This is a paper to be presented in ISCA96 This paper presents a novel disk storage architecture called *DCD*, Disk Caching Disk, for the purpose of optimizing I/O performance. The main idea of the *DCD* is to use a small log disk, referred to as *cache-disk*, as a secondary disk cache to optimize write performance. While the cache-disk and the normal data disk have the same physical properties, the access speed of the former differs dramatically from the latter because of different data units and different ways in which data are accessed. Our objective is to exploit this speed difference by using the log disk as a cache to build a reliable and smooth disk hierarchy. A small RAM buffer is used to collect small write requests to form a log which is transferred onto the cache-disk whenever the cache-disk is idle. Because of the temporal locality that exists in office/engineering work-load environments, the *DCD* system shows write performance close to the same size RAM (i.e. solid-state disk) for the cost of a disk. Moreover, the cache-disk can also be implemented as a logical disk in which case a small portion of the normal data disk is used as the log disk. Trace-driven simulation experiments are carried out to evaluate the performance of the proposed disk architecture. Under the office/engineering work-load environment, the *DCD* shows superb disk performance for writes as compared to existing disk systems. Performance improvements of up to two orders of magnitude are observed in terms of average response time for write operations. Furthermore, *DCD* is very reliable and works at the device or device driver level. As a result, it can be applied directly to current file systems without the need of changing the operating system. This paper presents a novel disk storage architecture called *DCD*, Disk Caching Disk, for the purpose of optimizing I/O performance. The main idea of the *DCD* is to use a small log disk, referred to as *cache-disk*, as a secondary disk cache to optimize write performance. While the cache-disk and the normal data disk have the same physical properties, the access speed of the former differs dramatically from the latter because of different data units and different ways in which data are accessed. Our objective is to exploit this speed difference by using the log disk as a cache to build a reliable and smooth disk hierarchy. A small RAM buffer is used to collect small write requests to form a log which is transferred onto the cache-disk whenever the cache-disk is idle. Because of the temporal locality that exists in office/engineering work-load environments, the *DCD* system shows write performance close to the same size RAM (i.e. solid-state disk) for the cost of a disk. Moreover, the cache-disk can also be implemented as a logical disk in which case a small portion of the normal data disk is used as the log disk. Trace-driven simulation experiments are carried out to evaluate the performance of the proposed disk architecture. Under the office/engineering work-load environment, the *DCD* shows superb disk performance for writes as compared to existing disk systems. Performance improvements of up to two orders of magnitude are observed in terms of average response time for write operations. Furthermore, *DCD* is very reliable and works at the device or device driver level. As a result, it can be applied directly to current file systems without the need of changing the operating system.

DCD—-Disk Caching Disk: A New Approach for Boosting I/O Performance

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

This paper presents a novel disk storage architecture called *DCD*, Disk Caching Disk, for the purpose of optimizing I/O performance. The main idea of the *DCD* is to use a small log disk, referred to as *cache-disk*, as a secondary disk cache to optimize write performance. While the cache-disk and the normal data disk have the same physical properties, the access speed of the former differs dramatically from the latter because of different data units and different ways in which data are accessed. Our objective is to exploit this speed difference by using the log disk as a cache to build a reliable and smooth disk hierarchy. A small RAM buffer is used to collect small write requests to form a log which is transferred onto the cache-disk whenever the cache-disk is idle. Because of the temporal locality that exists in office/engineering work-load environments, the *DCD* system shows write performance close to the same size RAM (i.e. solid-state disk) for the cost of a disk. Moreover, the cache-disk can also be implemented as a logical disk in which case a small portion of the normal data disk is used as the log disk. Trace-driven simulation experiments are carried out to evaluate the performance of the proposed disk architecture. Under the office/engineering work-load environment, the *DCD* shows superb disk performance for writes as compared to existing disk systems. Performance improvements of up to two orders of magnitude are observed in terms of average response time for write operations. Furthermore, *DCD* is very reliable and works at the device or device driver level. As a result, it can be applied directly to current file systems without the need of changing the operating system.

1 Introduction

Current disk systems generally use caches to speed up disk accesses. Such disk caches reduce read traffic more effectively than write traffic as shown in [1, 2, 3, 4]. As the RAM size increases rapidly and more read requests are absorbed, the proportion of write traffic seen by disk systems will dominate disk traffic and could potentially become a system bottleneck. In addition, small write performance dominates the performance of many current file systems such as on-line transaction processing [5] and office/engineering environments [3]. Therefore, write performance is essential to the overall I/O performance.

The purpose of this paper is to present a novel disk subsystem architecture that improves the average response time for writes by one to two orders of magnitude in an office and engineering workload environment without changing the existing operating system.

1.1 Background

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

Because of the mechanical nature of magnetic disks, the performance of disks has increased only gradually in the past. One of the most important architectural advances in disks is the RAID (Redundant Array of Inexpensive Disks) architecture pioneered by a group of researchers in UC Berkeley [2]. The main idea of the RAID is to use multiple disks in parallel to increase the total I/O bandwidth which scales with the number of disks. Multiple disks in a RAID can service a single logical I/O request or support multiple independent I/Os in parallel. Since the size and the cost of disks drop rapidly, RAID is a cost effective approach to high I/O performance. One critical limitation of the RAID architecture is that their throughput is penalized by a factor of four over nonredundant arrays for small writes which are substantial and are becoming dominant portion of typical I/O workloads. The penalty results from parity calculation for new data, which involves readings of old data and parity, and writings of new data and parity. Stodolsky et al. [5] proposed a very interesting solution to the small-write problem by means of parity logging. They have shown that with minimum overhead, parity logging eliminates performance penalty caused by the RAID architectures for small writes.

The RAID architectures are primarily aimed for high throughput by means of parallelism rather than reducing access latency. Especially for low average throughput workload such as office/engineering environment, performance enhancement due to RAID is very limited [6, 7]. Caching is the main mechanism for reducing response times. Since all write operations need eventually be reflected on a disk, volatile cache may pose a reliability problem. Nonvolatile RAM (NVRAM) can be used to improve disk performance, particularly write performance [1, 8]. However, because of the high cost of nonvolatile RAMs, the write buffer size is usually very small compared to disk capacity. Such a small buffer gets filled up very quickly and can hardly catch the locality of large I/O data. Increasing the size

of nonvolatile cache is cost-prohibitive making it infeasible for large I/O systems.

