Reducing Hybrid Disk Write Latency with Flash-Backed I/O Requests

Timothy Bisson
and Scott A. Brandt
University of California, Santa Cruz
Santa Cruz, CA

***Abstract*—One of the biggest bottlenecks in desktop-based computing is the hard disk with I/O write latency being a key contributor. I/O write latency stems from the mechanical nature of hard disks, with seek and rotational delays the major components. Hybrid disk drives place a small amount of flash memory (NVCache) on the drive itself which can be leveraged by the host and has the potential to increase I/O performance and reduce hard disk power consumption.**

We present an I/O scheduling algorithm, "Flash-Backed I/O Requests", which leverages the on-board flash to reduce write latency. Since flash memory and rotating media have different I/O characteristics, predominantly in random access context, an I/O scheduler can decide which media will most efficiently service I/O requests. Our results show that with Flash-Backed I/O requests, overall write latency can be reduced by up to 70%.

I. Introduction

Hard disks are slow mechanical storage devices. However, because they are inexpensive and offer large capacities (one terabyte hard disks are available), they are typically used as the back-end media for general purpose operating systems. Although disk capacities are expected to increase by a factor of 16 by 2013, disk bandwidth and seek time are not expected to scale as much [1]. As a result, the gap between drive capacity and performance will continue to grow.

One of the key contributors to hard disk latency is seek time, which is the penalty incurred when the head must move from one track location and settle on another. Seek time is a function of the inter-sector request distance and ranges from a few milliseconds to 10s of milliseconds.

Both operating systems and disk drives try to mitigate the performance penalty rotating media imposes by reducing I/Os and coordinating their execution. Solutions include page caching, I/O scheduling, and file system allocation/defragmentation. Page caching attempts to keep the most recently used data in memory, such as file and directory data. Several I/O scheduling algorithms have been proposed to minimize I/O latency [2], [3]. File system allocation and defragmentation policies try to minimize disk seeks by keeping file data contiguous. Even with such performance enhancements, hard disks still remain the most significant bottleneck in the I/O path.

Upcoming hybrid disks will place a small amount of flash memory (NVCache) next to the rotating media as shown in Figure 1. The first hybrid disks will have several hundred megabytes of NVCache in a 2.5 inform-factor [4], but because

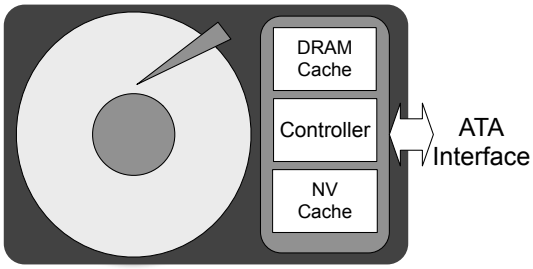


Fig. 1. Hybrid Disk

of the already low profit margins in the disk drive industry, it isn't clear how much the NVCache will grow [5]. The NVCache will have constant access time throughout its block address space because of its non-mechanical nature, unlike rotating media.However, rotating media sequential throughput is significantly faster than flash media. Table I shows the general trade-offs between flash memory and rotating media. Hybrid disks present an opportunity to increase performance over traditional disk drives and reduce power consumption while still minimizing the reliability and spin-up latency impact of spin-down algorithms. While the rotating media is spun-down write requests can be redirected to the NVCache, reducing contact start-stop operations and aggregate spin-up latency [6].

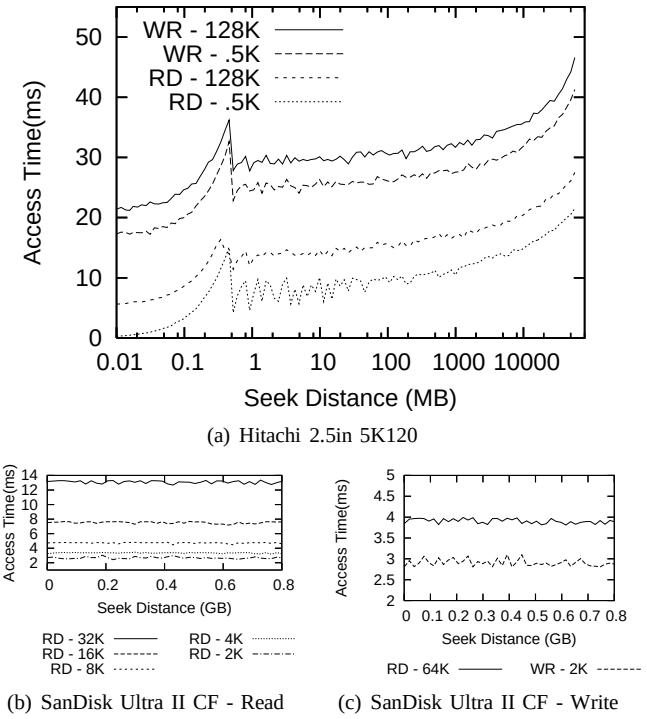
I/O scheduling algorithms are traditionally implemented to minimize access time to rotating media. However, with hybrid drives such a presumption may no longer be most efficient. In this paper we propose leveraging upcoming hybrid drives to reduce I/O write latency through "Flash-Backed I/O Requests". Flash-backed I/O requests redirect write requests to flash media when it is more efficient to service an I/O request with flash media rather than rotating media. The redirected request is kept in main memory until it can be written to rotating media without a significant write penalty. Redirected I/O requests and their dirty pages still remain in main memory; data safety is preserved in the case of power-failure because the dirty pages are backed in the NVCache. Additionally, since the flash and rotating media exist in the same enclosure data separation is not possible.

