A New Hierarchical Disk Architecture *

Yiming Hu and Qing Yang
Dept. of Electrical & Computer Engineering
University of Rhode Island
Kingston, RI 02881
e-mail: {hu,qyang}@ele.uri.edu

Abstract

Large RAM caches are generally used to speed up disk accesses. Such caches more effectively improve read performance than write performance, since write requests must be frequently written into disks to protect them from data loss or damage due to system failures. While Non-volatile RAM (NVRAM) caches can be used to improve write performance, large NVRAM caches are too expensive for many applications. This paper presents a new disk cache architecture called *DCD*, Disk Caching Disks. DCD takes the advantage of large data transfer sizes and uses inexpensive disk space to provide a high-performance, low-cost and reliable caching solution.

keywords:
Cache, Disk, NVRAM, Storage System, Storage Hierarchy

1 Introduction

Semiconductor technologies have advanced very rapidly for the past decades. Disk storage (hard disk drives), on the other hand, have not kept pace with RAM in terms of access speed because of the mechanical nature of magnetic disks, although the storage capacity of disks increased drastically. The result is a wider speed gap between RAM and disks. Such a gap will eventually become the main obstacle for further development of computer technology.

1.1 Background

There has been extensive research reported in the literature in improving disk system performance. Previous studies on disk performance can generally be classified into two categories: improving the disk subsystem architecture, and improving the file system that controls and manages disks.

One of the most important architectural advances in disks is the RAID (Redundant Array of Inexpensive Disks) architecture [1]. The main idea of RAID is to use multiple disks in parallel to increase the total I/O bandwidth that scales with the number of disks. Multiple disks in a RAID can serve a single large I/O request or support multiple independent I/Os in parallel. The most commonly used RAID architecture is RAID-5. While RAID-5 is an effective approach to high I/O performance, it suffers from the well-known "small write" problem, that is, its throughput is penalized by a factor of four over non-redundant arrays for small writes [1]. The penalty results from parity calculation for new data, which involves readings of old data and parity, and writings of new data and parity.

The RAID architectures are primarily aimed for high throughput by means of parallelism rather than reducing access latency. For low average throughput workloads such as those in office/engineering environments, performance enhancement due to RAID is very limited [1]. Caching is the main mechanism for reducing access latency. Modern file systems generally use large RAM caches to speed up disk accesses. Such caches more effectively reduce read traffic than write traffic, since write requests must be frequently written into disks to protect them from data loss or damage due to system failures [2, 3, 4]. As the RAM size increases rapidly and absorbs more read requests, the proportion of write traffic seen by disk systems will dominate disk traffic and may potentially become a system bottleneck [4]. While it is possible to improve the write performance by using Non-volatile RAM (NVRAM) cache [2, 5], the write buffer size is usually very small compared to disk capacity because of the high cost of NVRAM[1]. Such a small buffer gets filled up very quickly and can hardly catch the locality of large I/O data. Large NVRAM caches are cost-prohibitive making it infeasible for many applications except for large-scale systems such as banking applications where costs are not a primary concern.

Since attempts in improving the disk subsystem architecture have so far met with limited success for write performance, extensive research has been reported in improving file systems. One of the most important work in file systems is the Log-structured File System (LFS) [4, 3, 6]. The central idea of LFS is to improve write performance by buffering a sequence of small writes in a cache to form a large log, and then writing the large log to a disk in one disk operation when the cache is full. As a result, many small and random writes of the traditional file system are converted into a large sequential transfer in LFS. In this way, LFS eliminates the random seek times and rotational latencies associated with small write operations thereby improving the disk performance significantly.

While LFS has a great potential for improving write performance of traditional file systems, it has not been commercially very successful since it was introduced more than ten years ago. Applications of LFS are mainly limited to academic research such as Sprite LFS [3] and BSD-LFS [6] as well as some RAID systems. This is because LFS requires rewriting the file system and a host of utility programs [7, page 350], needs a high cost cleaning algorithm, and is much more sensitive to disk capacity utilization than traditional file systems [3, 4]. The performance

*Will appear in *IEEE Micro*.

[1] For example, Dallas Semiconductor sells NVRAM with embedded lithium-cell batteries for about $100/MB.

Disk Access Time Breakdown

For each disk access, there are 4 components that contributes to the total access time: *Controller Overhead, Seek Time, Rotational Latency* and *Data Read/Write Time*.

Data on disks are organized in *tracks* and *sectors*. When a disk drive receives a read or write request from the host, the disk controller must spend time to get the request command and data from/to the bus and analyse the request. This time is called controller overhead which is typically around 1–2 ms. To read or write data on the disk, the controller has to spend time to move the disk head to the target track, called seek time, which takes 8–12 ms on average. Once the head is positioned on the desired track, the disk has to wait until the target sector to rotate under the head before it can start to read or write data. This rotational latency has an average value of the half of the disk rotation time. For a modern disk with a rotational speed of 7200 revolutions per minute (RPM), the average rotational latency is about 4.2 ms. The data read/write time is for actually reading or writing data from/to the disk media. Its value depends on the request size as well as the read/write channel speed, which is the data transfer rate between the disk head and disk media (not to be confused with the data transfer rate of the interface bus). Many disks nowadays have a read/write channel speed of 20 MB/sec or higher. For data blocks of 4 KB and 1 MB, it takes 0.2 ms and 51.2 ms, respectively, to transfer the data from/to the disk media.

On average, to write a data block of 4 KB, a typical modern disk has to spend 1 ms on controller overhead, 8 ms on seek time and 4.2 ms on rotational latency. The total overhead is 13.2 ms. Only about 0.2 ms is needed to actually write the data to the disk media. The overhead dominates the total disk access time and limits the effective data transfer rate to about *0.3 MB/Sec* (4 KB/13.4 ms.) On the other hand, writing a data block of 1 MB takes 64.4 ms (13.2 ms on total overhead and 51.2 ms on writing the data to the disk media), translating to an effective data transfer rate of *15.9 MB/Sec* (1 MB/64.4 ms), which is 53 times higher than that of 4 KB writes. This example clearly shows that writing data to disks in large sizes is much more efficient than in small sizes.

of LFS degrades rapidly when the disk becomes full and gets worse than the current file system when the disk utilization approaches 80%. In addition, LFS needs to buffer a large amount of data for a relatively long period of time (typically 30 to 60 seconds) in order to write into disk later as a log, which may cause reliability problems. Finally, as pointed out by Stodolsky et al. [8], the performance of LFS in read-intensive workloads may be degraded if the read and write access patterns differ widely, since logically nearby blocks may not be stored physically nearby in a LFS.