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 the file systems. The most important work in file systems is the Log-structured File System (LFS) [4, 3, 9]. The central idea of an LFS is to improve write performance by buffering a sequence of file changes in a cache and then writing all the modifications to the disk sequentially in one disk operation. As a result, many small and random writes of the traditional file system are converted into a large sequential transfer in the log structured file system. In this way, the random seek times and rotational latencies associated with small write operations are eliminated 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 successful since it was introduced more than eight years ago. Applications of LFS are mainly limited to academic research such as Sprite LFS [3], BSD-LFS [9] and Sawmill [10]. This is because LFS requires significant changes in the operating system, needs a high cost cleaning algorithm, and is much more sensitive to disk capacity utilization than traditional file systems [3, 4]. The performance 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 in order to write into disk later as a log, which may cause reliability problems.

There are several other approaches such as log-structured array [11], Loge [12], and Logical Disk approach [13]. The Logical Disk approach improves the I/O performance by working at the interface between the file system and the disk subsystem. It separates file management from disk management by using logical block numbers and block lists. Logical Disk hides the details of disk block organization from the file system, and can be configured to implement LFS with only minor changes in operating system code. However, the Logical Disk approach requires a large amount of memory, about 1.5 MB for each GB of disk, to keep block mapping tables. Moreover, the mapping information is stored in main memory giving rise to reliability problems.

1.2 Our New Approach

From the above discussion, it is clear that caching is the main mechanism for reducing access latency. But caching has not been as effective as expected so far because of large data sizes and small cache sizes. For write accesses, caching is even more expensive due to the high cost of nonvolatile RAMs. It is also clear that a log structured file system can reduce access time significantly. It is shown in [5] that the data transfer rate in unit of tracks is almost eight times faster than in unit of blocks. Even faster data transfer rate can be achieved if the transfer unit is larger. Based on this observation, we propose a new disk organization referred to as *disk caching disk* or *DCD* for short. The fundamental idea behind the *DCD* is to use a log disk, called *cache-disk*, as an extension of RAM cache to cache file changes and to destage the data to the *data disk* afterward when the system is idle. Small and random writes are first buffered in a small RAM buffer. Whenever the cache-disk is idle, all data in the RAM buffer are written, in one data transfer, into the cache-disk which is located between the RAM buffer and the data disk. As a result, the RAM buffer is quickly made available for additional requests so that the two level cache appears to the host as a large RAM. When the data disk is idle, a *destage* operation is performed, in which the data is transferred from the cache-disk to the normal data disk. Since the cache is a disk with a capacity much larger than RAM, it can capture the temporal locality of I/O file transfers; it is also highly reliable. In addition, the log disk is only a cache which is transparent to the file system. There is no need to change the underlying operating system to apply the new disk architecture. Our trace-driven simulation experiments show that *DCD* improves write performance over traditional disk systems by two orders of magnitude for a small amount of additional cost. Furthermore, less cost is possible if the idea is implemented on the existing data disk with a fraction of the disk space used as the logical cache-disk.

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 a 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 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 RAMs 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 log structured file systems to minimize the seek time and rotational latency which are the major part of disk access times.

1.3 Paper Organization

In the next section, we will describe in detail how the *DCD* architecture is organized and how it works. Section 3 presents performance evaluation methodology and workload characteristics. Numerical results obtained from our trace-driven simulation are presented in Section 4. We will also evaluate and compare the performance of our DCD with traditional disk architectures in this section. Previous research as related to our work is discussed in Section 5. We conclude the paper in Section 6.

2 The DCD Architecture

The structure of the *DCD* is shown in Figure 1. The disk hierarchy consists of 3 levels. At the top of the hierarchy is a RAM buffer with the size ranging from hundreds of kilobytes to 1 megabytes. 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 sequential access disk or log disk. Note that the cache-disk can be a separate physical disk drive to achieve high performance as shown in Figure 1, or one logical disk partition physically residing on one disk drive or on a group of disk drives for cost effectiveness as shown in Figure 2. At the bottom level, the disk is the normal data disk drive in which files reside. The data organization on this disk is a

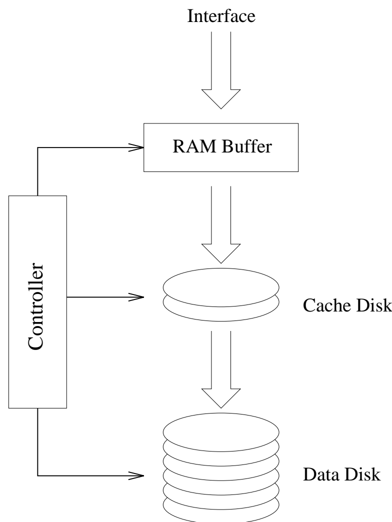

Figure 1: A DCD consisting of 2 physical disks

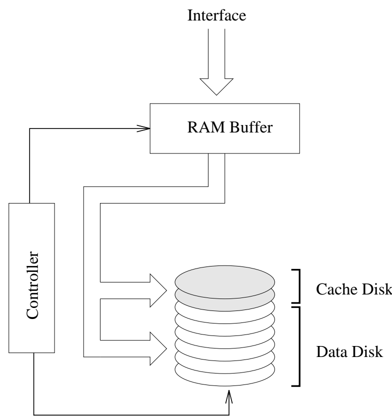

Figure 2: A DCD consisting of 2 logical disks

traditional, unmodified, read-optimized file system such as the UNIX Fast File System or Extended File System.