II. Background and Motivation

I/O write latency is responsible for a large portion of total disk access time. File system allocation policies try to minimize write latency by providing spatial locality of

	Flash	Rotating Media
Capacity	Smaller	Larger
Random Access	Faster	Slower
Sequential Access	Slower	Faster

TABLE I
GENERALIZED FLASH AND ROTATING MEDIA PERFORMANCE AND CAPACITY COMPARISON



(a) Hitachi 2.5in 5K120

(b) SanDisk Ultra II CF - Read

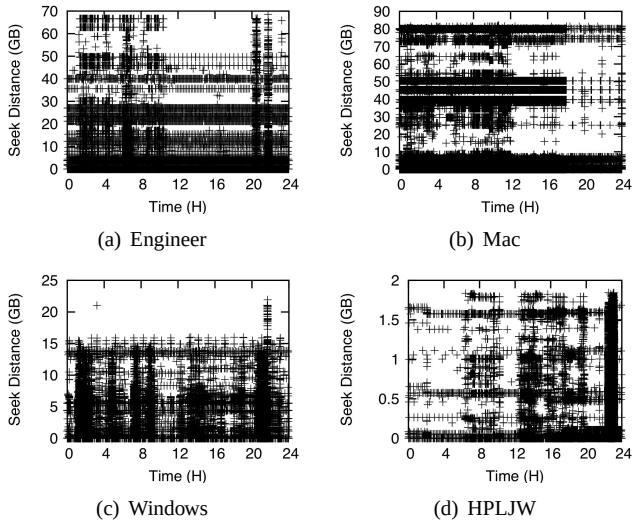
(c) SanDisk Ultra II CF - Write

Fig. 2. Access Time with 99% Confidence Interval.

file data through contiguous file block allocation. However, some file systems do not provide a high degree of inter-file locality. For example, FFS-derived file systems, such as ext2/3, try to allocate files evenly across the entire address space in block groups [7]. Additionally, user-initiated file system operations prevent sequential write operations from occurring. For example, a read between two writes. Log-based file systems which employ various versions of copy-on-write can address this issue [8], [9]. However, such file systems must employ cleaner processes to make room for new writes.

Hybrid drives bring forth a new storage technology and with it an opportunity to improve hard disk write performance without the need for a log-structured file system allocation policies and the associated cleaner overhead. Hybrids drives provide such a capability because two storage media are co-located with each other in the same enclosure and I/O can be directed to either storage media. Having both storage media in the same enclosure is also important for data safety—data from either media cannot be separated. This may not be true with other upcoming technologies that also leverages flash such as Intel Robson, in which flash is located on the motherboard instead of the drive [10]. File system integrity can be lost if the drive is detached from the motherboard..

I/O scheduler implementations provide different polices. For



(a) Engineer

(b) Mac

(c) Windows

(d) HPLJW

Fig. 3. Seek Distance Frequency

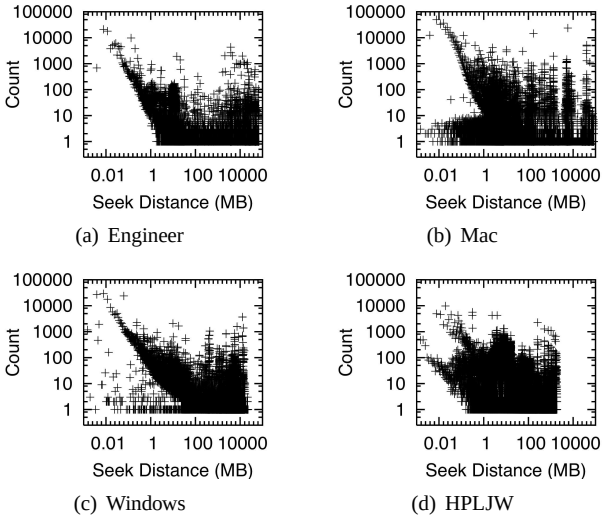
example, some schedulers push for fairness while others aim solely for efficiency. Traditionally, I/O schedulers used the access time along the single dimension of a block device's logical address space to guide scheduling decisions. With hybrid drives, two access times are available: one along the rotating media address space, and one along the flash media address space. As a result, an I/O scheduler can improve the I/O performance on hybrid drives by leveraging the additional block address space. Figure 2 shows the access times for a flash media and rotating media device. This figure shows the potential benefit of utilizing a hybrid drive to reduce write latency. By augmenting an I/O scheduler to be aware of the access-time discrepancy between flash and rotating media, it can select which media is most efficient to service any

A. Trace-Based Motivation

We performed an analysis of access patterns for four block level traces (shown in Table II). The traces were gathered from desktop systems and represent 7 days of activity. They span four operating systems: Windows XP, Mac OS X, Linux, and HPUX. Each system was used in a desktop environment and described in more detail in Section IV-A.

