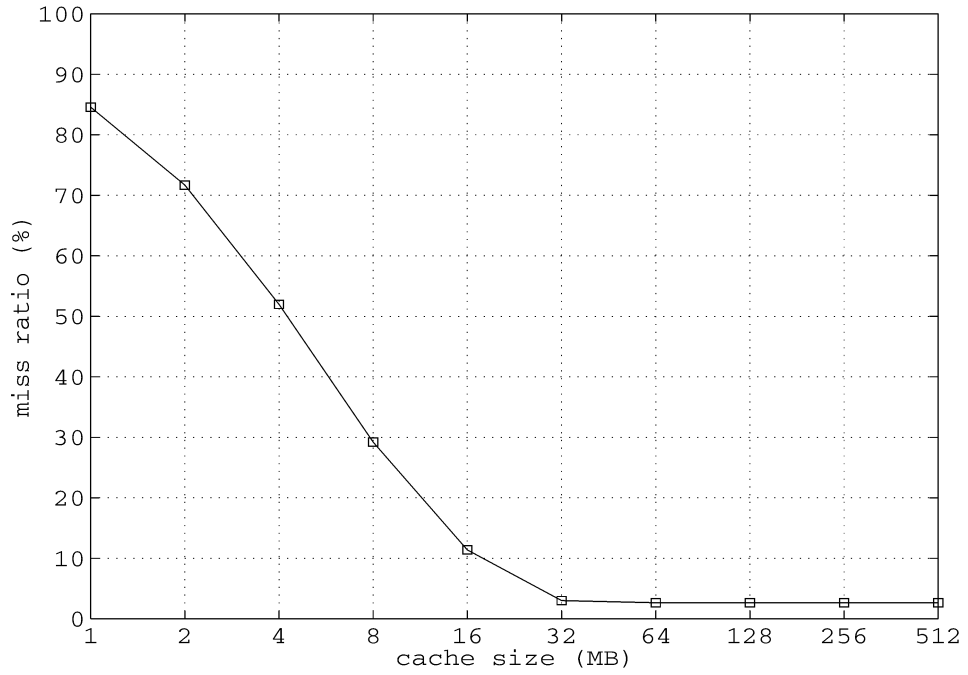

Fig. 11. Miss ratios versus cache sizes of trace Synthetic-D.

completely overlapped and it is often difficult to tell which line corresponds to which system. Similar results are observed for other traces and configurations.

4.2.2 The Maximum Throughput of the Backup Cache

In a RAPID-Cache, both the primary cache and the backup cache may affect the overall performance. Since the goal of the RAPID-Cache is to provide a low-cost backup cache that matches the write performance of the primary cache, we do not want the backup cache to become a potential performance bottleneck. In this section, we try to isolate the backup cache from the rest of the system and study its performance limitations. Basically, we assume a near-perfect primary cache that is much larger than the working-set of the workload. In such a system, the overall performance is only limited by the backup cache. Note here, we will only study the case of the dedicated cache disk. For the logical cache disk, it is difficult to isolate the interaction between the log writes and normal data reads/writes and we are currently studying ways to evaluate the performance limits of logical cache disks.

We constructed a new trace called *synthetic-D* for this purpose. The trace contains only write requests since we are only interested in the performance of the backup cache which is write only. As shown in Fig. 11, the trace has a working set of 32 MB. A cache larger than 32 MB can absorb more than 97 percent of all requests, eliminating much of the traffic to the data disks.

We chose a primary cache consisting of a 128 MB read cache and a 128 MB write cache, which is much larger than the working-set of the trace. The NVRAM in the backup cache is 2 MB. In addition to the default segment size of 128 KB, we also used a larger segment size of 256 KB to study the effect of larger segments. Fig. 12 shows the simulation results. In the case of the 128 KB segment size, the maximum throughput of the backup cache is about 1,400 write requests/second. For this pure-write trace, the backup cache with a 256 KB segment size has a higher throughput of 1,600 write requests/second because larger segments utilizes the disk bandwidth more efficiently. However, we found that for read-dominated traces, there is no noticeable performance difference between the cases of 128 KB and 256 KB segment sizes.