2.1 Writing

Upon a disk write, the controller 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 sends the request to the RAM buffer that buffers small writes from the host and to form a log of data to be written into the cache-disk. As soon as the data are transferred to the RAM buffer, the controller sends an acknowledgement of "write complete" to the host, referred to as *immediate report*. The case for report after disk transfer is complete will be discussed shortly. The data copy in the RAM is then sent to the cache-disk to ensure that a reliable copy resides on the cache-disk if the cache-disk is not busy with writing a previous log or reading. Since the disk head of the log disk is usually positioned on an empty track that is available to write a log, called Current Log Position (CLP), seeking is seldom necessary except for the situation

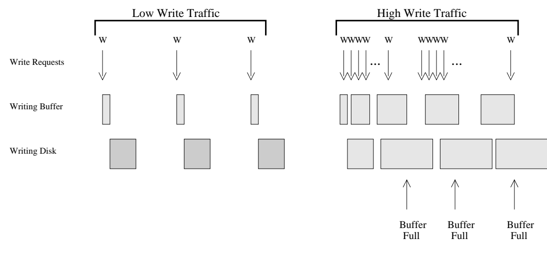

Figure 3: Timing relationship between log collecting and writing

where the log transfer occurs when the log disk is being read or destaged. The write can start immediately after the rotation latency. While the writing is in progress on the cache-disk, the controller continues to collect incoming write requests, putting them into the RAM buffer, combining them to form a large log, and committing them as finished immediately after the data transfer is finished. When the cache-disk finishes writing, the new large log is written immediately into the cache-disk again, and another round of collecting small write requests to a large log starts.

One important feature here is that data do not wait in the buffer until the buffer is full. Rather, they are written into the cache-disk whenever the cache-disk is available. In other words, DCD never lets the cache-disk become idle as long as there are write requests coming or in the buffer queue. This policy has two important advantages. First, data are guaranteed to be written into the cache-disk when the current disk access finishes. Thus, data are stored in a safe storage within tens of milliseconds on average resulting in much better reliability than other methods that keep 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 milliseconds depending on the RAM size and the speed of the disk. This situation occurs when a write request arrives just when the cache-disk starts writing a log. Another advantage is that, since data are always quickly moved from the RAM buffer to the cache-disk, the RAM buffer can have more available room to buffer a large burst of requests which happens very frequently in office/engineering workload.

Although seek times are eliminated for most write operations on the cache-disk, at the beginning of each log write there is on average a half-revolution rotation latency. Such rotation latency will not cause severe performance problem because of the following reasons. In case of low write traffic, the log to be written on the cache-disk is usually small making the rotation time a relatively large proportion. However, such large proportion does not pose any problem because the disk is in idle state most of time due to the low write traffic. In case of high write traffic, the controller is able to collect a large amount of data to form a large log. As a result, the rotation latency becomes a small percentage of the log to be written and is negligible. Therefore, the DCD can dynamically adapt to the rapid change of write traffic and perform very smoothly. Figure 3 shows the timing relationship between log collecting and log writing. From this figure, we can see that the total throughput of the DCD will not be affected by the above delay. At low load, the cache-disk has enough time to write logs as shown in the left hand part of the figure. At high load, on the other hand, the cache-disk continuously writes logs that fill the RAM buffer as shown on the right hand part of Figure 3.

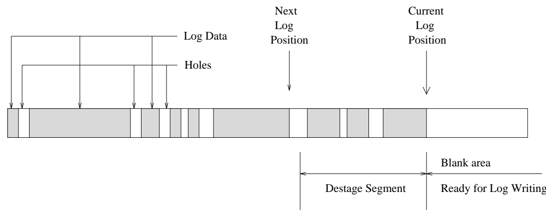

Figure 4: Destage algorithm

2.2 Reading

Read operations for the DCD are straightforward. When a read access arrives, the 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 seek operation to the data track is needed. If the data has already been destaged to the data disk, the read request is 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 traditional disk because of reduced traffic at the data disk as evidenced later in this paper.

2.3 Destaging

The basic idea of the DCD is to use the combination of the RAM buffer and the cache-disk as a cache to capture both spatial locality and temporary locality. In other words, the RAM buffer and the cache-disk are used to quickly absorb a large amount of write quests when the system is busy. Data are moved from the cache-disk to the data disk when the system is idle or less busy. Since the destage process competes with disk accesses, a good algorithm to perform data destaging is important to the overall system performance.

Since a disk is used as the disk cache, we have a sufficiently large and safe space to temporarily hold newly written data. In addition, we observed from traces under the office/engineering environment that write requests show bursty patterns as will be discussed shortly in Section 3. There is usually a long period of idle time between two subsequent bursts of requests. We therefore perform the destage only at idle time so that it will not interfere with incoming disk accesses. There are several techniques to detect idle time in a disk system [14]. In our experiment, we use a simple time interval detector. If there is no incoming disk access for a certain period of time (50 ms in our simulation), we consider the disk as idle and start destaging.

A Last-Write-First-Destage (LWFD) destage algorithm was developed for the DCD. When the idle detector finds the system idle, the LWFD algorithm is invoked by reading back a fixed length of data called a *destage segment* from the cache-disk to a destage buffer. The length of the destage segment is normally several tracks starting from the CLP (current log position). As shown in Figure 4, among data logs there may be holes that are caused by data overwriting. The LWFD will eliminate these holes and pack the data when destaging is performed. The data are re-ordered in the destage buffer and written back to the data disk to their corresponding physical locations. If a read or a write request comes during destaging, the destaging process is suspended until the next idle time is found. After destaging, destage segment on the cache-disk will be marked as blank; the CLP is moved back to a new CLP point; and the next round of destaging starts until all data on the cache-disk are transferred to the data disk and the cache-disk becomes empty.

The LWFD algorithm has two distinct advantages. First of all, after reading the destage segment and writing them to the data disk, the destage segment in the cache-disk is marked as blank and the CLP can be moved back. The disk head is always physically close to the blank track either right on it or several tracks away. When a new write request comes, the disk head can start writing very quickly. Secondly, in many cases especially for small or medium write bursts, the latest writes are still in the RAM buffer. The LWFD picks up current destage segment from the buffer rather than from the cache-disk when it starts destaging. The corresponding segment in the cache-disk is marked as blank. In this way, read frequency from the cache-disk is reduced for destaging.