1.2 The Burstiness of Disk I/O

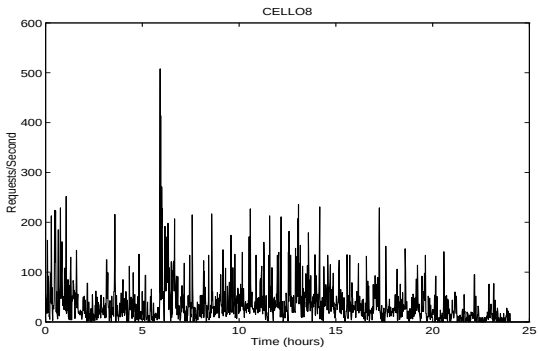

Figure 1: The Burstiness of Disk I/O

Many disk I/O workloads show the characteristic of burstiness—that is, requests are often clustered together in a short time frame. In addition, there is usually a relatively long period of interval time between two consecutive request bursts. During the interval time, the I/O system shows significantly fewer activities or even becomes idle. Figure 1 shows the changes of disk request rates over a 24-hour period in a typical office/engineering workload. While the average request rate of this trace is only about 10 requests/second, the figure shows many bursts as high as 100–200 requests/second. The highest peak in this graph goes over 500 requests/second.

Such burstiness is very common in office/engineering environments, as observed by Ruemmler and Wilkes [9]. One possible reason to this bursty pattern is the periodical flushing of dirty data from the cache by the UNIX operating system. Another possible reason is that, in a UNIX system, each file creation/deletion operation causes 5 disk accesses and each file read takes at least 2 disk accesses. Moreover, users tend to read or write a group of files, such as copying, moving, deleting or compiling a group of files. Moving and compiling are especially file system intensive operations because they involve reading, creating, writing and deleting files.

Many other systems also demonstrate burstiness. For example, Treiber and Menon [5] found that in a database I/O trace, the peak read rate is 1250 blocks/second and the peak write rate is 440 blocks/second, while the average I/O rate is only about 3.5 blocks/second/GB of data.

While conventional disk caches and RAID work well with the "background" requests, the large peaks pose a serious challenge to them. These large peaks will quickly overflow a RAM cache, unless the cache size is very large. A normal RAID system will not work well either, unless tens of disks are available in the RAID system to process them in parallel and all the requests can be distributed into different disks. Furthermore, a very large RAM or a large number of disks presents a poor performance/cost ratio, since most of time the system is idle or less busy, resulting in very low hardware utilization.

1.3 The DCD Approach

One important characteristic of disks is that writing data to disks in large sizes is an order of magnitude more efficient than in small sizes (See side-bar *Disk Access Time Breakdown*). Base on this observation, we propose a new disk organization referred to as *Disk Caching Disk*, or DCD for short. The fundamental idea behind DCD is to use a log disk, called *cache-disk*, as an extension of a small RAM buffer on top of a data-disk. The RAM buffer and the cache-disk together cache write data. Cached-data are moved to the *data-disk* afterward when the system is idle.

In a DCD system, the small RAM buffer captures the "background" random write requests. When a large write burst comes in, the RAM buffer is quickly filled. The DCD then writes all the data blocks in the RAM buffer, *in one large data transfer*, into the cache-disk. This large write finishes quickly since it requires only one seek instead of tens of seeks. As a result, the RAM buffer is very quickly made available to absorb additional requests left in the large burst. The two-level cache appears to the host as a large virtual RAM cache with a size close to the size of the cache-disk. When the data-disk is idle or less busy, it performs *destaging* operations which transfer data from the cache-disk to the data-disk.

Since the cache is a disk with a capacity much larger than a normal RAM cache, it can capture the temporal locality of I/O requests with much less cost. It is also non-volatile thus highly reliable. In addition, the cache-disk is only a cache that is transparent to the file system. DCD works at the device or device driver level. There is no need to change the underlying operating system to apply the new disk architecture. The large cache-disk enables DCD to achieve very high write performance.

1.4 Orthogonal Architectures

It is interesting to note a surprising similarity between the development of memory systems and the recent advances in disk systems. A few decades ago, computer architects proposed the concept of memory interleaving to improve memory throughput. Later, cache memories were introduced to speedup memory accesses for which interleaved memory systems were not able to do. We view the RAID systems as being similar to the interleaved memories while our *DCD* system is similar to multi-level CPU caches. Existing disk caches that use either part of main memory or dedicated RAM, however, are several orders of magnitude smaller than disks because of the significant cost difference between RAM and disks. Such caches can hardly capture the locality of I/O transfers and can not reduce disk traffic as much as a CPU cache can for main memory traffic. Therefore, traditional disk caches are not as successful as caches for main memories, particularly for writes. Our new *DCD* architecture marks a new start of caching disk using a disk that has a similar cost range as the data-disk making it possible to have the disk cache large enough to catch the data locality in I/O transfers. However, it is not easy to make one disk physically much faster than the other so that the former can become a cache as done in main memory systems. The trick is to exploit the temporal locality of I/O transfers and to make use of the idea of LFS to minimize the seek time and rotational latency that are the major part of disk access times.

2 The DCD Architecture and Operations

Figure 2 shows the structure of DCD. The disk hierarchy consists of 3 levels. At the top of the hierarchy is a small RAM buffer with the size ranging from hundreds of KB to several MB. The second level cache is a disk drive with capacity in the range of a few MB to tens of MB, called *cache-disk*. The cache-disk is a small and sequentially accessed disk that stores data in a log format. Note that the cache-disk can be a separate physical disk drive to achieve high performance as shown in Figure 2(a), or one logical disk partition physically residing on one disk drive for cost effectiveness as shown in Figure 2(b). At the bottom level is a normal disk drive, called *data-disk*, on which files reside. The data organization on this disk is a traditional, unmodified, read-optimized file system such as the UNIX Fast File System or Extended File System.

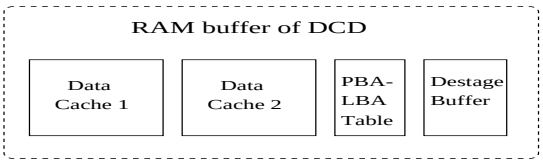

Figure 3: The structure of the DCD RAM buffer

Figure 3 shows the detailed structure of the RAM buffer of DCD, which consists of 4 components, namely two Data Caches, a PBA-LBA Table and a Destaging Buffer. The functions of these components will become clear as we discuss the operations of DCD.

2.1 Writing

Of the two data caches in the RAM buffer, only one is "active" at anytime. When a write request comes in, the DCD system first checks the size of the request. If the request is a large write, say over 64 KB or more, it is sent directly to the data-disk. Otherwise, the controller checks the free space of the active data cache. If there is enough free space to buffer the request, the controller allocates a cache block and copies the data into the cache block. As soon as the data is transferred into the cache, the controller sends an acknowledgment of "write complete" to the host. We refer to this acknowledgment as *immediate report*. The write response time in this case is only the time to transfer the data from the host computer to the data cache. The case for report after the disk transfer is complete will be discussed shortly.

If the active data cache is full, the controller turns the other data cache as active to accept new data. Meanwhile, the controller writes the entire contents of the previous active data cache into the cache-disk, in one large log format. When the log write finishes, the whole data cache is freed and ready to become active again.