Notice however, that, the maximum throughput of 1,400-1,600 requests/second is only for this pure-write trace. On the other hand, most transaction-processing workloads are read-dominated. If a workload contains 60 percent of reads and 40 percent of writes, the write throughput of 1,400-1,600 requests/second will translate to a read/write throughput of 3,500-4,000 requests/second (because read traffic does not consume the backup cache bandwidth).

There are several possible ways to further improve the performance of the physical RAPID-Cache at very high workloads:

1. *Use a faster disk with a higher read-channel bandwidth.* Many new disk drives offer a higher bandwidth than the one we used in our simulations.
2. *Use multiple physical cache disks to obtain a higher aggregated bandwidth.* For example, in our simulation we use only one physical cache disk for 16 data disks. We can easily double the cache disk bandwidth by using two physical cache disks. Since many RAID products contain several spare disks, we can use multiple spare disks as multiple cache disks for free. Moreover, a disk costs only several hundreds dollars. Compared to expensive NVRAM, a RAPID-Cache with two or more physical cache disks is still a low cost solution, even when we have to use an extra dedicated cache disk instead of a "free" spare disk.

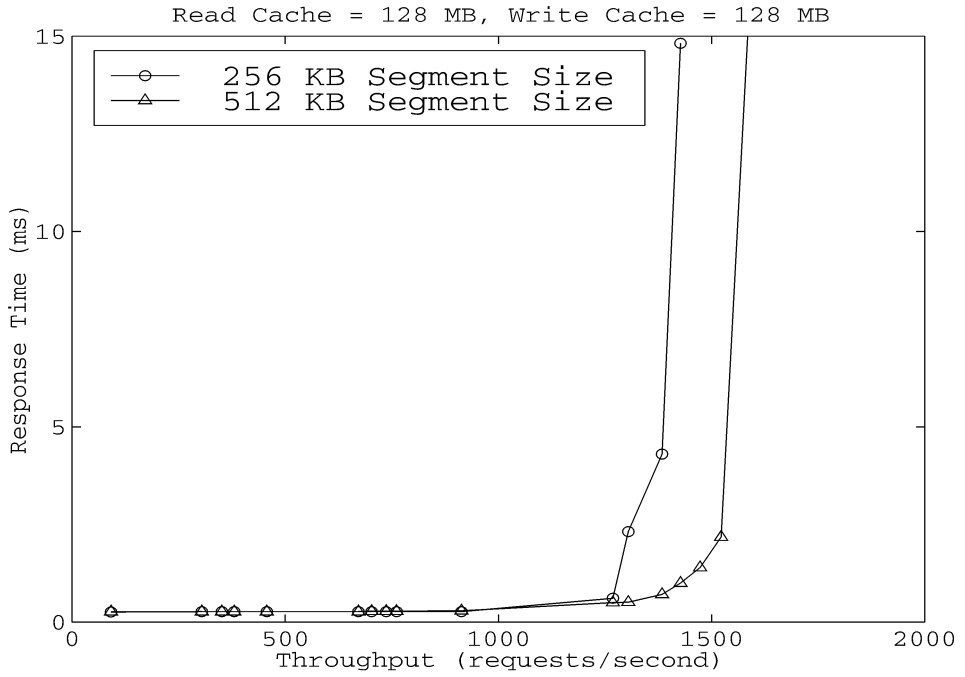

Fig. 12 Maximum throughput of the backup cache for trace Synthetic-D.

3. *Use a low cost solution that combines a physical cache disk with a logical cache disk.* In such a scheme, logs are normally written into a dedicated cache disk so that log writes will not affect the read performance. Once the load becomes so high that the dedicated cache disk saturates, the controller distributes a part of the log write traffic to the logical cache distributed among data disks, resulting in a higher combined log writing, bandwidth.

5 Reliability and Costs

5.1 Reliability Model

In this section, we analyze the reliability of the cache system based on exponentially distributed failures and repairs for both RAM and disks. We will consider four different cache configurations, namely, *RAPID-Cache with an NVRAM primary cache*, *RAPID-Cache with a DRAM primary cache*, *Single NVRAM cache*, and *Dual NVRAM caches*.