Figure 3 shows seek distance for write operations in gigabytes as a function of time for each trace. Seek distance is the difference in logical block address between two successive requests. Seeks for read requests are not shown in this figure, and all seek distances are absolute values. The point of these figures is to show the horizontal banding that occurs for several seek distances for each trace. The horizontal banding represents potential I/O redirection to flash media. The banding that occurs for large seek distances represents write requests which incur long seeks that can potentially be satisfied by the NVCache. These figures only show the first 24 hours of each trace so that the horizontal banding can be clearly seen.

To further motivate the need for reducing write latency we show the total I/O count as a function of the seek distance for the entire 7 days of each trace in Figure 4. In these figures



(a) Engineer (b) Mac

(c) Windows (d) HPLJW

Fig. 4. Seek Distance Count

only write operations contribute to the total count, and the seek distance is again an absolute value. This figure shows that most frequent write operations occur for short seek distances and the frequency continues to decrease until roughly a 1MB seek distance. After reaching 1MB, the I/O frequency at a particular seek distance remains relatively constant. We also consider write operations after 1MB ideal candidates flash-backed I/O requests.

B. Hybrid Drives

The NVCache on hybrid disks can be controlled through new ATA commands specified in the ATA8 specification [11]. The NVCache does not extend a disk's capacity, rather it can store particular sectors. Sectors stored in the NVCache are either pinned or unpinned, which when referred to as a collection are known as the pinned and unpinned set, respectively. The host manages the pinned set, while the disk manages the unpinned set. The disk uses the unpinned set in a new power mode, NV Cache Power Mode, in which the drive will redirect all I/O to the NVCache if possible and use an aggressive disk spin-down policy.

The host controls I/O to the NVCache pinned set. A host can issue a command to pin one or more sectors in the NVCache; any subsequent I/O to those sectors will be redirected to the NVCache. A host specifies multiple sectors with an extent-like interface, called LBA Range Entries. When pinning LBAs into the NVCache the host specifies the source for that LBA range entry: host or rotating media, by setting a Populate Immediate bit. If set, those sectors are read from rotating media into the NVCache, otherwise, a write operation from the host for those sectors will go into the NVCache. Pinned sectors can also be queried or removed from the NVCache. Finally, unpinned sectors can be flushed from the NVCache to make room for the pinning of additional sectors.

III. Flash-Backed I/O Requests

Using hybrid disk drive technology, the goal of flash-backed I/O requests is to reduce write latency. In order to do this, we augment an I/O scheduler by adding an additional I/O queue in which certain write requests (along with the data) persist in main memory, but are backed in the NVCache. Keeping such requests in main-memory provides additional opportunities for normal I/O requests to be coalesced, and read requests a higher chance of being satisfied from main-memory. Although the I/O bottleneck is reduced, there is the trade-off in additional memory consumption. It is important to note that the redirected I/O requests are backed in non-volatile flash memory, so data integrity is preserved in the event of a power failure.

A. I/O Redirection

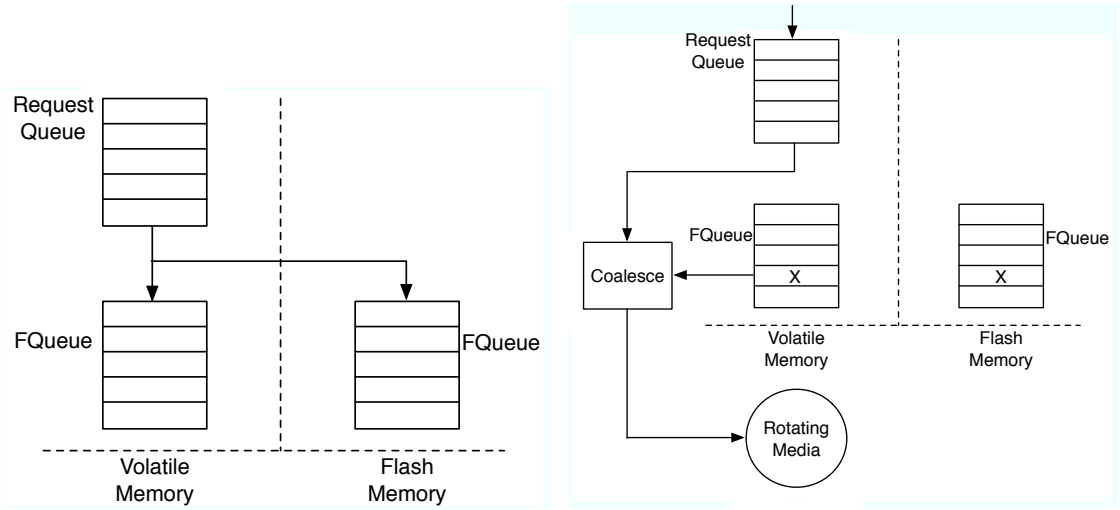
This section describes how I/O requests are redirected and coalesced with Flash Backed I/O Requests. Figure 5(a) shows how I/O requests are redirected. When a write request is redirected, it is removed from the Request Queue, which exists in main memory. The Request Queue refers to the traditional I/O scheduling queue. After being removed from the Request Queue, the request is added to two new request queues: the main memory FQueue and a non-volatile FQueue (NVCache). Adding the request to the main memory FQueue moves the request reference between the two queues. Adding the request to the non-volatile FQueue involves issuing an asynchronous write request to it. However, before the write request is issued, the host must pin the corresponding sectors and not set the Populate Immediate bit. The pinning and unpinning operations is very fast (microseconds) because the disk need only set bits in its volatile memory to accomplish this.