Writing a large log into a disk presents significant saving in disk access time as opposed to writing each request individually. Consider a situation where the average disk block size is 8 KB and the cache size is 256 KB. When the whole contents of the cache are written into the cache-disk as a log, only one

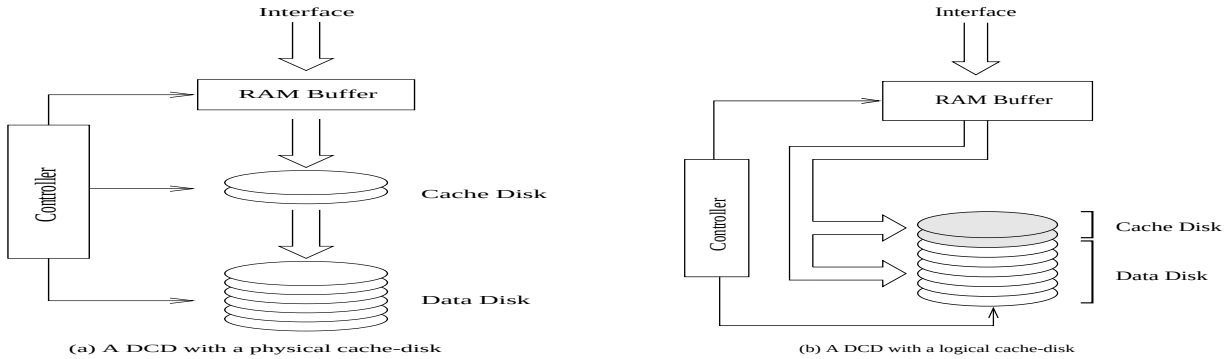

Figure 2: The Structure of DCD

large write is required, instead of 32 small writes each of which suffers from expensive seek and rotational latencies. The log write finishes quickly so that the data cache is available again to take the following write requests. Therefore the two level cache appears to the host as a large virtual RAM cache with a size close to the size of the cache-disk.

In DCD, data do not have to wait in the buffer until the buffer is full. Rather, they are written into the cache-disk whenever the cache-disk is available, even the buffer is only partially full. In other words, DCD never lets the cache-disk become idle unless the RAM buffer is empty. This policy has two important advantages. First, data are guaranteed to be written into the cache-disk when the current cache-disk access finishes. Thus, data are stored in a safe storage within tens of milliseconds on average, resulting in much better reliability than keeping data in the RAM buffer for a long time. Even in the worst case, the maximum time that data must stay in the RAM is the time needed for writing one full log, which takes less than a few hundreds of milliseconds depending on the RAM size and the speed of the disk. This situation occurs when a write request arrives right after the cache-disk starts writing a log. Another advantage is that, since data are always quickly moved from the data cache to the cache-disk, the data cache can have more available room to buffer a large burst of requests that happens very frequently in office/engineering workloads.

2.2 Reading

When a read request arrives, the DCD controller first searches the RAM buffer and the cache-disk. If the data is still in the RAM buffer then the data is immediately ready. If the data is in the cache-disk, then a read from the cache-disk is needed. If the data has already been destaged to the data-disk, the read request is sent to the data-disk. And finally, if the data is partially in the cache-disk and partially in the data-disk, then the overlapped data in the cache-disk must be first destaged to the data-disk before the request can be sent to the data-disk. We found in our simulation experiments that more than 99% of read requests are sent to the data disk. Reading from buffer or cache-disk seldom occurs. This is because most file systems use a large read cache so that most read requests for the newly written data are captured by the cache while the least recently used data are most likely to have a chance to be destaged from the cache-disk to the data disk. The read performance of the DCD is therefore similar to and some times better than that of a traditional disk because of the reduced traffic at the data disk as evidenced later in this paper.

2.3 Cache-disk Data Organization

When the host sends a request to the disk system to access a disk block, it provides a Logic Block Address, or LBA, to indicate the position of that block in the data-disk. In the case of DCD, however, the data in the request may be physically cached in the cache-disk with a different block address, referred to as Physical Block Address or PBA. The DCD system maintains a table for the mapping information between LBA's and PBA's for all blocks in the cache-disk. To speed up searching, all entries in the mapping table are in a hash table indexed by the LBAs. These entries are also in a doubly linked list, ordered with their PBA values. Each entry is 12 bytes. The total table size is about 24 KB for a 16 MB cache-disk with a cache block size of 8 KB.

Figure 4 shows the data organization in the cache-disk. As shown in the figure, each block in the cache-disk has a corresponding entry in the PBA-LBA mapping table. The holes in the cache-disk are obsolete blocks caused by invalidations of overwritten data. The PBA-LBA entries of these obsolete blocks are removed from the linked list and recycled into a free-entry list (not shown in the figure) for future use. A special pointer called Current Log Position, or CLP, is maintained to indicate the end of the last log.

The cache-disk works in a way similar to a stack. The CLP acts as the stack pointer. A new log is always "pushed" into the cache-disk by writing the log to the disk starting from CLP, and appending the corresponding PBA-LBA mapping entries for the blocks in the log to the end of the mapping list.

Similarly, the data blocks are "popped" out from the cache-disk during the destaging process. The controller reads a chunk of data backward from the CLP and writes the data into the data-disk. It then deletes the corresponding entries in the PBA-LBA mapping list and moves the CLP back to a new position. We will discuss the destaging process in more detail later.

This stack-like data organization greatly simplifies the data structures and the related algorithms. The CLP partitions the cache-disk into two separate areas, one containing log data and

the other a continuous free space. As long as there is free space, the controller can keep pushing new logs into the cache-disk without worrying about the placement of the logs. The scheme also performs well. When a log write finishes, the disk head is located above the track pointed by the CLP. The next log write can quickly start without the need of moving the disk head, unless there are other activities happened between the two log writes which move the disk head away from the CLP. Furthermore, when the destaging process starts to pop data out of the cache-disk, it may find the last log data in the RAM buffer, thus saves one or several disk read operations.

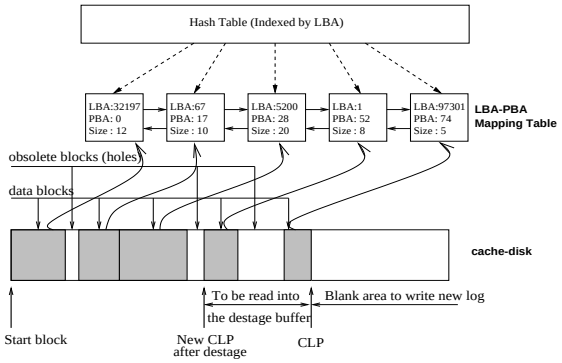

Figure 4: Data Organization in Cache-Disk

2.4 Destaging

The destaging process, which moves data from the cache-disk to the data-disk, starts when the system detects an idle period. The controller first reads a large block of data from the cache-disk into the destaging buffer. It then reorders the blocks in the destaging buffer according to their LBA numbers to reduce seek overheads, and writes the blocks into their original locations in the data-disk one by one. After a data block in the destaging buffer is written, its corresponding entry in the PBA-LBA mapping table can be invalidated to indicate that the data is not in the cache-disk anymore. The process continues until the cache-disk becomes empty. If a read or a write request comes during destaging, the destaging process is suspended until the next idle time is found.