2.4 DCD with Report After Complete

In the previous discussion, we assumed that the DCD sends an acknowledgement of a write request as soon as the data are transferred into the RAM buffer. This scheme shows excellent performance as shown in our simulation experiment. 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 for a long time as discussed previously. If high reliability is essential, the RAM can be implemented using a nonvolatile memory for some additional cost, or using convention RAM but committing a write request as complete only after it has been actually written into a disk similar to the traditional 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.

2.5 Crash Recovery

Crash recovery for *DCD* is relatively simple. Since data are already saved as a reliable copy on the cache-disk, only the mapping information between the PBA (Physical Block Address) in the data disk and the LBA (Log Block Address) in the cache-disk needs to be reconstructed. One possible way to do this is to keep an additional summary sector for each log write. The additional summary sector which contains the information about the changes of the mapping table is written as the first block of the current log, similar to the technique used in the Log-structured File System. During a crash recovery period, the whole cache-disk is scanned and the mapping table is reconstructed from the summary sectors. The capacity of the cache-disk is very small compared to that of data disk, so the recovery can be done very quickly. Another possible way is to keep a compact copy of the mapping table in the NVRAM making crash recovery much easier. The size of NVRAM needed for this information is from tens of kilobyte to hundreds of kilobyte, depending on the size of the cache-disk.

3 Performance Evaluation Methodology

3.1 Workload Characteristics

The performance of an I/O system is highly dependent on the workload environment. Therefore, correctly choosing the workload is essential to performance evaluation. There are basically three types of I/O workloads that are common in practice as outlined below.

The general office/engineering environment is the most typical workload environment, and is considered by some researchers as the most difficult situation to handle [3]. In this environment, disk traffic is dominated by small random file read and write requests [3]. Two important characteristics of this environment are bursty requests and a low average request rate. When there is a request, it is usually accompanied by a cluster of requests in a short time frame. We define this bursty request pattern as the *temporal locality* of disk requests. It is common to find more than 5 write requests waiting in a queue and the maximum queue length goes up to 100 and even 1000 [15]. 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.

In addition, there is usually a relatively long period of idle time between two consecutive request bursts. Ruemmler and Wilkes [15] found that in three UNIX file systems (*cello* for timesharing, *snake* for file server and *hplajw* for workstation), the average disk access rate is as low as 0.076/second (*hplajw*) to the highest of 5.6/second (*cello*). That is, over 70% of time the disk stays in an idle state. Such a low average request rate in the office/engineering environment is very common as indicated in [3]. For file system /swap2 which has an unusually high traffic than others, the maximum disk write workload is about 13.3 MB per hour. With such high traffic, we have, on average, a write access rate of only 3.69 or 0.5 times per second if the average write request size is 1K bytes or 8K bytes, respectively. Taking into account the bursty requests phenomenon, we expect a very long idle period between two request bursts.

Another type of important workloads is transaction processing which can be found in many database applications such as airline ticket reservation systems and banking systems. The characteristics of this workload are quite different from the office/engineering environment. The average access rate is medium to high and the distribution of disk accesses is fairly uniform over time unlike office/engineering environment. Throughput is the major concern in this environment. The performance of such systems, however, is largely determined by the small write performance [5].

The I/O access pattern in scientific computing or supercomputing environment is dominated by sequential readings or writings of large files [2, 16, 3]. The I/O performance of this kind of workload is mainly determined by the raw performance of the I/O hardware such as the disk speed and the I/O channel bandwidth.

Clearly, different workloads have different disk access patterns. There has been no one optimal I/O system for all different workloads. For example, the Log File System performs much better than the Fast File System for small file writes in the office/engineering environment. However, both the Fast File System and the Extended File System are still good choices for transaction processings. One may suggest to choose different file systems for different work loads, but for a system with mixed workloads, keeping multiple file systems in one I/O system is prohibitively expensive. *DCD* is the winner here because it is implemented at the device driver or device level so that only one file system is needed to satisfy diverse workloads. A *DCD* can also change its configuration on-the-fly to adapt to a changing workload. One command can make the DCD redirect all following requests to bypass the cache-disk and go directly to the data disk, which is equivalent to changing the DCD back to a traditional disk.

Parameter	Value
Formatted Capacity:	335 MB
Track Buffer:	none
Cylinders:	1449
Data Head:	8
Data Sector per Track:	113
Sector Size:	256 B
Rotation Speed:	4002 rpm
Controller Overhead (read)	1.1 ms
Controller Overhead (write)	5.1 ms
Average 8 KB access:	33.6 ms

Table 1: HP C2200A Disk Drive Parameters

Due to the time limit, we concentrate on the typical workload environment in our performance evaluation, the office and engineering workload environment. We use the real-world workload to carry out the simulation. The trace files are obtained from Hewlett-Packard. The trace files contain all disk I/O requests made by 3 different HP-UX systems during a four-month period [15]. The three systems represent 3 typical configurations of the office/engineering environment. Among them, *cello* is a timesharing system used by a small group of researchers (about 20) at HP laboratories to do simulation, compilation, editing and mail. The *snake* is a file server of nine client workstations with 200 users at the University of California, Berkeley. And *hplajw* is a personal workstation.

For each system, we randomly selected 3 days of the trace data and concatenate the 3 files together into one. We selected the following three days: 92-04-18, 92-05-19 and 92-06-17 for *hplajw*, 92-04-18, 92-04-24 and 92-04-27 for *cello*, and 92-04-25, 92-05-06, and 92-05-19 for *snake*. Because each system contains several disk drives, we used the most-active disk trace from each system and use it as our simulation data. The exception is *cello*, because the most active disk in it contains a news feed that is updated continuously throughout the day resulting in high amount of traffic similar to the transaction processing workload. We excluded the disk containing the news partition from our simulation.

3.2 Trace-Driven Simulator

A trace-driven simulation program was developed for our performance evaluation purpose. The program was written in C++ and run on Sun Sparc workstations. The core of the simulation program is a disk simulation engine based on the model presented in [17]. The disk parameters that we use in this simulation are chosen based on the HP C2200A [17], as shown in Table 1.