Consider a RAPID-Cache with an NVRAM primary cache. Assume that the primary cache consists of a number of NVRAM memory modules. Let $MTTF_{NVRAM}$ and $MTTF_{disk}$ represent the mean time to failure of an NVRAM module and the mean time to failure of a disk, respectively. In case of a failure, a repair process can start by replacing a failed memory module or a failed disk. It is reasonable to assume that the mean repair time for both RAM and disk is same denoted by $MTTR$. Recall that each write operation in the RAPID-Cache is performed in both the primary cache and the backup cache. If a memory module in the primary cache fails, the data are lost only if the component containing the same data copy in the backup cache also fails before the repair for the failed module is done. Similarly, if a component in the backup cache fails first, that data is lost if the copy in the primary cache also fails before repair. Let S_{PrimeC} and $S_{bkupRAM}$ be the size, in terms of memory modules, of primary cache and the size of the NVRAM in the backup cache, respectively. The mean failure rate caused by both a primary cache failure and a failure of the NVRAM of the backup cache is

$$S_{PrimeC}S_{bkupRAM}MTTR/(MTTF_{NVRAM})^2.$$

And the mean failure rate caused by both a primary cache failure and a disk failure is

$$S_{PrimeC}MTTR/(MTTF_{NVRAM}MTTF_{disk}).$$

The probability that the mirror copy of a primary cache data resides in the NVRAM of the backup cache is $S_{bkupRAM}/S_{PrimeC}$, and the probability that the mirror copy resides in the disk of the backup cache is $1 - S_{bkupRAM}/S_{PrimeC}$. Therefore, the mean failure rate of the entire cache system is given by

$$\lambda_1 = \frac{S_{PrimeC}S_{bkupRAM}MTTR}{(MTTF_{NVRAM})^2}\left(\frac{S_{bkupRAM}}{S_{PrimeC}}\right) + \frac{S_{PrimeC}MTTR}{MTTF_{NVRAM}MTTF_{disk}}\left(1 - \frac{S_{bkupRAM}}{S_{PrimeC}}\right).$$

It reduces to

$$\lambda_1 = \frac{(S_{bkupRAM})^2 MTTR}{(MTTF_{NVRAM})^2} + \frac{(S_{PrimeC} - S_{bkupRAM})MTTR}{MTTF_{NVRAM}MTTF_{disk}}. \quad (1)$$

The *mean time to data loss* ($MTTDL$) of the RAPID-Cache system with an NVRAM Primary cache is therefore given by

$$MTTDL_{RAPID-NP} = \frac{1}{\lambda_1} = \frac{(MTTF_{NVRAM})^2 MTTF_{disk}}{MTTR(MTTF_{disk}(S_{bkupRAM})^2 + MTTF_{NVRAM}(S_{PrimeC} - S_{bkupRAM}))}. \quad (2)$$

TABLE 3
Reliability and Costs Comparison of the Different Write Cache Architectures

Cache sizes	Reliability (MTTDL in hours)				Cost (US Dollars)			
Read/Write	Single	Dual-copy	RAPID	RAPID	Single	Dual-copy	RAPID	RAPID
(MB/MB)	NVRAM	NVRAM	Cache-DP	Cache-NP	NVRAM	NVRAM	Cache-DP	Cache-NP
64/16	$1.25*10^4$	$5.21*10^7$	$6.70*10^5$	$4.53*10^7$	\$912	\$1,792	\$151	\$1,023
64/32	$6.25*10^3$	$2.60*10^7$	$6.36*10^5$	$3.86*10^7$	\$1,792	\$3,552	\$159	\$1,903
64/64	$3.13*10^3$	$1.30*10^7$	$5.77*10^5$	$2.98*10^7$	\$3,552	\$7,072	\$175	\$3,663

The size of the NVRAM buffer of the RAPID-Cache is 2 MB. Note: RAPID Cache-DP refers to a RAPID-Cache with a DRAM primary write cache. RAPID Cache-NP refers to a RAPID-Cache with an NVRAM primary write cache.