We use a simple algorithm to detect the idle time, that is, if there is no activity in the disk system for a certain period of time (we chose a 50 ms threshold in our simulation), we consider the disk as idle and start destaging. When the cache-disk utilization approaches 60% of the total cache-disk capacity, the time threshold reduces as the disk utilization increases. In other words, the destaging algorithm works "harder" as the cache-disk fills up to prevent the cache-disk becomes full.

2.5 DCD with Report After Complete

In the previous discussion, we assumed that the DCD sends an acknowledgment of a write request as soon as the data are transferred into the RAM buffer. This scheme has excellent performance as will be shown in our simulation experiments. With only 512 KB to 1 MB RAM buffer and tens of MB cache-disk, the DCD can achieve performance close to that of a solid-state disk. The reliability of the DCD is also fairly good because data do not stay in the RAM buffer longer than a few hundreds milliseconds in the worst case, as discussed previously. If high reliability is essential, the RAM can be implemented using NVRAM for some additional cost, or using convention RAM but committing a write request as complete only after it has been actually written into the cache-disk or the data-disk. We call this a *report after complete* scheme. The performance of this configuration would be lower than that of immediate reporting because a request is reported as complete only when all requests in its log are written into a disk.

2.6 Enhanced DCD with an NVRAM cache

NVRAM can be used by DCD to improve its reliability. While the DCD architecture discussed so far works well with both RAM and NVRAM caches, better performance can be obtained by exploiting the reliability feature of NVRAM. In this subsection, we present an enhanced DCD architecture that works with an NVRAM cache.

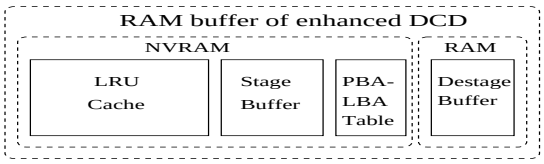

Figure 5: The RAM buffer structure of an Enhanced DCD

The differences between the enhanced DCD and the "normal" DCD discussed so far are in the RAM buffer. As shown in Figure 5, the NVRAM buffer of the enhanced DCD uses an LRU (Least Recently Used) cache and a staging buffer to replace the double data caches in the normal DCD. The size of the staging buffer is of 64 to 256 KB. The PBA-LBA mapping table is also located in the NVRAM for easy crash recovery. Because data in the destaging buffer is read from the cache-disk, the destaging buffer can use DRAM to reduce costs, without compromising the reliability.

The write operation of an enhanced DCD is similar to that of a normal DCD. However, the controller does not flush the contents of the LRU cache to the cache-disk when the cache-disk is idle. Instead, dirty data are kept in the LRU cache as long as possible in order to capture the locality of write requests. In many I/O workloads, a major portion of data are overwritten repeatedly [9], therefore keeping data in the cache longer helps capturing the overwriting and reducing the disk traffic. The reliability of the data is not a concern because of the NVRAM.

When a write request comes in and the LRU cache is full, the DCD controller copies the LRU blocks to the staging buffer until the staging buffer is full, or until there is no more LRU block left in the cache. The spaces in the LRU cache can safely be released now since the staging buffer is a part of NVRAM. Now a large portion of the LRU cache is freed so that the current write request can immediately be satisfied by allocating a free block in the cache and copying the data into it. At the same time, the whole contents of the staging buffer are written

into the cache-disk, in one large log format. When the log write finishes, the staging buffer is freed so it can take other LRU blocks from the cache if necessary. The size of the staging buffer is large enough (64 - 256 KB) so the log write is efficient.

The enhanced DCD always works in the immediate report mode because of the reliability provided by the NVRAM. Its read and destaging operations are similar to those of a normal DCD.

3 Baseline Systems

We will compare the performance of the DCD with two traditional disk systems as baseline systems. One is a disk with a built-in RAM cache, the other is a disk with a built-in RAM cache and an external NVRAM LRU cache.

3.1 Traditional Disks with Built-in Caches

The performance of "normal" DCDs with RAM buffers will be compared to traditional disks with a built-in RAM cache in the disk controller. A DCD and its traditional disk counterpart work in the same report mode — that is, they either both use the immediate report mode, or both use the report-after-complete mode. The size of the built-in RAM cache of the traditional disk is fixed at 4 MB, while the size of the RAM buffer of DCD is fixed at 512 KB.

3.2 Traditional Disks with NVRAM LRU caches

For the enhanced DCD with an NVRAM cache, we will compare its performance with a baseline system that has a built-in cache and an external NVRAM cache. The NVRAM cache uses an LRU (Least Recently Used) algorithm to manage its data. If the cache is full, the controller destages the least recently used data block to the data-disk to make room for the incoming request. We refer to this baseline system as an *LRU cached-disk*. When the system is idle, a destaging process moves dirty LRU data from the RAM cache to the disk to make room for future incoming requests. The LRU cached-disk always uses immediate-report mode because of the use of NVRAM.

We vary the NVRAM cache sizes of both DCD and the traditional LRU cached disks from 256 KB to 4 MB. For DCD, the staging buffer size is 256 KB and the destaging buffer size 64 KB, except for the case of 256 KB total NVRAM size. In the later case the sizes of the staging buffer and the destaging buffer are 96 KB and 32 KB, respectively, to leave more space for the LRU cache.

4 Performance Evaluation Methodology

4.1 Workload Characteristics

We use a set of real-world traces to carry out the simulation. The trace files are obtained from Hewlett-Packard. The traces contain all disk I/O requests made by 3 different HP-UX systems during a four-month period, and are described in detail in [9]. The three systems represent 3 typical configurations of the office/engineering environment. Among them, *cello* is a time-sharing system used by a small group of researchers (about 20) at HP laboratories to do simulation, compilation, editing and mail. *Snake* is a file server of nine client workstations with 200 users at the University of California, Berkeley. And *hplajw* is a personal workstation at HP laboratory for editing and mail.

In order to find a range of workloads with different intensities that is suitable for DCD, we overlaid multiple trace files corresponding to different days. By mixing more trace files into one new trace, we can have a higher traffic I/O workload. Similar approach has been used by Varma et. al. [10] to study the destaging algorithms for RAID caches. They overlaid up to 6 days of *cello* traces. In this study, we selected up to 9 days of the trace data for each system and overlaid them together to get a very wide variation of I/O traffic. We tried to look for a group of trace files with roughly similar file lengths (which means that they contain similar numbers of requests). In this way when we overlay 2 trace files we can double the I/O traffic. The particular traces we chose are from April 18 - 24, 1992 for *cello* and May 11 - 19, 1992 for *snake*. For *hplajw*, we are not able to find consecutive days during which all trace files have roughly similar lengths. Instead, the trace files for *hplajw* are picked from: 92-04-29, 92-04-30, 92-05-11, 92-05-15, 92-05-17, 92-05-18, 92-06-09, 92-06-10 and 92-06-12.

Overlaying multiple traces to form a single trace enables us to obtain a much greater range of workloads with different traffic rates than what can be provided by a single trace file. The lightest workload we can get now is a single day of *hplajw* which has only about 12000 requests and 63 MB of total requested data size. When we mix 9 days of *snake* traces together, however, we get a workload of over 1,030,000 requests and 7000 MB of total requested data size, which is quite busy for a single disk, especially when the burstiness is taken into account.