The detailed disk features such as seek time interpolation,

head positioning, head switching time and rotation position are included in the simulation model. It is also assumed that the data transfer rate between the host and the DCD disk controller is 10 MB/s. For the DCD consisting of 2 physical disks, the program simulates two physical disk drives at the same time, one for the cache-disk and the other for the data disk. The same disk parameters are used for both the cache-disk and the data disk except for the capacity difference. For the *DCD* consisting of only one physical disk, two logic disk drives are simulated by using two disk partitions on a single physical drive. Each of the partitions corresponds to a partition of one physical HP C2200A disk.

One difficult task in designing the *DCD* disk system is to keep the mapping information of the Physical Block Address (PBA) in the data disk and the Log Block Address (LBA) in the cache-disk so that the information retrieval is efficient. In our simulation program, several data structures were created for this including a PBA hash chain, an LBA table and a buffer list for the LRU/Free buffer management. Some structures are borrowed from the buffer management part of UNIX [18].

4 Numerical Results

In this section, we evaluate the performance of the *DCD* system described in the previous section by means of trace-driven simulation. The most important I/O performance parameter for the office/engineering environment is the response time. Users in this computing environment concern more about the response time than about the I/O throughput. A system here must provide a fairly short response time to its users. Two response times are used in our performance evaluation. One is the *response time* faced by each individual I/O request and the other is the *average response time* which is the sum of all individual response times divided by the total number of access requests in a trace.

We have run the simulation program under various configurations using the trace data described above. The RAM buffer size is assumed to be 512 KB and cache-disk is assumed to be 20 MB in our simulation. We have simulated both the physical cache-disk *DCD* and the logical cache-disk *DCD* systems. For the logical cache-disk *DCD* system, we assign the first 80,000 sectors (20 MB) in a disk drive as the logical cache-disk and the remaining partition as the logical data disk to run the simulation. All results are obtained with destage process running unless otherwise specified.

4.1 Write Performance with Immediate Report

For the purpose of comparison, we simulated the write performance of both a traditional single-disk system and the *DCD* system. Response times of all individual I/O requests in a trace file are plotted in Figure 5 through Figure 7 for the traditional disk system (on the left hand side of each figure) and the *DCD* system. Each black dot in a figure represents the response time of a write request in the specified trace. Figures 5 and 6 show response times of the *hplajw* traces and *snake* traces, respectively. It can be seen from Figures 5 and 6 that large response times that are present in the plots for the traditional disk completely disappear from the response time plots of the *DCD* system. The response times for the *DCD* system are too small to show in these figures with such large scales. Note that the y axis in these figures goes as high as 3000ms. We plotted the response times of the *DCD* for these two traces again with smaller y scale as shown in Figure 8. It can be seen from this figure that most requests have very small response times, mainly data transfer time, except for one peak in Figure 8(b) that approaches 110 ms. For *hplajw* traces (Figure 8(a)), write response times are all between 0.1 and 0.8 ms because the size of most requests is 8 KB which is also the maximum size in the trace. It is interesting to note in this figure that there is virtually no queuing at all. Similar performance improvements are observed for the *cello* trace as shown in Figure 7.

Traces	traditional disk		logical *DCD*	
	avg	max	avg	max)
hplajw	134	2848	0.65	0.8
cello	205.3	4686	5.9	808.65
snake	127.6	7899	0.75	109.4

Table 2: Write response time comparison between the traditional disk and the *DCD* with a logical cache-disk with immediate report (ms).

As shown in Figures 5, 6, and 7, The individual write response times of the *DCD* are significantly lower than that of the traditional disks The few peaks in the curves for the *DCD* system correspond to the situations where the RAM buffer becomes full before the cache-disk finishes the writing of a current log so that incoming requests must wait in the buffer queue. As expected, most write requests in the *DCD* system do not need to wait in the queue. Response times are approximately identical to data transfer times or the data copy times by the CPU if the *DCD* is implemented at the device driver level.

Table 2 lists average and maximum response times for the three traces. As shown in the table, the average write response time of the traditional disk system is as high as 205 ms. The maximum write response time shown here is 7899 ms implying a very long waiting queue at the controller. However, the average write response times of the *DCD* system for *hplajw* and *snake* are less than 1 ms which is more than two orders of magnitude improvement over the traditional disk. The relatively long response time for *cello*, 5.9 ms, representing about one order of magnitude improvement over the traditional disk, is mainly due to several large peaks in Figure 7 because of the limited RAM buffer size. Other than a few peaks, the performance improvements are similar to those of *hplajw* and *snake*.

4.2 DCD with Report After Complete

This scheme has good reliability because a write is guaranteed to be stored in a disk before the CPU is acknowledged. If the RAM buffer were a volatile memory, this scheme would be much more reliable than the immediate report scheme. But the performance of the former is lower than the latter because a request is acknowledged as complete when all requests in its group are written into a disk. Nevertheless, the *DCD* still shows superb performance as shown in the right most curves of Figures 5, 6, and 7 that show the performance of the *DCD* system with *report after complete* scheme. In these figures, a separate physical disk is used as the cache-disk. The number of peaks and the height of each peak of the *DCD* system are significantly lower than that of the traditional disk system as shown in the figures.

Table 3 shows the average and the maximum write response times for the two architectures. We observed that the *DCD* system show about 2 to 4 times better performance than that of the traditional disk. Note that in our simulation the old HP

Traces	traditional		physical DCD		logical DCD	
	avg	max	avg	max	avg	max
hplajw	134	2848	40.3	211	53.1	266.4
cello	205	4686	56.5	849	74	5665.9
snake	127.6	7898	29.4	491	59.4	4613.4

Table 3: Write response time comparison between traditional disk and the DCD with report after complete (ms).