As mentioned previously, NVRAM is orders of magnitude more expensive than regular DRAM. With the new RAPID-Cache architecture, it is possible to implement the primary cache using DRAM instead of NVRAM. Using DRAM as a primary write cache, on the other hand, may compromise the reliability of the cache. In addition to RAM failures, data may get lost due to several other reasons such as a power failure, hardware failures such as CPU failures, environment failures, etc. To cope with frequent power failures, most systems use an UPS to prevent data loss from sudden power failures. In this case, data loss occurs in the DRAM only when the power fails and the UPS also fails. If a hardware failure such as a CPU failure occurs, replacing the failed hardware usually requires a system shutdown. As a result, all data in a DRAM will be lost and they have to be recovered from the backup cache. The data failure rate at the backup cache is $S_{bkupRAM}/MTTF_{NVRAM} + 1/MTTF_{disk}$. Let *MTTPF*, $MTTF_{UPS}$, and *MTTHF* be the mean time to power failure, mean time to failure for the UPS, and the mean time to hardware failure, respectively. Assume that the mean time to repair is the same for all types of failures ($MTTR$). Then, the mean failure rate is given by

$$\lambda_2 = \left(\frac{S_{bkupRAM}}{MTTF_{NVRAM}} + \frac{1}{MTTF_{disk}}\right) \left(\frac{S_{PrimeC}MTTR}{MTTF_{DRAM}} + \frac{MTTR}{MTTHF} + \frac{MTTR^2}{MTTPF * MTTF_{UPS}}\right), \quad (3)$$

and the mean time to data loss of the RAPID-Cache with a DRAM Primary cache is

$$MTTDL_{RAPID-DP} = \frac{1}{\lambda_2}. \quad (4)$$

The reliability analysis of the other two cache architectures is straightforward. For the single NVRAM cache case, the mean time to data loss is simply $MTTF_{NVRAM}/S_{PrimeC}$. The mean time to data loss for the dual NVRAM case is $(MTTF_{NVRAM})^2/(S_{PrimeC}MTTR)$

5.2 Reliability and Costs Comparison

One important factor that determines the reliability of the write cache is the mean time to failure of NVRAM. Unfortunately, remarkably little data is available from literature or data sheets of various RAM products. Savage and Wilkes [7] cited 25-87K hours of data retention lifetimes of Integral Lithium-cell-backed static RAM that are extremely expensive and 15K hours of predicted MTTF for the popular PrestoServe card. With the lack of published data for MTTF, we assume optimistically an MTTF of RAM to be 200k hours, which favors conventional NVRAM cache architectures and represents a conservative evaluation for RAPID-Cache. Our analysis also assumes that the NVRAM cache consists of a number of 2MB modules [21]. Some existing disk systems such as the RAIDs from Storage Computer Co. use independent modules to constitute a write cache. Power failures and UPS failure are another source of possible data loss if DRAM is used. We assume that all the systems considered are backed up by UPS systems. We chose the mean time to power failure (MTTPF) of 4,300 hours [22] and the MTTF of UPS ($MTTF_{UPS}$) of 200k hours [7] in our analysis. The mean time to hardware failures and environment failures etc. is assumed to be one month or 720 hours [4]. The MTTF for disks, $MMTF_{disk}$, is assumed to be one million hours. The mean time to repair for all types of failures here is assumed to be 48 hours.

Cost figures for semiconductor devices and disks change very rapidly. It is difficult to give an accurate and up-to-date cost evaluation. In order to give a general idea for the cost of the different cache architectures, we made the following assumptions: The cost of DRAM is \$0.5/MB and the cost of NVRAM is \$55/MB which was quoted by a major NVRAM manufacturer as of December 2000. The cost of disk space is 0.5 cents/MB. For a logical cache disk, it is quite reasonable to use this cost number since the cache space of a logical cache disk is located in partitions of data disks. In the case of a physical cache disk, the cache space is located in a partition of a hot standby disk as explained previously. Therefore, it is also reasonable to use the per MB cost figure since no additional disk drive is needed. If no spare disk were available in a RAID, a dedicated cache disk would be required that will add to the cost an additional \$100 for a minimum size disk drive available in the market.

Table 3 lists the reliability and cost comparison between the baseline cache and our RAPID-Cache for three typical configurations. For easy comparison, we also summarize the performance of different cache architectures under different traces in Table 4.