Using 3 different set of traces from 3 quite different systems also makes it possible for us to test the system under different I/O workloads with different characteristics such as burstiness and read/write ratio. For example, write requests dominate the *hplajw* and *snake* traces, accounting for about 67% and 60% of total requests, respectively. *Cello*, on the other hand, is a read-dominated workload, and write requests take only 37% of the total requests.

4.2 Trace-Driven Simulator

We developed a trace-driven simulation program for our performance evaluation purpose. In our previous study [11, 12] we used one of our own disk simulator which simulates an old HP C2200A. In this study we use a disk simulator developed by Kotz et. al. [13]. The simulator models an HP 97560 disk drive, which is a 5.25-inch, 1.26 GB disk with 128 KB built-in cache. HP 97560 has an average access time of 23 ms for an 8 KB data block. The disk model provides accurate and detailed simulation, including SCSI bus contention, built-in cache read-ahead and write-behind, head-skewing, etc. We have made some modifications to the simulator so it can easily simulate multiple disks simultaneously.

For physical DCDs, the program simulates two physical disk drives at the same time, one for the cache-disk and the other for the data-disk. For logical DCDs, two logical disk drives are simulated by using two disk partitions on a single physical drive. The simulator charges a controller overhead of

0.3 ms for the baseline system with an LRU cache to simulate the time of cache management. The controller overhead is set to be 0.5 ms for DCD.

The cache-disk size is assumed to be 10 MB for *hplajw* and 30 MB for *cello* and *snake*. The cache-disk sizes are determined by observing the maximum disk space needed for caching data during our simulations.

5 Numerical Results

We present the simulation results in this section. We choose the *response time* as the performance metrics because we believe it is one of the most important I/O performance parameters for office/engineering environment and other interactive environments. Users in these computing environments are more concerned with the response time than with the I/O throughput. A system here must provide a fairly short response time to its users.

5.1 Normal DCD with Immediate Report

Figures 6, 7 and 8 show the average I/O response times of DCD and the traditional disk. Both DCD and the traditional disk use the immediate-report mode. For easy comparison, we draw the write response times in solid lines and the read response times in dotted lines. In addition, the response times of the baseline systems are in thick lines while those of the DCDs are in thin lines.

Although the traditional disk has a built-in cache of 4 MB and uses the immediate-report mode, its performance is far from satisfactory. The average write response times are hundreds of milliseconds most of time, implying long waiting queues. One reason for the poor performance is that the built-in cache of HP 97560 uses a simple non-LRU algorithm to manage write data. Since the cache is made of RAM, dirty data should be written into the disk as soon as possible therefore the LRU algorithm can not be used.

The DCD systems, on the other hand, show significantly better write performance. For *hplajw* and *snake*, the average write performance of a DCD is 1 to 2 orders of magnitudes better than that of a traditional disk. The average write response times of a DCD are close to those of a solid-state disk (about 1–2 ms). Their write response time curves are very close to the X-axis. For *cello*, which is a read-intensive workload, the performance improvement of DCD is relatively small because the system is busy for reads most of time. Still, a logical DCD are about 2–5 times faster than a traditional disk for writes. A physical DCD performs even better.

While the main purpose of DCD is to improve the write performance, the simulation results show that a DCD also has better read performance. The improvement in read performance of a DCD becomes greater as the workload increases. For example, a DCD has similar or slightly better read performance than a baseline system for a single trace. For 2–9 overlaid traces, a DCD shows 2–10 times better performance. Such a performance improvement can mainly be attributed to the reduction of write traffic at the data-disk. Since most write requests are removed from the critical path and will not be written into the data disk until the system is idle, the data-disk has more available time for processing read requests. The improvement of read performance is important to the overall system performance, since most read requests are synchronized.

5.2 Normal DCD with Report After Complete

DCD with report after complete has good reliability because a write is guaranteed to be stored in a disk before the CPU is acknowledged. If the RAM buffer is a volatile memory, this scheme is much more reliable than the immediate report scheme, but it may not perform as well because a request is acknowledged as complete only after all requests in its group (i.e., the log) are written into a disk. Nevertheless, the DCD still demonstrates superb performance as shown in Figures 9, 10 and 11.

For *hplajw* and *cello*, a logical DCD is about 2 times faster than a traditional disk for most workloads in terms of average write times. A physical DCD has even better performance, especially for high workloads. DCD also shows faster read response times than a traditional disk.

For *snake*, a DCD performs slightly better than a traditional disk for the single trace workload. When the number of overlaid traces increases, the performance of the traditional disk degrades rapidly because of the increased contention for the disk bandwidth between read and write requests. The DCD, on the other hand, shows much better performance at high workloads, because write requests are removed from the critical path and processed only when the system is idle. For example, for the workload of 9 overlaid traces, a logical DCD shows an average write response time 4 times shorter than that of a traditional disk, and an average read response time 6 times shorter. A physical DCD shows 20 times better performance in terms of write response time and 14 times better performance in terms of read response time.

5.3 Enhanced DCD with NVRAM

When NVRAM is used as caches, data reliability in the cache is not a problem anymore. As a result, more sophisticated data management algorithms such as LRU can be used in both DCD and the traditional disk system to manage write data for better performance. Figures 12 to 17 compare the average write and read times of enhanced DCD with that of LRU-cached baseline systems. We varied the RAM buffer sizes between 256 KB and 4 MB. Because of the space limitation, we show only the results of 1, 3, 6 and 9 overlaid traces for each workload.

Compared to a baseline disk with a large built-in non-LRU RAM cache, the baseline disk with an LRU NVRAM cache shows much better performance. The improvement mainly comes from the LRU algorithm that is much more efficient than the simple algorithm used by the disk built-in cache. However, a DCD system still does a much better job than a baseline disk with an LRU NVRAM cache. The average write response times of DCD are significantly lower than those of the baseline system — 2 to 20 times less for small RAM sizes (256 – 1024KB). Increasing the RAM size for the baseline system always notably improves its performance. Conversely, for *hplajw* and *snake*, increasing the amount of RAM for DCD beyond a threshold, which is between 512 KB and 1024 KB, has almost no effect on the write performance. A DCD with a RAM size around the threshold performs close to or better than an LRU cached disk with 4 MB of RAM. Thus we believe DCD utilizes the RAM space very efficiently. For *cello*, which has a large percentage

of read requests, there is no such a clear threshold. Still, the DCD performs significantly better than the LRU-cached disk with the same RAM size.

All three traces used in our simulation are relatively lightly-loaded, even when we overlaid multiple traces together. A DCD with a RAM cache of 512 KB to 1 MB can absorb almost all write requests. A LRU cached disk with a 4 MB cache can also absorb almost all writes. As a result, if both the DCD and the LRU cached disk have a cache of 4 MB, they perform almost equally well. In such a case a DCD system does not show obvious advantages over a traditional system. The exception is *cello*, for which the write response time of a DCD with a 4 MB cache is only 50% of that of a LRU cached disk with a same-size cache.