Traces	Physcal DCD		Logical DCD	
	On	Off	On	Off
hplajw	28.5	28.9	39.6	28.5
cello	77	43	112	44
snake	32	32.2	66.7	57.5

Table 4: Effects of destaging algorithm in terms of average write response time (ms) "On" means that the destage is turned on while "Off" means that the destage is turned off.

2200A disk model is used that has slow spindle speed and low recording density. We expect that the speedup factor will increase when the disk spindle speed and linear density improves. Our prediction comes from the fact that the performance improvement of our DCD mainly results from the reduction of the seek time and rotation latency, while the disk write time stays the same as traditional disk. Therefore, the DCD will show even better performance if the proportion of seek time in each disk access increases. It is fairly clear that the average seek time is not likely to reduce very much in the near future, but the rotation speed is expected to increase to 7200 rpm. Some disk drives have already used the speed of 7200 rpm. The linear density are expected to double in the next few years. As a result, write time will be reduced to one-third of its present value. Therefore, we expect the speedup factor of the DCD to increase in the near future.

4.3 Destage Cost

The performance results presented in the previous subsections were obtained with the destage algorithm running. In order to study the performance effects of the destage process, we deliberately disable the destage process and run the simulation again for shorter traces until the cache-disk is full. The results were shown in Table 4. We only measured the response time for *report after complete* scheme because the performance of the DCD with the *immediate report* is not very sensitive to the destage process. Destaging has almost no effects on the performance of the physical DCD for *hplajw* and *snake* indicating that the LWFD destage algorithm performs very well. But it does affect the performance of the logical DCD system because the data disk is utilized more. Performance degradation caused by destaging ranges from 16% for *snake* to 39% for *hplajw*. It also has dramatic effect on *cello* (up to 254%) because of the high request rate and relatively uniform access pattern in *cello*. It is not easy to find a long idle time on the data disk to perform destaging by our simple idle detector. We expect that more performance gains in the DCD system can be obtained by using a good idle detector, and by applying a disk-arm-scheduling algorithm [19, 20], and other optimal schedule algorithms such as linear threshold scheduling [21], etc. to the destage process.

Traces	traditional		physical DCD		logical DCD	
	avg	max	avg	max	avg	max
hplajw	53.5	2873	21.1	156.5	22.1	156.5
cello	159.3	3890	149.6	3890	150.4	3890
snake	189	7276	103	769	106	810

Table 5: Read performance in response times (ms)

4.4 Logical Disk Cache vs Physical Disk Cache

The DCD system can be implemented either using two physical disk drives, or using two logical drives. The DCD with two physical drives has good performance but the cost is relatively high because it requires an additional disk drive though with small capacity. While the DCD configured using two logical disk drives may not perform as well as the DCD with two physical disks, its cost is just a small disk space (5 - 50 MB) which is a very small fraction of the total capacity of today's disk drives that are usually more than 1 GB each. In order to compare the performance difference between the physical cache-disk and the logical cache-disk, we have listed results for both cases in Tables 3 to 5. As expected, the performance of the DCD with the logical cache-disk performs very well. For immediate report, the average write response times are two orders of magnitude faster than those of traditional disk as shown in Table 2. The performance of the DCD with Report After Complete is lower than the DCD with two physical drives as shown in Table 3. However, the performance of the logical cache-disk DCD is several times better than that of a traditional disk as shown in the tables. We expect that the speed up factor will get larger with the increase of the disk spindle speed and the linear density.

4.5 Read Performance

The read performance of the DCD and the traditional disk is compared in Table 5. The results are better than expected. For *hplajw*, the average read performance of the DCD is about 2 times better than the traditional disk while the maximum response time of the DCD is 10 times smaller than that of traditional disk. For *snake*, the DCD shows about 50% better average response time and about 9 times better maximum response time than the traditional disk. It is important to note that the above improvements are true for both 2 physical disk DCD and 2 logical disk DCD systems. The performance improvements for read requests can mainly be attributed to the reduction of write traffic at the data disk. The data disk has more available disk time for processing read requests. For *cello*, the DCD shows a similar read performance to the traditional disk due to high read traffic and the limitation of buffer size as indicated before.

4.6 Cost Considerations

While 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. Based on the current technology for memories [22], the cost of 1 MB storage is about $0.25 for disks and $120 for nonvolatile RAMs. Additional 20 to 50 MB of disk space and 512 KB of NVRAM will bring up the disk system cost by a small fraction. 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, this cost is a small fraction of the total cost of a file server. If the logical cache-disk is used in the *DCD*, the additional cost boils down to the cost of the RAM buffer which can be either DRAM or NVRAM depending on the reliability requirement. With a 512 KB NVRAM being used as the buffer, we have calculated the response reduction per additional dollar spent. For instance, the write performance for traces *hplajw* and *snake* triples for each additional dollar spent. Therefore, we argue that the *DCD* is a very cost-effective approach.

5 Related Work

Baker et al. presented a study on using a NVRAM as a disk cache in distributed client/server systems [1]. They found that one-megabyte of NVRAM at each client can reduce the write traffic to the server by 40-50%, and one-half megabyte NVRAM write buffer for each file system on the server side reduces disk accesses by 20% to 90%. Ruemmler and Wilkes reported in [15] their simulation results of applying an NVRAM as a write cache to a disk system. They found that placing an NVRAM with the size of 128 to 4096 KB as a write cache can reduce the I/O response time by a factor of 2 to 3, since overwrites account for a major portion of all writes (25% for *hplajw*, 47% for *snake* and about 35% for *cello*)

Another advantage of using a large RAM to buffer disk write requests is that the requests can be reordered in the buffer. Such reordering makes it possible to schedule disk writes according to seek distance or access time so that the average head positioning time can be reduced substantially. Extensive studies have been conducted and many good algorithms such as SCAN, Shortest Access Time First (SATF) and Weighted Shortest Time First have been proposed [19, 20]. In the DCD system, the data are first written into the cache-disk in a log format, which eliminates most seeks and rotation latencies. The disk arm scheduling is not needed. However it can be applied to the destage algorithm to reduce the cost of the destaging. This is especially important for relatively high and uniform time-sharing workloads such as *cello*, and transaction processing workloads.