It is clear from the table that the difference between the baseline cache and the RAPID-Cache in terms of reliability and cost is significant. Compared to a single copy NVRAM write cache, the reliability of the RAPID-Cache with an NVRAM write cache is five orders of magnitudes higher than that of the baseline cache system while the additional cost introduced by the RAPID-Cache is only between 3 percent and 11 percent. Compared to the dual copy

TABLE 4
Performance of Different Cache Architectures Under Synthetic Traces

Cache sizes Read/Write (MB)	Logical RAPID-Cache Max. Throughput (Req./Sec.)			Baseline and Physical RAPID-Cache Max. Throughput (Req./Sec.)		
	Synthetic-A	Synthetic-B	Synthetic-C	Synthetic-A	Synthetic-B	Synthetic-C
64/16	710	1130	1040	740	1130	1040
64/32	1220	1420	1300	1300	1420	1300
64/64	1380	2470	1400	1440	2520	1440

NVRAM caches, the RAPID-Caches with an NVRAM primary cache still have higher reliability than the baseline cache system (because disks are more reliable than NVRAM) with approximately half of the cost. For the RAPID-Cache with a DRAM write primary cache, its reliability is three orders of magnitude better than a single-copy NVRAM cache. More importantly, the cost of the RAPID-Cache is dramatically lower than both single-copy NVRAM caches and dual-copy NVRAM caches. In some cases, the cost of the RAPID-Cache is only 5 percent of the single-copy NVRAM caches and 3 percent of the dual-copy NVRAM caches. In other words, the RAPID-Cache provides similar reliability to that of the baseline cache which costs over 1,000 dollars. Moreover, a RAPID-Cache can use a much larger primary cache to improve its performance while still maintaining its low cost. For example, a RAPID-Cache with a 64 MB DRAM read cache and a 64 MB DRAM write cache costs only $175. Yet it provides more than two times higher throughput and three orders of magnitude better reliability than a single-copy NVRAM write cache of the 16 MB, which costs $912.

As we pointed out, cost figures for semiconductor devices and disks change very rapidly. In the past three years, the costs of DRAM and disk have dropped by 10 times. The cost of NVRAM, on the contrary, has dropped by less than 50 percent. It is very likely that this trend will continue. As a result, the cost difference between our schemes and the traditional NVRAM Cache systems will only increase in the future. Moreover, I/O systems are using larger and larger write caches to accommodate the increasing I/O demands. The larger the caches, the bigger the savings our scheme will provide.

6 Related Work

Ng and Chen [23], [24] proposed Rio (RAM I/O) to implement reliable RAM. Rio is a software system that makes RAM to survive operating system crashes. The goals of RAPID-Caches and RIOs are very different. RAPID-Caches are hardware soultions that tolerate hardware failures, power outages, and software crashes. Rio focuses on surviving software crashes only.

The idea of using a disk-based log to improve system performance or to improve the reliability of RAM has been used in both file systems and database systems for a long time. For example, the Log-structured File System (LFS) [11], [12], [13], the Journal File System (JFS), and other similar systems all use disk-based data/metadata logging to improve file system performance and speed-up crash recovery. Database systems have long been using elaborate logging techniques to improve the reliability of the RAM buffer and to implement the transaction semantics. NVRAM has been used by many database systems to reduce the overhead of logging.

Several RAID systems have implemented the LFS algorithm at the RAID controller level [25], [17]. LFS collects writes in a RAM buffer to form large logs and writes large logs to data disks. While LFS has demonstrated superior performance for many workloads, studies have shown that the garbage collection overhead of LFS can become a major performance bottleneck in transaction-processing environments, decreasing the system performance by 34-40 percent [13], [26]. The garbage collection overhead becomes very high when the disk utilization reaches 80 percent of the total disk capacity [13], [25].