Although the goal is to reduce write latency, we still must push FQueue requests back to rotating media. As a result, the volatile FQueue is fundamentally a cache for rotating media, backed by non-volatile memory. Therefore, the content of the two FQueues at any given time are identical—the sole purpose of the flash FQueue is to prevent data loss from the volatile FQueue in the case of power failure. Since the volatile FQueue is akin to a cache, when requests are kept in memory longer more opportunities exist to coalesce the volatile FQueue contents with write and read requests from the Request Queue.

Figure 5(b) shows how I/O requests are coalesced. Coalescing requests involves four steps: (1) unpin the non-volatile FQueue data, (2) coalesce the request from the DRAM FQueue with the current write request, (3) submitting the I/O write request, and (4) removing request and data from the non-volatile FQueue.

The first step is to use the ATA NVCache unpin command to remove a particular set of LBAs from the NVCache. Step two coalesces the in-memory requests data and step three submits the requests with the modified request data and request itself to reflect the new content. Since none of the LBAs in the request are pinned, the request is directed entirely to rotating media and not the NVCache. Finally the last step is to remove the request and its data from the non-volatile FQueue.

It must be noted that the DRAM and flash size of the FQueue for hybrid I/O scheduling must be the same. It doesn't make sense to have more or less DRAM FQueue than flash



(a) Redirection (b) Coalesce

Fig. 5. HIO Scheduling

FQueue. If there is more DRAM FQueue memory, then safety is lost because any requests in the DRAM FQueue may not be backed by the NVCache. If there is more non-volatile FQueue, the performance benefit of I/O redirection is mitigated because fewer requests can be stored in the DRAM FQueue.

B. When to Redirect

The fundamental goal of flash-backed I/O request is to reduce the overall I/O write bottleneck. To that end, only if an I/O request will reduce write latency should it be placed on the volatile and non-volatile FQueues. In order to make such a decision, the redirection algorithm relies on the inter-block random access times of flash and rotating media. Since random access time is a function of the inter-block distance for rotating media, it is necessary to know the location of the last I/O (head location). However for flash, such information is unnecessary as it has constant random access time, as previously shown in Figure 2.

We have developed an algorithm to determine which requests are redirected to the FQueue. The decision takes as input the disk drive's head location, current request information, and information about the next request information. The algorithm is shown in Figure 6. The first thing to notice is that the algorithm first tries to coalesce the current request with a request from the FQueue. If this is possible the algorithm will not attempt to redirect I/O because the corresponding FQueue request is able to piggy-back on the current request to rotating media. Note that this algorithm is only entered on a write request. If the request isn't coalesced (*coalesced* = false) we extract I/O access times for four possible requests for the NVCache and rotating media. Given these access times, if the inequality:

$(h\_n + n\_nn) \geq (fn + h\_nn)$

is true, the current request is redirected to the FQueues. In this inequality, *h_n* is the access time from the current LBA location of the disk head to the current request's LBA location. Correspondingly, *n_nn* is the access time from the current request's LBA location to the next request's LBA location. *fn* is the flash access time for current request, which dependent only on the request size, while *h_n* is the access time from the current LBA location of the disk head to the next request's LBA location.

```
/* Decision to redirect request */
REDIRECT_REQUEST(head, request):
  /* Attempt to coalesce request with data from dram fqueue */
  {coalesced, new_req} := COALESCE_WITH_FQUEUE(request)

  if (coalesced == true) then
    /* Remove coalesced data from flash */
    UNPIN_REQUEST_FROM_FLASH(request, new_req)

    /* Submit coalesced request */
    SUBMIT_COALESCED_REQUEST(new_req)

    /* Clear coalesced data from dram fqueue */
    CLEAR_DRAM_FQUEUE_REQUEST(request, new_req)
    return
  endif

  /*Get I/O access times*/
  h_n := ACCESS_TIME(head, request->lbn)
  n_nn := ACCESS_TIME(request->lbn, request->next->lbn)
  fn := FLASH_ACCESS_TIME(request->size)
  h_nn := ACCESS_TIME(head, request->next->lbn)

  /*Should redirection occur*/
  if ((h_n + n_nn) > (fn + h_nn)) then
    /*pin request in flash*/
    PIN_REQUEST_IN_FLASH(request)

    /*add request to dram fqueue*/
    ADD_REQUEST_TO_DRAM_FQUEUE(request)
  endif

  /*submit redirected request*/
  SUBMIT_REQUEST(request)
```

Fig. 6. Algorithm for REDIRECT_REQUEST

The goal of the redirection inequality is to redirect I/O requests to the FQueues that will result in less disk head seeking. In effect, this algorithm is looking into the future

to see whether redirecting the current request to the FQueues is more efficient than sending it to rotating media because it includes the access time of the next request into the redirection decision.