An enhanced DCD uses the LRU algorithm to keep active data in the NVRAM to capture data overwriting. Compared to a "normal DCD", disk traffic in an enhanced DCD is reduced, resulting in better write and especially better read performance. Occasionally a normal DCD slightly outperforms the enhanced DCD for writes for the following reasons. Since an enhanced DCD keeps active data in the NVRAM, the RAM buffer is seldom empty. When a large write burst comes, the NVRAM cache may not have enough space, and the DCD has to evict data from the NVRAM cache to the cache-disk before accepting new data. However, on average, an enhanced DCD does a better job for the same RAM size than a normal DCD.

5.4 Overhead of DCD

While DCD shows superb performance, for each data write, DCD has to do extra work compared to a traditional disk. It has to write the data into the cache-disk first and read it back from the cache-disk later. In a logical DCD these extra operations compete with normal data reads and writes for the disk bandwidth. Therefore one would expect performance degradation at high loads. However, we found that while a logical DCD does show some performance degradation at high loads, it still performs much better than a baseline system.

In DCD, all writes to and reads from the cache-disk are performed in large sizes — typically up to 32 requests can be written into the cache-disk from the buffer, and 8 requests can be read from the cache disk into the destaging buffer, both in one disk access. Therefore the overhead per write access increases only slightly. Moreover, this small overhead can be compensated by the fact that data can stay in the cache-disk longer because the cache-disk is larger than the RAM cache. As a result, the data in the cache-disk has a better chance to be overwritten giving less number of destaging operations. Our simulation shows that the disk in a logical DCD system sees slightly lower write traffic and slightly higher read traffic compared to a LRU-cached disk. The overall disk traffic in the logical DCD is slightly *lower* (about 1 percent) than in the baseline system for *hplajw* and *snake*, and is only about 0.6% higher for *cello*.

5.5 Cost Considerations

Although the DCD architecture improves I/O performance, it also introduces an additional cost to the traditional disk system. One immediate question is whether such additional cost is justified.

Currently the cost of 1 MB disk space is about 5 cents. As a result, the additional 10 to 50 MB of disk space used by a logical DCD is a very small fraction of a modern disk drive, which is typically around several GB or larger. On the other hand, we have shown that DCD uses a significant less amount of NVRAM than a baseline disk needs for similar performance. This represents significant cost reductions because NVRAM is extremely expensive.

If a physical cache-disk is to be implemented for high traffic disk systems such as file servers, the cost will be high because the smallest hard drive that is available in the market has a few hundreds MB capacity and the cost is around a couple of hundreds dollars. However, the cost can be amortized if a cache-disk is used to serve several data-disks. Moreover, many systems have two or more disks. Typically not all disks in a system are busy at the same time. We can divide each disk in a system into a cache-partition and a data-partition. When the system accesses a data-disk, the DCD chooses a cache-partition in another disk that is idle or less busy to act as a logical cache-disk. Such a system should perform closely to a physical DCD, while its cost is close to that of a logical DCD.

6 Conclusions

We have proposed a new disk architecture called Disk Caching Disk, or *DCD* for short, for the purpose of improving write performance in the most-widely-used office/engineering environment. The basic idea of the new architecture is to exploit the temporal locality of disk accesses and the dramatic difference in data transfer rate between large and small disk transfer sizes. The DCD is a hierarchical architecture consisting of three levels: a RAM buffer, a cache-disk which stores data in a log format, and a data-disk that stores data in the same way as traditional disks. The disk cache including the RAM and the cache-disk is transparent to the operating system so that there is no need to change the operating system to incorporate this new disk architecture.

We studied several architectural variations of DCD. In a *logical DCD*, the cache-disk is implemented as a logical partition of the data disk for low cost. In a *physical DCD*, the cache-disk is a dedicated small disk drive for better performance. For a DCD with a RAM cache, it can use either the *immediate report mode* for higher performance, or use *report after complete mode* for higher reliability. We also proposed an *enhanced DCD* organization that uses an LRU algorithm to manage its NVRAM buffer.

Extensive simulation experiments have been carried out by using traces representing 3 typical office/engineering workload environments. Multiple traces are overlaid together to form a single workload to test the systems under different I/O intensities. Numerical results have shown that the new DCD architecture is very promising in improving write performance across a great range of workloads. With immediate report, the *DCD* improves write performance by one to two orders of magnitude over the traditional disk systems with a large built-in RAM cache. A factor of 2 to 20 performance improvements over traditional disks are observed for the DCD with the report-after-complete scheme. An enhanced DCD with an NVRAM cache of 512 – 1024 KB also outperforms a traditional disk with an external NVRAM cache of 4096 KB most of time. It is noted that the DCD also improves read performance in many

cases. The additional cost introduced by the *DCD* is a small fraction of the disk system cost. *DCD* can be implemented as a device-driver in the host system, as a stand-along system including the controller, the cache-disk and the data-disk. It is also possible to implement a logical DCD at the disk-controller level.

As future work, we are currently implementing the DCD concept on two platforms, one is a device driver for Solaris 2.6, and the other is a DCD SCSI drive controlled by a PC. Both systems are functioning. Measured performance results are very promising. For example, when running real-world programs that are I/O intensive, our Solaris DCD device driver outperforms the traditional disk device driver by a factor of 2–5 in terms of program execution times. Further measurement and optimization are underway.

Acknowledgments

This research is supported in part by NSF under grants MIP-9505601 and MIP-9714370. The authors would like to thank Dr. Wilkes of Hewlett-Packard for providing trace files to us, and thank Dr. Kotz of Dartmouth College for letting us use his disk simulator. Mr. Changsheng Xie suggested the idea of using a cache-partition in another disk as a logical cache-disk for better performance. Tycho Nightingale implemented the Solaris DCD device driver, and Jason Lu implemented the prototype hardware system.

References

[1] 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.

[2] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer, "Non-volatile memory for fast, reliable file systems," in *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*, (Boston, MA), pp. 10–22, ACM Press , New York, NY , USA, Oct. 1992.

[3] M. Rosenblum and J. Ousterhout, "The design and implementation of a log-structured file system," *ACM Transactions on Computer Systems*, pp. 26 – 52, Feb. 1992.

[4] J. Ousterhout and F. Douglis, "Beating the I/O bottleneck: A case for log-structured file systems," tech. rep., Computer Science Division, Electrical Engineering and Computer Sciences, University of California at Berkeley, Oct. 1988.

[5] K. Treiber and J. Menon, "Simulation study of cached RAID5 designs," in *Proceedings of Int'l Symposium on High Performance Computer Architectures*, (Raleigh, North Carolina), pp. 186–197, Jan. 1995.

[6] M. Seltzer, K. Bostic, M. K. McKusick, and C. Staelin, "An implementation of a log-structured file system for UNIX," in *Proceedings of Winter 1993 USENIX*, (San Diego, CA), pp. 307–326, Jan. 1993.

[7] U. Vahalia, *UNIX Internals — The New Frontiers*. Prentice Hall, 1996.