Disk Caching Disks (DCD) proposed in [16] shows that it is possible to implement a large nonvolatile write cache inexpensively. DCD uses a small NVRAM cache and a small cache disk to form a two-level cache. Write data are first assembled in the small NVRAM cache and logged into the cache disk later. Data in the cache disk is destaged to the data disk during idle periods. The two-level hierarchical structure acts as a large nonvolatile cache, but its cost is much lower than that of a large NVRAM cache. While DCD has excellent performance for low to medium traffic workloads, directly applying DCD to high I/O workloads may face the following problems. DCD requires destaging which involves reading dirty data from the cache disk and writing them into the data disk. The destaging process may become a performance bottleneck at high loads because the destage reads and the log writes will compete for the cache disk bandwidth. Moreover, the read speed of DCD is also slow because data may have to be read from the cache disk.

eNVy [8] is a large nonvolatile main memory storage system based on flash EPROM. Flash EPROM has some disk-like characteristics, i.e, data must be erased in blocks and the write speed is slow. eNVy solved the write problem by using a battery-backed SRAM in front of the flash-EPROM. Data are first written into the SRAM and then parallelly transferred into the flash EPROM in large blocks. The whole system appears to users as a large high-speed NVRAM.

While the idea of RAPID-Cache is inspired by the previous research, especially LFS and DCD, there are several important differences as highlighted below.

- LFS and DCD do not address the reliability problem of single copy caches.

- In both LFS and DCD, data are collected in a RAM buffer and logged into disks when the buffer is full. In the backup cache of RAPID-Cache, data are written into an LRU cache made of NVRAM. Active data may be overwritten in the LRU cache frequently. Only the inactive data evicted from the LRU cache are collected in a segment buffer and logged into the cache disk. The separation of active data from inactive data significantly reduces the cache disk traffic and garbage collection cost.
- In DCD, data in the cache disk must be destaged into the data disk, which may become a performance bottleneck at high workloads. In RAPID-Cache, there is no need to read the cache disk during destaging since all dirty data can be accessed from the primary cache. Therefore, the destage overhead of RAPID-Cache is the same as that of a conventional system with a single-copy or a dual-copy NVRAM write cache.
- LFS needs garbage collection which significantly limits the system performance in some cases. The asymmetrically parallel architecture of RAPID-Cache and the separation of active data from inactive data in the backup cache enable us to design a garbage collection algorithm that is much more efficient than the one used by LFS. Moreover, RAPID-Cache seldom requires garbage collection. In fact, the garbage collection overhead of RAPID-Cache is so low that it has virtually no impact on system performance.

7 Conclusions

Using NVRAM caches can significantly improve the write performance of disk systems. However, because of the high cost of NVRAM, in some disk systems the cost of NVRAM caches is much higher than that of disks themselves and dominates the overall system cost. In addition, a single-copy NVRAM cache creates a single point of failure in a highly reliable disk system while a dual-copy NVRAM cache is even more expensive.

In this paper, we have presented a new disk cache architecture called RAPID-Cache. While RAPID-Caches can be used for any I/O systems, it is particularly useful for improving the performance and reliability of large, parallel disk systems such as RAIDs. The main feature of the RAPID-Cache is its asymmetrically parallel architecture that consists of a fast-write-fast-read primary cache and an inexpensive, fast-write-slow-read hierarchical backup cache. We trade the read performance of the backup cache for economy and reliability. Fortunately, the compromise in read performance of the backup cache does not affect the system performance in any way because read operations from the backup cache are necessary only during error recovery periods. On the other hand, the economy, reliability and performance gained have been shown to be dramatic. Such win-win trading is made possible by exploiting the locality of disk accesses and efficiency of large disk transfers. We have shown through simulation experiments and analysis that it is possible to configure the RAPID-Cache in a number of ways to optimize throughput, reliability, or system cost:

- Compared to a single-copy NVRAM cache, a RAPID-Cache with a DRAM primary cache has much higher reliability and similar performance with only a fraction of the cost.
- Compared to a single-copy NVRAM cache, a RAPID-Cache with an NVRAM primary cache has much better reliability and similar performance with only slightly higher cost.
- Compared to a dual-copy NVRAM cache, a RAPID-Cache with an NVRAM primary cache has similar or better reliability and similar performance with only half of the cost.
- Because its low cost, with the *same budget*, RAPID-Caches can have significantly higher performance compared to conventional NVRAM cache architectures by affording a much larger primary cache size, while still maintaining good reliability.
- The asymmetrically parallel architecture of RAPID-Caches and its algorithm that separates active data from inactive data virtually eliminate the garbage collection overhead, which are the major problems associated with previous solutions such as LFS and DCD.