For example, consider a request with a starting sector far away from the head's current location but in between several requests spatially near each other. If the decision to redirect is based solely on the head's current position and the next request, no consideration is given about the impact of the decision on following requests. If the request is redirected to flash, the head will remain at the same location. As a result, the following requests (spatially near each other) will also be redirected to flash because of the disk head's relative location to the spatially nearby requests, and cause the FQueue space to quickly become polluted. However, since the following requests are spatially near each other, it would have been more efficient to service them with rotating media. In order to do that, the scheduling algorithm needs to consider the following request's location when making a decision about whether or not to redirect a given request. This is in effect what the algorithm in Figure 6 does.

C. Idle-Time Processing

A benefit of flash-backed I/O requests is that the FQueue requests act like a cache for disk requests, but are backed by non-volatile storage. We propose that an operating system leverage the FQueue to perform delayed I/O. By waiting until the Request Queue is empty, and then flushing FQueue requests back to rotating media, requests can be written to rotating media asynchronously. From the user's perspective, any request that exists in the FQueue is considered completed because it backed in non-volatile media, so no additional time is incurred (from the client's perspective) to move the request data from the FQueue to rotating media. Therefore flushing requests from the FQueue to rotating media while the Request Queue is empty is considered Idle-Time Processing, and another example application of Golding's hypothesis that idleness is not sloth [12].

The order in which idle-time processing purges the FQueue occurs is the same order as flushing—FIFO order with request merging. Additionally, it should be noted that more opportunities to perform idle-time processing exist with larger FQueue sizes, further reducing user perceived I/O latency. Perceived I/O latency is the time to push an incoming user request to a non-volatile media, be it flash or rotating media.

D. Flushing Requests

The volatile and non-volatile FQueues have a finite size. If the rate at which requests are redirected to the FQueues is higher than the coalescing rate, the FQueues will become full. As a result FQueue requests need to be flushed. Fortunately, because there is a copy of the FQueue request (and request data) in volatile memory, only an unpin and write request operation are needed—reading the redirected data from the flash is unnecessary.

Flushing requests from the FQueue is comprised of three steps: (1) when to flush, (2) how much to flush, and (3) the order of request flushing. In order to minimize flushing overhead, we use two watermarks (high and low) to guide when to begin flushing and how much to flush. The high watermark is used to initiate flushing of FQueue requests back to rotating media. Requests are flushed from the FQueue until a low watermark is reached. While flushing occurs, I/O alternates between the FQueue and Request Queue with a 1:1 ratio. All I/O operations originating from the Request Queue only execute the COALESCE_WITH_FQUEUE phase from Figure 6 and are directed to rotating media. The goal of the low watermark is that some requests always exist in the FQueue and as a result there is always potential for I/O redirection. The high-watermark prevents FQueue flushing from the blocking Request Queue processing.

The order of request flushing from the FQueue is done in FIFO order. FIFO is used because coalescing removal is random access removal. This means the oldest FQueue requests haven't been coalesced yet and may be unlikely to be coalesced with a Request Queue request. Therefore, such requests should be removed from the FQueue to make room for other FQueue requests. To minimize the total number of FQueue requests issued back to rotating media, FQueue flushing first attempts to merge any existing requests in the FQueue with the next request scheduled to be written to rotating media.

IV. Evaluation

In order to evaluate the potential benefit of Hybrid I/O scheduling, we implemented a hybrid disk simulator using the I/O access times from a Hitachi EK1000 2.5 in drive and a Sandisk Ultra II Compact Flash media card shown in Figure 2. Using the access times from the two devices we replayed several block-level I/O traces through an augmented I/O scheduler. The following sections describe the workload and results of flash-backed I/O requests.

A. Traces

To evaluate the proposed enhancements, we use block-level access traces gathered from four different desktop machines, which are shown in Table II. Each workload is a trace of disk requests from a single disk, and each entry is contains: I/O time, sector, sector length, and read or write. The first workload, *Eng*, is a trace from the root disk of a Linux desktop used for software engineering tasks; the ReiserFS file system resides on the root disk. The trace was extracted by instrumenting the disk driver to record all accesses for the root disk to a memory buffer, and transfer it to userland (via a system call) when it became full. A corresponding userland application appended the memory buffer to a file on a separate disk. The trace, *HP*, is from a single-user HP-UX workstation [13]. The *WinPC* trace is from an Windows XP desktop used mostly for web browsing, electronic mail, and Microsoft Office applications. The trace was extracted through the use of a filter driver. The final trace, *Mac* is from

Name	Type	Duration	Year
Eng	Linux Engineering Workstation	7 days	2005
HP	HP-UX Engineering Workstation	7 days	1992
WinPC	Windows XP Desktop	7 days	2006
Mac	Mac OS X 10.4 Powerbook	7 days	2006

TABLE II
BLOCK-LEVEL TRACE WORKLOADS

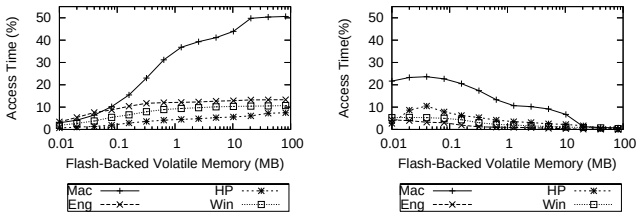
a Macintosh PowerBook running OS X 10.4. The trace was recorded using the Macintosh command line tool, fs_usage, by filtering out file system operations and redirecting disk I/O operations for the root disk to a USB thumb drive.