[8] D. Stodolsky, M. Holland, W. V. Courtright II, , and G. A. Gibson, "Parity logging disk arrays," in *ACM Transaction of Computer Systems*, pp. 206–235, Aug. 1994.

[9] C. Ruemmler and J. Wilkes, "UNIX disk access patterns," in *Proceedings of Winter 1993 USENIX*, (San Diego, CA), pp. 405–420, Jan. 1993.

[10] A. Varma and Q. Jacobson, "Destage algorithms for disk arrays with non-volatile caches," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, (Santa Margherita Ligure, Italy), pp. 83–95, June 22–24, 1995.

[11] Y. Hu and Q. Yang, "DCD—disk caching disk: A new approach for boosting I/O performance," in *Proceedings of the 23rd International Symposium on Computer Architecture*, pp. 169–178, May 1996.

[12] Q. Yang and Y. Hu, "System for destaging data during idle time by transferring to destage buffer, marking segment blank, reordering data in buffer, and transferring to beginning of segment." U.S. Patent No. 5,754,888, May 1998.

[13] D. Kotz, S. B. Toh, and S. Radhakrishnan, "A detailed simulation model of the HP 97560 disk drive," Tech. Rep. PCS-TR94-220, Department of Computer Science, Dartmouth College, July 1994.

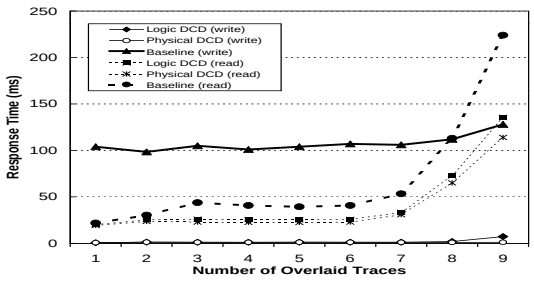

Figure 6: Normal DCD vs Traditional Disk for *hplajw* (Immediate Report)
DCD buffer = 512 KB; Baseline cache = 4 MB

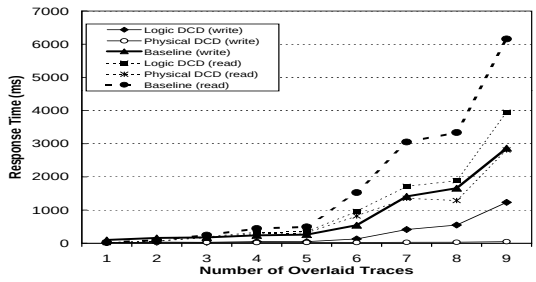

Figure 8: Normal DCD vs Traditional Disk for *cello* (Immediate Report)
DCD buffer = 512 KB; Baseline cache = 4 MB

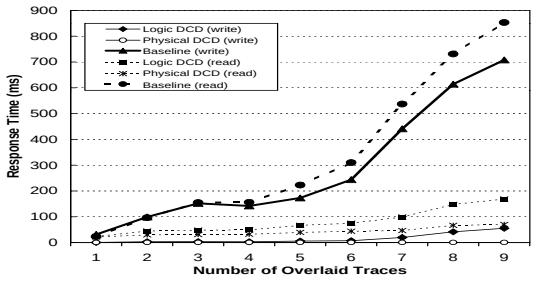

Figure 7: Normal DCD vs Traditional Disk for *snake* (Immediate Report)
DCD buffer = 512 KB; Baseline cache = 4 MB

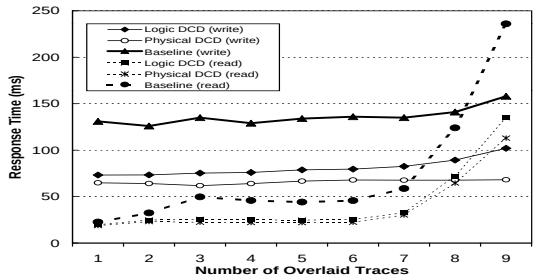

Figure 9: Normal DCD vs Traditional Disk for *hplajw* (Report After Complete)
DCD buffer = 512 KB; Baseline cache = 4 MB

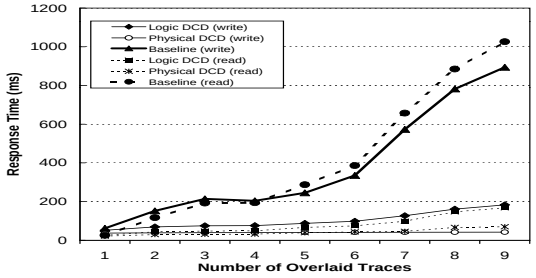

Figure 10: Normal DCD vs Traditional Disk for *snake* (Report After Complete)
DCD buffer = 512 KB; Baseline cache = 4 MB

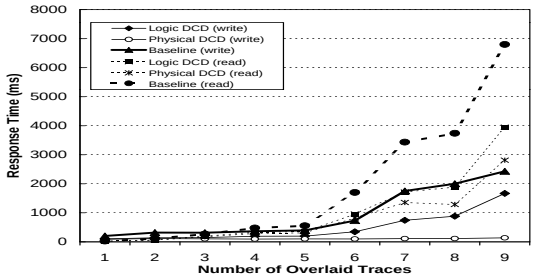

Figure 11: Normal DCD vs Traditional Disk for *cello* (Report After Complete)
DCD buffer = 512 KB; Baseline cache = 4 MB

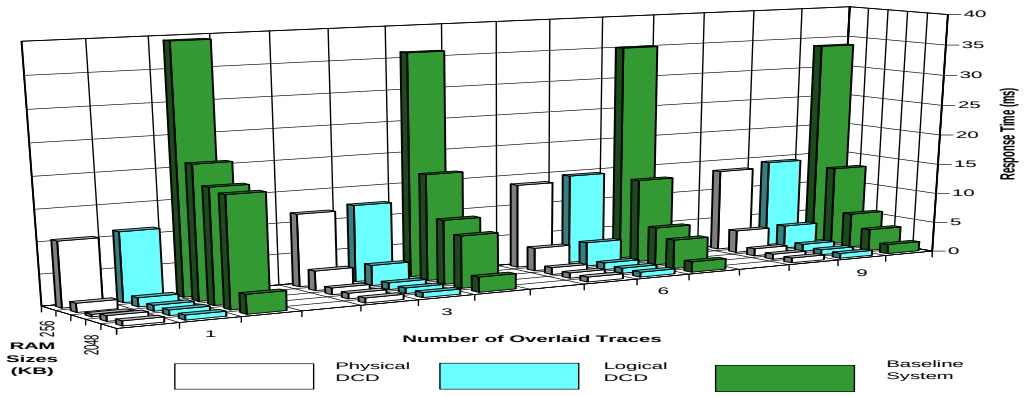

Figure 12: *HPLAJW* Write Response Time (Enhanced DCD vs LRU Cached Disk)