Several techniques have been reported in the literature in minimizing small write costs in RAID systems. Parity logging, an elegant mechanism proposed by Stodolsky et al. [5], utilizes the high transfer rate of large sequential data to minimize the small write penalty in RAID systems. They have shown that with minimum overhead, the parity logging eliminates performance penalty caused by the RAID architectures for small writes. Solworth and Orji [23] proposed a very interesting approach called write-twice to reduce the small write penalty of mirror disks. In their method several tracks in every disk cylinder are reserved. When a write request comes, it is immediately written to a closest empty location, and the controller acknowledges the write as complete. Later the data are written again to its fixed location. Up to 80% improvement in small-write performance was reported. It can also be used to reduce write response time in normal disks. The write-twice method is normally implemented in the disk controller level since it needs detailed timing information of disk drive. It also requires substantial amount of the disk storage to reserve tracks in each cylinder. Except for a few high-end products, most disk drives now use 2 or 3 platters per drive, implying only 4 to 6 tracks per cylinder. Therefore, the write-twice approach is mainly for those applications that the cost is not a primary concern.

The Episode file system which is a part of the DECorum Distributed Computing Environment, uses log to improve crash recovery of meta-data [10, 24]. The changes of meta-data in the write buffer are collected into logs and are periodically (typically every 30 seconds) written into disk to ensure a reliable copy of the changes. Cache logging eliminates many small writes caused by meta-data updates. The cache logging works in the file system level while the *DCD* works at the device level. The cache logging works horizontally where the content of the log disk is basically a mirror image of the large RAM buffer, whereas the log disk and RAM buffer in the *DCD* work vertically in the sense that the log disk acts as an extension of a small NVRAM buffer to achieve high performance with limited cost.

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 the log disk system and the traditional disk system. 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 cache-disk can be implemented either using a separate physical drive or a logical disk that is a partition of the data disk depending on performance/cost considerations. The disk cache including the RAM and the cache-disk is transparent to the CPU so that there is no need to change the operating system to incorporate this new disk architecture. Simulation experiments have been carried out by using traces representing 3 typical office/engineering workload environments. Numerical results have shown that the new *DCD* architecture is very promising in improving write performance. With immediate report, the *DCD* improves write performance by one to two orders of magnitude over the traditional disk systems. A factor of 2 to 4 performance improvements over traditional disks are observed for the *DCD* with the report-after-complete scheme. 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.

As a future work, we are currently investigating the possibility of applying the *DCD* architecture to the RAID architecture. The objective is to optimize both throughput and response time of future RAID systems. We will also investigate the behavior of the *DCD* under different workload environments using synthetic traces.

Acknowledgements

This research is supported in part by NSF under grants MIP-9208041 and MIP-9505601. The authors would like to thank Hewlett-Packard for providing trace files to us.

References

[1] 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 Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, (Boston, MA), pp. 10–22, Oct. 1992.

[2] R. H. Katz, G. A. Gibson, and D. A. Patterson, "Disk system architectures for high performance computing," *Proceedings of the IEEE*, pp. 1842–1858, Dec. 1989.

[3] M. Rosenblum and J. Ousterhout, "The design and implementation of a log-structured file system," *ACM Transcations 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, Electrial Engineering and Computer Sciences, University of California at Berkeley, Oct. 1988.

[5] 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.

[6] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID : High-performance, reliable secondary storage," in *submitted to ACM Computing Surveys*, Oct. 1993. Published as Performance Evaluation Review Special Issue, Volume 23, No.1.

[7] ShengbinHu and Q. Yang, "A closed form formula for queueing delays in disk arrays," in *Proceedings of the 1994 International Conference on Parallel Processing*, (St. Charles), pp. II–189–192, Aug. 1994.

[8] 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.

[9] 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.

[10] K. W. Shirriff, *Sawmill: A Logging File System for a High-Performance RAID Disk Array.* PhD thesis, University of California at Berkeley, 1995.

[11] J. Menon, "A performance comparison of RAID-5 and log-structured arrays," in *Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing*, pp. 167–178, Aug. 1995.

[12] R. M. English and A. A. Stepanov, "Loge: A self-organizing disk controller," in *USENIX Conference Proceedings*, (San Francisco, CA), pp. 237–252, USENIX, Winter 1992.

[13] W. de Jonge, M. F. Kaashoek, and W. C. Hsieh, "The logical disk: A new approach to improving file systems," in *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, (Asheville, NC), pp. 15–28, Dec. 1993.

[14] R. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes, "Idleness is not sloth," in *Proceedings of the USENIX 1995 Technical Conference on UNIX and Advanced Computing System*, (Naw Orleans), pp. 201–212, Jan. 1995.

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

[16] D. Kotz and N. Nieuwejaar, "File-system workload on a scientific multiprocessor," *IEEE Parallel & Distributed Technology*, pp. 51–60, Spring 1995.

[17] C. Ruemmler and J. Wilkes, "An introduction to disk drive modeling," *IEEE Computer*, pp. 17–28, Mar. 1994.

[18] S. J. Leffler, M. K. McKusick, M. J. Karels, and J. S. Quarterman, *The Design and implementation of the 4.3BSD UNIX operating system.* Addison-Wesley, 1989.

[19] D. M. Jacobson and J. Wilkes, "Disk scheduling algorithms based on rotational position," Tech. Rep. HPL-CSP-91-7rev1, Hewlett-Packard Laboratories, Mar. 1991.

[20] M. Seltzer, P. Chen, and J. Ousterhout, "Disk scheduling revisited," in *Proceedings of the 1990 Winter USENIX*, (Washington, D.C.), pp. 313–324, Jan. 22-26 1990.

[21] A. Varma and Q. Jacobson, "Destage algorithms for disk arrays with non-volatile caches," in *Proceedings of the 22nd Ann. Int'l Symposium on Comput. Arch.*, pp. 83–95, June 22–24, 1995.

[22] M. Wu and W. Zwaenepoel, "eNVy, a non-volatile, main memory storage system," in *Proceedings of the 6th International Symposium on Architectural Support for Programming Languages and Operating Systems*, pp. 86–97, Oct. 1994.