Since main memory is a resource that will be directly consumed by flash-backed I/O requests, it is important to know how much memory each system had at the time the trace was taken. The Mac and Eng systems both had 1GB of DRAM, while the WinPc had 512MB of memory. We do not know how much memory HP had at the time the traces were taken. Therefore, for those systems in which we are aware of the memory capacity, 100 megabytes for flash-backed I/O request is a 10–20% overhead.

In this section we present results for flash-backed I/O requests. We use several I/O schedulers for the Request Queue: FCFS, SSTF, and CLOOK. Figure 7 shows the normalized write access time for each trace using the three I/O schedulers. Each scheduler is normalized to itself without flash-backed I/O requests. This figure shows two things: (1) there is a significant decrease in write latency when using flash-backed I/O requests, and (2) the relative performance of individual I/O schedulers does not differ significantly. Although the relative performance of each I/O scheduler does not change, the raw numbers do. Since the Request Queue scheduling algorithm does not significantly impact performance, for the remainder of the paper we only show results for CLOOK, a fair and efficient I/O scheduler [14]. One additional thing to note about these figures is that flash-backed I/O request performance varies significantly from workload to workload, even though they are each classified as desktop computing.

One reason flash-backed I/O requests can reduce write latency by up to 70% is that flash-backed I/O request synchronization (back to rotating media) can be deferred until the Request Queue is empty. The initial I/O writing to flash can be faster than writing to rotating media when there is a significant seek penalty on the rotating media.. Figure 8(a) shows the total time spent synchronizing flash-backed I/O requests to rotating media relative to total write-latency without flash-backed I/O requests. This figure shows that at least 20% of total write latency work is performed by idle-time processing, and 90% of the HP workload is synchronized with only 10MB of FQueue. The significance of this result is that with the proposed redirection algorithm we can defer writing to the disk until the Request Queue is empty without any outstanding I/O requests.

The next figure, Figure 8(b), shows the percentage of I/O that occurs due to flush-back I/O. As the number of flush operations increases the benefit of I/O redirection decreases. In these figures we don't flush the entire FQueue, rather, we only



(a) Idle-time FQueue purging (b) FQueue flush purging

Fig. 8. Normalized I/O write access times

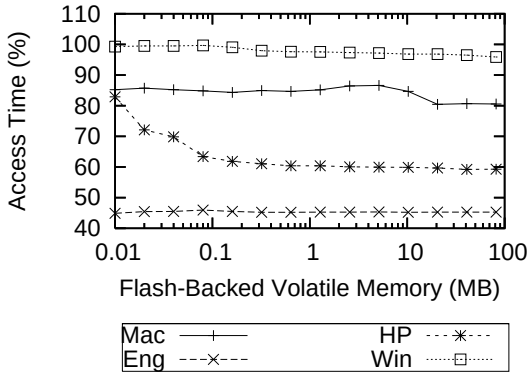
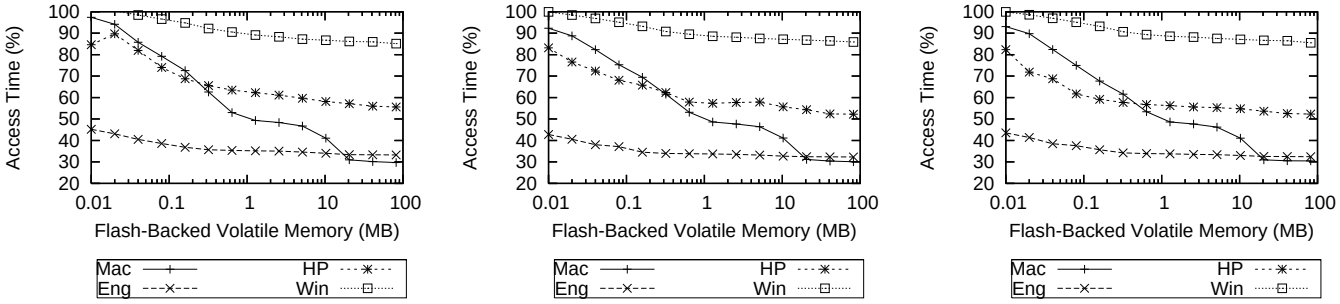


Fig. 9. Normalized I/O write latency with flash-backed I/O requests, including idle-time request purging

flush 1/10th of the FQueue, and since we flush in FIFO order, we flush the oldest requests first, which weren't overwritten or removed through a coalesced I/O.

It is also useful to see what the total write latency is when idle-time processing is included, as it shows total write latency for data to be committed on rotating media. The results of this experiment are shown in Figure 9, and normalized to the total write-latency without flash-backed I/O requests. This figure shows that, if idle-time processing is included in the write latency, the benefit of flash-backed I/O requests is workload dependent. The Eng and HP workloads show that write latency decreases by 30–50%, while the Mac and Win trace actually increase write latency. The increase in write-latency occurs because the cost of pushing requests out of flash and then back rotating media negates the benefit of redirecting to flash in the first place.