Furthermore, using DRAM for the primary cache makes it economically feasible to combine the read cache with write cache, resulting in a unified cache that has significant performance advantages. Such a unified cache would be very expensive to implement with the existing dual-copy cache architectures because of the requirement of the large read cache that would have to be NVRAM if combined with a write cache. While we have not presented the results of unified RAPID-Cache in this paper because of the space limitation, our simulation results show that a unified RAPID-Cache can achieve 2-4 times higher throughput than a split cache with the same total cache size. The low cost feature of the RAPID-Cache also makes it possible to use a very large primary cache to achieve very high performance for high-end systems. Therefore, a wide range of disk I/O systems can benefit from the RAPID-Cache architecture.

Acknowledgments

This research is supported in part by US National Science Foundation Grants MIP-9505601 and MIP-9714370, US National Science Foundation Career Award CCR-9984852, and an Ohio Board of Regents Computer Science Collaboration Grant. The authors would like to thank Dr. David Kotz of Dartmouth College for the use of his disk simulator. The authors would also like to thank HP Laboratories and EMC Corporation for providing them with disk traces. The authors benefited from discussions with Dr. Jien-Chung Lo of University of Rhode Island on reliability issues of NVRAM. Some algorithms of the synthetic trace generator were initially designed by Qi Zhang. A preliminary work of this research [1] was presented at the Fifth International Symposium on High Performance Computer Architecture (HPCA-5), January 1999, Orlando, Florida.

REFERENCES

[1] Y. Hu, Q. Yang, and T. Nightingale, "RAPID-Cache—A Reliable and Inexpensive Write Cache for disk I/O Systems," *Proc. Fifth Int'l Symp. High Performance Computer Architecture (HPCA '99),* Jan. 1999.

[2] J. Menon and J. Cortney, "The Architecture of a Fault-Tolerant Cached RAID Controller," *Proc. 20th Ann. Int'l Symp. Computer Architecture,* pp. 76-86, May 1993.

[3] K. Treiber and J. Menon, "Simulation Study of Cached RAID5 Designs," *Proc. Int'l Symp. High Performance Computer Architectures,* pp. 186-197, Jan. 1995.

[4] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, and D.A. Patterson, "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys,* vol. 26, pp. 145-188, June 1994.

[5] A. Varma and Q. Jacobson, "Destage Algorithms for Disk Arrays with Nonvolatile Caches," *Proc. 22nd Ann. Int'l Symp. Computer Architecture,* pp. 83-95, June 1995.

[6] C. Ruemmler and J. Wilkes, "UNIX Disk Access Patterns," *Proc. Winter 1993 USENIX,* pp. 405-420, Jan. 1993.

[7] S. Savage and J. Wilkes, "AFRAID—A Frequently Redundant Array of Independent Disks," *Proc. 1996 USENIX Technical Conf.,* Jan. 1996.

[8] M. Wu and W. Zwaenepoel, "eNVy, A Nonvolatile, Main Memory Storage System," *Proc. Sixth Symp. Architectural Support for Programming Languages and Operating Systems,* pp. 86-97, Oct. 1994.

[9] S. Akyürek and K. Salem, "Management of Partially Safe Buffers," *IEEE Trans. Computers,* vol. 44, no. 4, pp. 394-407, Mar. 1995.

[10] D. Coombs, "Drawing up a New RAID Roadmap," *Data Storage,* vol. 3, pp. 59-61, Dec. 1996.

[11] J. Ousterhout and F. Douglis, "Beating the I/O Bottleneck: A Case for Log-Structured File Systems," technical report, Computer Science Division, Electrical Eng. and Computer Sciences, Univ. of California at Berkeley Oct. 1988.

[12] M. Rosenblum and J. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Trans. Computer Systems,* pp. 26- 52, Feb. 1992.

[13] M. Seltzer, K. Bostic, M.K. McKusick, and C. Staelin, "An Implementation of a Log-Structured File System for UNIX," *Proc. Winter 1993 USENIX,* pp. 307-326, Jan. 1993.