[23] J. A. Solworth and C. U. Orji, "Distorted mirrors," in *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pp. 10–17, 1991.

[24] M. L. Kazar, O. T. A. B. W. Leverett, V. A. postolides, B. L. Bottos, S. Chutani, C. F. Everhart, W. A. Mason, S. Tu, and E. R. Zayas, "Decorum file system architectural overview," in *Proceedings of the 1990 USENIX Summer Conference*, pp. 151–163, June 1990.

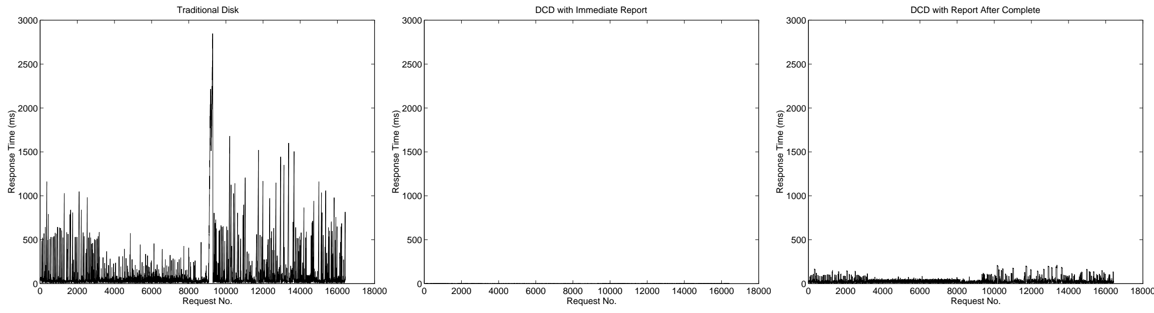

Figure 5: Response times for trace *hplajw*. Each black dot in this figure represents the response time of an individual write request. The left-most figure is for the traditional disk system; the middle one is for the DCD with *Immediate report scheme*, i.e. as soon as the data are transferred to the RAM buffer, the controller sends an acknowledgement of write complete to the host; the right-most figure is for the DCD with *report after complete* scheme, i.e. a write request is acknowledged as complete only after it has been actually written into a disk.

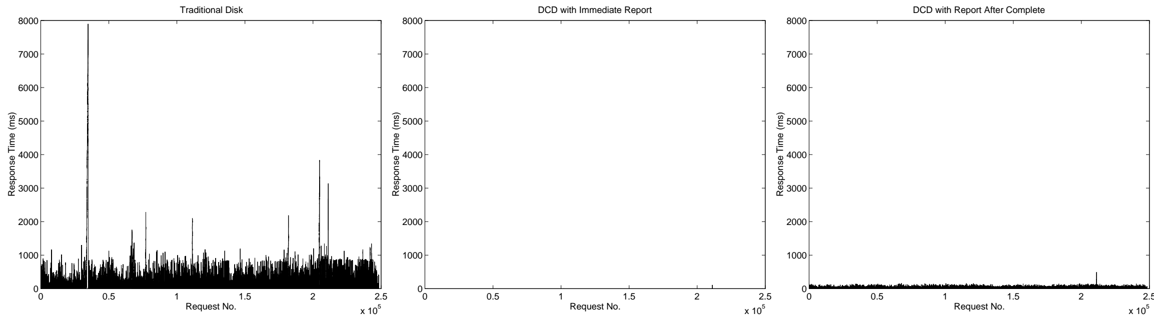

Figure 6: Response times for trace *snake*. Each black dot in this figure shows the response time of an individual write request.

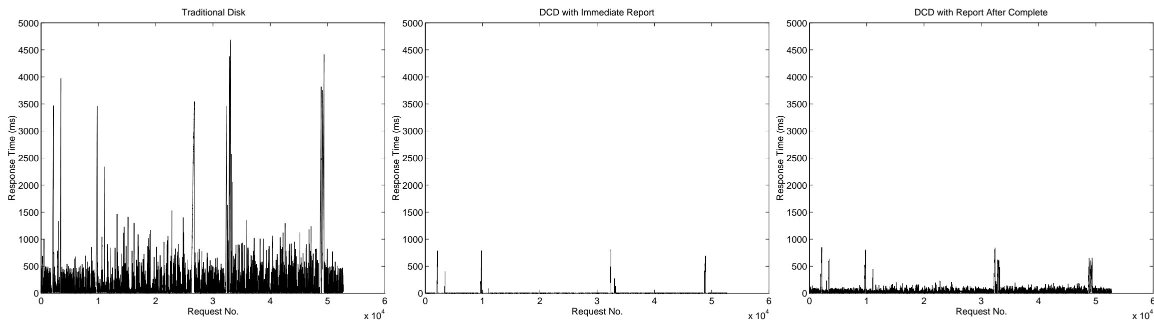

Figure 7: Response times for trace *cello*. Each black dot in this figure shows the response time of an individual write request.

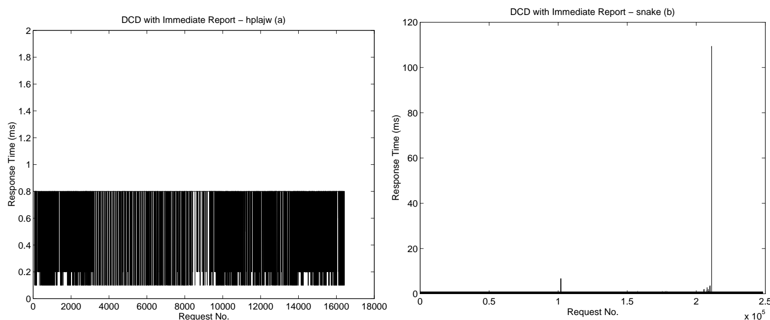

Figure 8: Response times for *hplajw* and *snake* with smaller scale. Each black dot in this figure shows the response time of an individual write requ est. Note that the figure scale has been changed to show the small response times. *Immediate report* is used.