In order to assess the impact of using the NVCache for selective write caching, we measured how many erase operations occur during the 7 day traces, and then use the observed frequency to extrapolate the number of years to exceed an NVCache with 100,000 block erase count. We assume an optimal wear-leveling algorithm which spreads all writes across the entire device's physical address space. Figure 10 shows the results of this experiment. This figure shows that the flash endurance increases linearly with respect



(a) FCFS (b) SSTF (c) CLOOK

Fig. 7. Normalized I/O write access with flash-backed I/O requests

to the flash size, and that there is a 10/1 ratio between the NVCache size (MB) and endurance years.

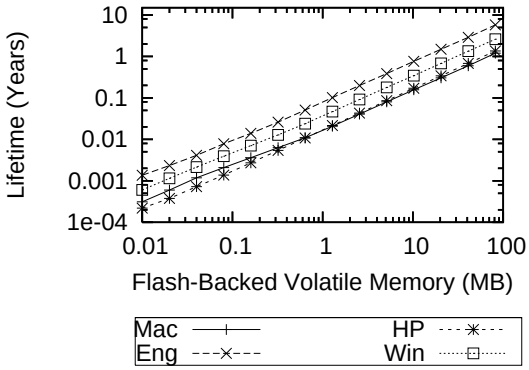


Fig. 10. Years to exceed 100K block erase cycles

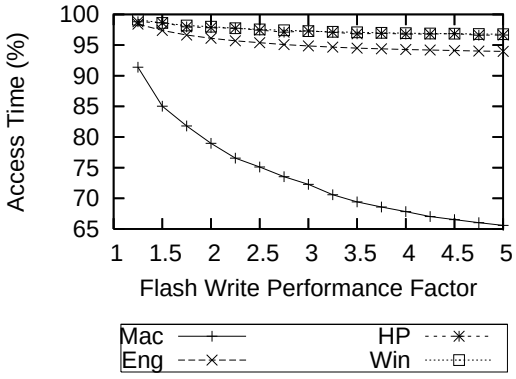


Fig. 11. Scaling Flash Write Performance with 100MB

Figure 12 shows the percentage of DRAM FQueue overwrites. This is an important figure because it shows what percentage of redirected I/Os that do not have to be coalesced or flushed back to rotating media. This figure shows that as the size of the FQueue grows more operations that are redirected to the FQueue result in overwrites - at most 15% of all writes are redirected.. If all I/Os were cached in the FQueue, this fraction would be 100% higher but would also contribute to a significantly larger access time.

Figure 13(a) shows total disk time, which can be translated into duty cycle. These results are also normalized to no

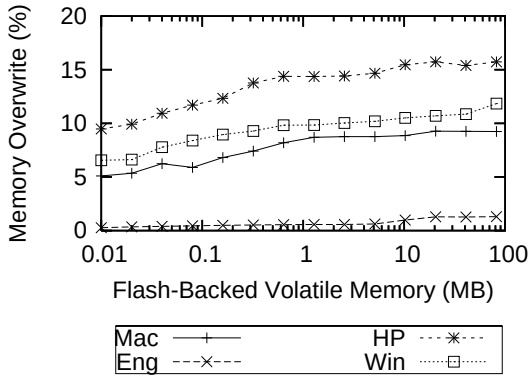
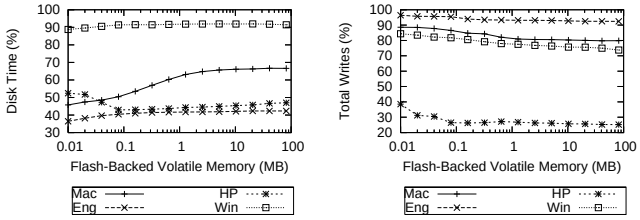


Fig. 12. In-memory overwrites



(a) Total Disk Time (b) Total Disk Writes

Fig. 13. Disk duty cycle

FQueue. One of the benefits of redirecting I/O to flash, in addition to reducing write latency, is that we can increase the chances of coalescing and overwrite operations, thus reducing the effective rotating media duty cycle (one of many variables that dictate hard disk reliability). Figure 13(b) shows the total disk writes normalized to total disk writes without an FQueue, which can also be interpreted as a form of duty cycle. These results show that total normalized disk writes decreases slightly which means that by adding the FQueue, more requests are merged or forgotten because of FQueue use.

V. Related Work

There are several works that propose or leverage a non-volatile cache to reduce disk I/O. Ruemmler and Wilkes [13], Baker *et al.* [15], and Hu*et al.* [16] have all argued that NVRAM can be used to reduce disk I/O. Hu's Rapid-Cache leverages NVRAM to aggregate larger writes to a log-disk, whose content is eventually written out to a backing store.

This 2-level hierarchical caching model improves upon other systems which also use NVRAM, such as WAFL [9].

Haining *et al.* also investigated the use of non-volatile caches to buffer disk I/O [17], [18]. Their primary focus was on NVRAM cache replacement policies. Their most successful algorithm leveraged track-layout knowledge along with spatial and temporal locality to reduce disk I/O. Our work differs in that we are selective about what I/O is redirected to the non-volatile backing store. Therefore this work is complementary to our own.

Hybrid disks place a small amount of flash memory logically adjacent to the rotating media. Interfaces to leverage the NVCache are specified in the ATA8 specification [11]. However, implementation functionality is largely left to the manufacturer. Unfortunately, this means most manufacturer hybrid disk technology will not be published. Therefore, it is our goal in this work to propose functionality leveraging hybrid disk technology.