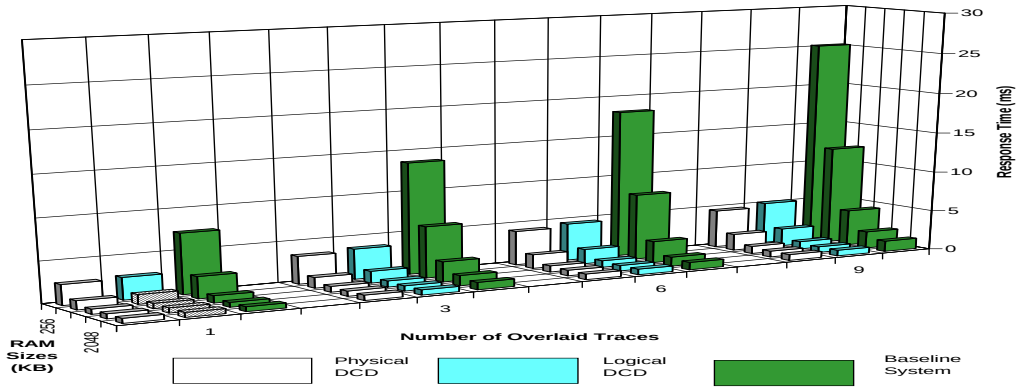

Figure 13: *SNAKE* Write Response Time (Enhanced DCD vs LRU Cached Disk)

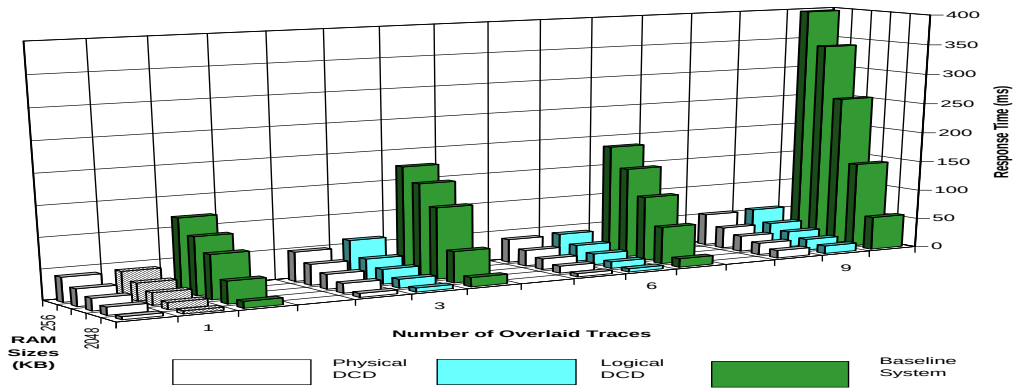

Figure 14: *CELLO* Write Response Time (Enhanced DCD vs LRU Cached Disk)

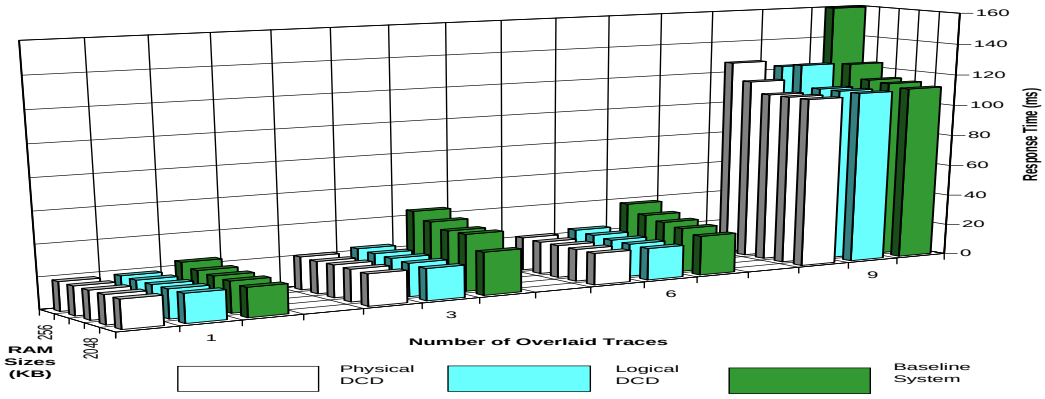

Figure 15: *HPLAJW* Read Response Time (Enhanced DCD vs LRU Cached Disk)

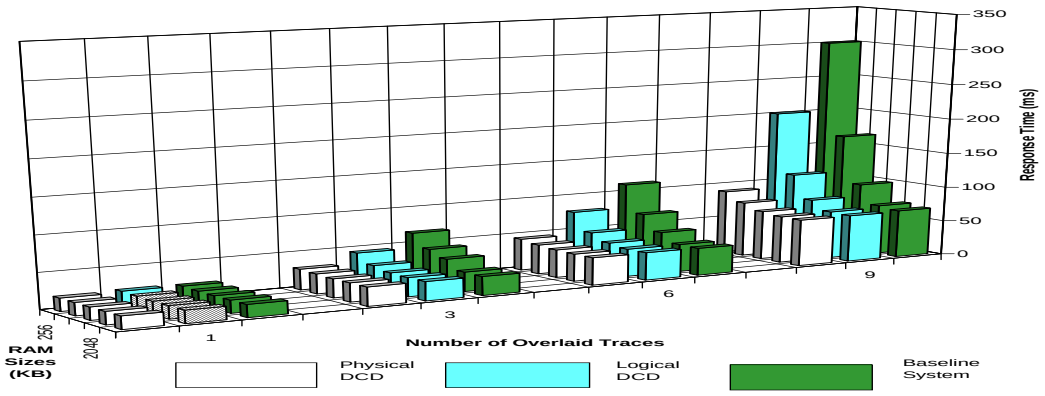

Figure 16: *SNAKE* Read Response Time (Enhanced DCD vs LRU Cached Disk)

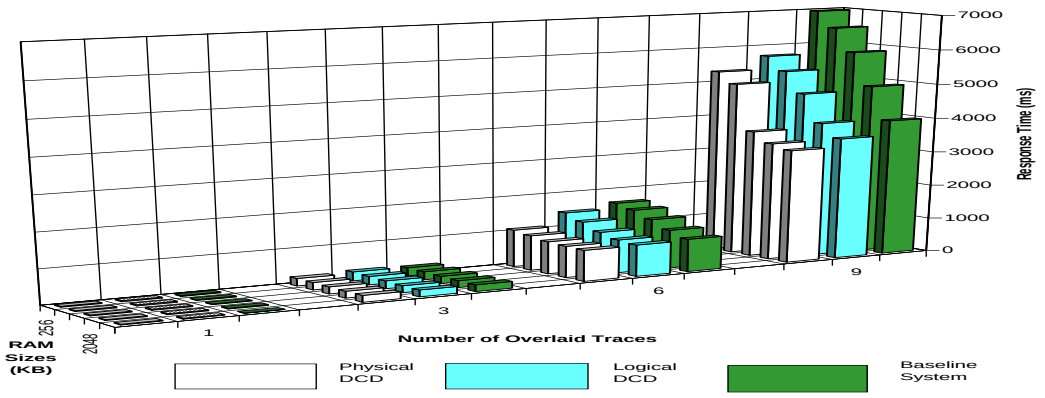

Figure 17: *CELLO* Read Response Time (Enhanced DCD vs LRU Cached Disk)