[14] D. Stodolsky, M. Holland, and W.V. Courtright II, and G.A. Gibson, "Parity Logging Disk Arrays," *ACM Trans. Computer Systems,* pp. 206-235, Aug. 1994.

[15] B.T. Zivkov and A.J. Smith, "Disk Caching in Large Databases and Timeshared Systems," Technical Report CSD-96-913, Computer Science Division, Univ. of California, Berkeley, Sept. 1996.

[16] Y. Hu and Q. Yang, "DCD—Disk Caching Disk: A New Approach for Boosting I/O Performance," *Proc. 23rd Int'l Symp. Computer Architecture (ISCA '96),* pp. 169-178, May 1996.

[17] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan, "The HP AutoRaid Hierarchical Storage System," *ACM Trans. Computer Systems,* vol. 14, pp. 108-136, Feb. 1996.

[18] D. Kotz, S.B. Toh, and S. Radhakrishnan, "A Detailed Simulation Model of the HP 97560 Disk Drive," Technical Report PCS-TR94-220, Dept. of Computer Science, Dartmouth College, July 1994.

[19] C. Ruemmler and J. Wilkes, "An Introduction to Disk Drive Modeling," *IEEE Computer,* pp. 17-28, Mar. 1994.

[20] G.R. Ganger, "Generating Representative Synthetic Workloads—An Unsolved Problem," *Proc. Computer Measurement Group (CMG) Conf.,* pp. 1263-1269, Dec. 1995.

[21] "DS1270Y/AB 16M Nonvolatile SRAM data sheet. "Dallas Semiconductor Year?????

[22] G.A. Gibson and D.A. Patterson, "Designing Disk Arrays for High Data Reliability," *J. Parallel and Distributed Computing,* vol. 17, pp. 4-27, Jan./Feb. 1993.

[23] W.T. Ng and P.M. Chen, "The Design and Verification of the Rio File Cache," *IEEE Trans. Computers,* vol. 50, no. 4, pp. 1-16, Apr. 2001.

[24] W.T. Ng and P.M. Chen, "Integrating Reliable Memory in Databases," *Proc. 1997 Int'l Conf. Very Large Data Bases (VLDB),* pp. 76-85, Aug. 1997.

[25] J. Menon, "A Performance Comparison of RAID-5 and Log-Structured Arrays," *Proc. Fourth IEEE Int'l Symp. High Performance Distributed Computing,* pp. 167-178, Aug. 1995.

[26] M. Seltzer, K.A. Smith, H. Balakrishnan, J. Chang, S. McMains, and V. Padmanabhan, "File System Logging versus Clustering: A Performance Comparison," *Proc. USENIX 1995 Technical Conf.,* pp. 249-264, Jan. 1995.

Yiming Hu received the BE degree in computer engineering from Huazhong University of Science and Technology, China. He received the PhD degree in electrical engineering from the University of Rhode Island, Kingston, in 1998. He is an assistant professor of computer science and engineering at the University of Cincinnati, Ohio. His research interests include computer architecture, operating systems, and performance evaluation. He is a recipient of a US National Science Foundation CAREER Award. He is a senior member of the IEEE and a member of the ACM.

Tycho Nightingale received the BA and MS degrees in electrical engineering in 1997 and 1999, respectively, from the University of Rhode Island, Kingston. He is a software engineer at Sun Microsystems, where he works in the area of full-system simulation. His interests include computer architecture, system simulation, and storage systems.

Qing (Ken) Yang received the BSc in computer science from Huazhong University of Science and Technology, Wuhan, China, in 1982, the MASc degree in electrical engineering from University of Toronto, Canada, in 1985, and the PhD degree in computer engineering from The Center for Advanced Computer Studies, University of Louisiana at Lafayette, in 1988. Presently, he is a distinguished engineering professor in the Department of Electrical and Computer Engineering at the University of Rhode Island where he has been a faculty member since 1988. His research interests include computer architectures, memory systems, disk I/O systems, computer networks, parallel and distributed computing, and performance evaluation. Dr. Yang is a senior member of the IEEE and a member of the SIGARCH of the ACM.