Hybrid disks also present potential optimizations in the area of power consumption [6]. This work leverages hybrid disks in the OS to reduce power consumption, spin-up latency, and wear-leveling impact. It presents four algorithms exploiting I/O that occurs while the rotating media is spun-down.

There are several I/O schedulers in literature which increase performance. Tangential to our work is Disk Mimic. Disk Mimic develops an I/O scheduler by using shortest-mimicked-time-first (SMTF). Disk Mimic records access time given a set of input parameters such as I/O type and logical sector distance from previous requests and uses the recorded access times to schedule and predict future request service times [2]. The Anticipatory I/O scheduler also improves I/O performance by pausing briefly before servicing a request from another process, as many applications have very short idle-times between I/O requests [3]. If the application issues an I/O within that time-window, the anticipatory scheduler will service that request first.

VI. Conclusions

Disk write latency is a significant component of the overall I/O bottleneck because applications often request acknowledgment that written data is safely placed on non-volatile media. Upcoming disk drives (hybrid disks) will place a small amount of flash adjacent to the rotating media, which can be used to store specific sectors on an alternative non-volatile media. Since flash memory I/O characteristics differ from that of rotating media, an operating system can exploit both media types to reduce write latency of hybrid disks.

In this paper we have proposed to use the flash memory to reduce write latency by selectively caching write requests to the NVCache. Writes are cached to flash if it results in lower access time than servicing it with rotating media. The redirection algorithm also considers the location of other requests in the queue to ensure that the rotating media does service requests. Fundamentally, the NVCache cached write content serves as a backup as it also persists in the main-memory request queue. Cached writes are written to rotating media when they can be coalesced with normal write requests going to rotating media. If the NVCache becomes full, redirected requests are flushed to rotating media, but because data is still located in DRAM, a read request from flash is unnecessary.

Our results show that by using flash-backed I/O requests, we can significantly reduce the write latency by up to 70% with 80MB of both NVCache and DRAM. A major contributor to this performance improvement is idle-processing capability flash-backed I/O requests introduces.

References

[1] M. Kryder, "Future storage technologies: A look beyond the horizon," http://www.snwusa.com/documents/ presentations-s06/MarkKryder.pdf, 2006.

[2] F. I. Popovici, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Robust, Portable I/O Scheduling with the Disk Mimic," in *Proceedings of the USENIX Annual Technical Conference (USENIX '03)*, San Antonio, Texas, Jun 2003, pp. 297–310.

[3] S. Iyer and P. Druschel, "Anticipatory scheduling: a disk scheduling framework to overcome deceptive idleness in synchronous I/O," *SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 117–130, Dec 2001.

[4] Samsung, "Samsung to unveil first commercial, hybrid hard drive prototype for windows vista at winhec," http://www.samsung.com, May 2006.

[5] R. E. Born, "The low-profit trap in hard disk drives, and how to get out of it," Apr 2001.

[6] T. Bisson, S. A. Brandt, and D. D. Long, "A hybrid disk-aware spin-down algorithm with I/O subsystem support," in *Proceedings of the 26th IEEE International Performance, Computing and Communications Conference*, Apr 2007.

[7] S. C. Tweedie, "Journaling the linux ext2fs file system," in *Proceedings of the Fourth Annual Linux Expo*, May 1998.

[8] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Transactions on Computer Systems*, vol. 10, no. 1, pp. 26–52, Feb 1992.

[9] D. Hitz, J. Lau, and M. A. Malcolm, "File system design for an NFS file server appliance," in *Proceedings of the USENIX Winter 1994 Technical Conference*, Jan 1994, pp. 235–246.

[10] M. Trainor, "Overcoming disk drive access bottlenecks with intel robson technology," http://www.intel.com/technology/magazine/computing/robson-1206.htm, 2007.

[11] C. E. Stevens, "At Attachment 8 - ATA/ATAPI Command Set (ATA8-ACS)," http://www.t13.org, 2006.

[12] R. A. Golding, P. B. II, C. Staelin, T. Sullivan, and J. Wilkes, "Idleness is not sloth." in *Proceedings of 1995 USENIX Winter*, Jan 1995, pp. 201–212.

[13] C. Ruemmler and J. Wilkes, "Unix disk access patterns." in *Proceddings of the USENIX 1993 Winter Technical Conference*, Jan 1993, pp. 405–420.

[14] B. L. Worthington, G. R. Ganger, and Y. N. Patt, "Scheduling algorithms for modern disk drives." in *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, May 1994, pp. 241–252.

[15] 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 Systems*, Oct 1992, pp. 10–22.

[16] Y. Hu, T. Nightingale, and Q. Yang, "Rapid-cache a reliable and inexpensive write cache for high performance systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 290–307, Mar 2002.

[17] T. R. Haining and D. D. Long, "Management policies for non-volatile write caches," in *Proceedings of the 18th IEEE International Performance, Computing and Communications Conference*, Feb 1999, pp. 321–328.

[18] J.-F. Pâris, T. R. Haining, and D. D. E. Long, "A stack model based replacement policy for a non-volatile write cache." in *Proceedings of the 17th IEEE Symposium on Mass Storage Systems*, Mar 2000, pp. 217–224.