

# Accelerating Enterprise Solid-State Disks with Non-Volatile Merge Caching

Clinton W. Smullen, IV      Joel Coffman      Sudhanva Gurumurthi

Department of Computer Science

University of Virginia

Charlottesville, Virginia

Email: cws3k@cs.virginia.edu    joel.coffman@email.virginia.edu    gurumurthi@virginia.edu

**Abstract**—Flash memory is now widely used in the design of solid-state disks (SSDs) as they are able to sustain significantly higher I/O rates than even high-performance hard disks, while using significantly less power. These characteristics make SSDs especially attractive for use in enterprise storage systems, and it is predicted that the use of SSDs will save 58,000 MWh/year by 2013. However, Flash-based SSDs are unable to reach peak performance on common enterprise data patterns such as log-file and metadata updates due to slow write speeds (an order-of-magnitude slower than reads) and the inability to do in-place updates. In this paper, we utilize an auxiliary, byte-addressable, non-volatile memory to design a general purpose *merge cache* that significantly improves write performance. We also utilize simple read policies that further improve the performance of the SSD without adding significant overhead. Together, these policies reduce the average response time by more than 75%, making it possible to meet performance requirements with fewer drives.

**Keywords**—Servers; Storage; Solid-State Disks

## I. INTRODUCTION

### *The enterprise storage power challenge:*

Enterprise datacenters use a large number of hard disk drives (HDDs) to meet both capacity and performance requirements. However, HDD performance improvements significantly lag that of processors, and estimates by Freitas and Wilcke show that, within the next decade, enterprise servers will require millions of HDDs to achieve the required level of performance [7]. Storage already represents more than a third of a typical datacenter’s direct power consumption [27], as shown in Figure 1, and can be over 70% for storage-heavy installations [33].

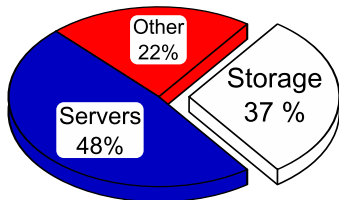


Figure 1. Typical datacenter power breakdown [27]

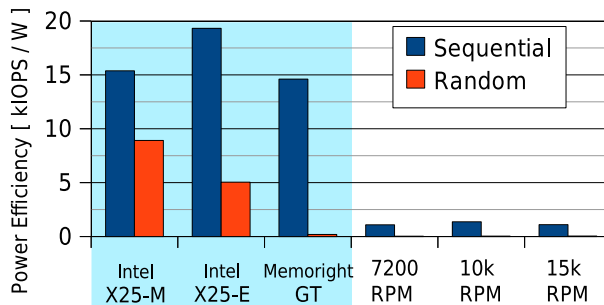


Figure 2. Power efficiency of writes for various storage devices. The left three drives (highlighted) are solid-state disks (SSDs) [Performance data from Polte et al. [24], with average power estimated from datasheets and online reviews.]

The growing interest in Flash-based SSDs is due their ability to match or exceed the performance of high-speed HDDs while using less power, as shown in Figure 2. Improved SSD design has largely mitigated the reliability issues of Flash, and they are now available from a diverse range of manufacturers, from storage companies like Seagate and Fusion-io to traditional chip companies like Intel and Samsung.

Narayanan et al. predict that a smaller number of SSDs can displace many of the high-power, high-performance hard disks [22]. As Flash-based SSDs cannot match the cost-per-GB and GB-per-Watt of hard disks, energy efficient disk drives will still be required for capacity [11] [12] [29], while the SSDs hold the most active data. The significant reduction in the required number of drives (both SSDs and hard disks) can significantly improve energy efficiency as the needs of data centers continue to grow. It has been predicted that the addition of SSDs to enterprise servers will save 58,000 MWh/year by 2013 [25] [31]. However, before Flash-based SSDs can become ubiquitous in enterprise storage systems, a key performance bottleneck must be overcome.

### *The in-place update problem:*

Polte et al. have shown that enterprise-level Flash-based SSDs can provide more than two orders-of-magnitude improvement in performance over hard disks

for random access patterns [24]. We have incorporated published power data on six of the storage devices they tested to estimate the power efficiency (in thousands of I/O operations per second (IOPS) per Watt) as shown in Figure 2. The Flash-based SSDs, shown highlighted on the left, significantly improve both the random and sequential power efficiency over any of the hard drives, though a gap between the two values persists. As Flash memory is a (mostly) random access device, the gap shows that further improvements are possible.

The difference between the sequential and random write performance is explained by the inability of Flash to perform in-place updates. Modifying a small piece of data necessitates writing the data to a new Flash page, leaving stale data within the original erase-unit. Large numbers of random writes, common in many enterprise workloads, will increase the fragmentation within erase-units, reducing performance as expensive merge operations must be performed. This magnifies the already slow speed of writes, preventing Flash-based SSDs from achieving their full potential. This reduces the energy savings and increases costs by requiring a larger number of SSDs to meet the same performance requirements.

Previous work in making modifications to the file-system and changing the disk-block interface have shown that adding byte-addressable non-volatile memory (NVM) can significantly improve both the performance and the endurance of Flash-based SSDs [6], [14]. However, the changes required by these previous proposals prevent them from being a drop-in replacement for magnetic hard disks.

Our contributions are as follows:

- We use byte-addressable NVM for both Flash Translation Layer (FTL) data and as a general purpose merge cache. This architecture requires neither a new file system nor modifications to the device interface. As the merge cache can hold data indefinitely, the system can coalesce writes, thus reducing the number of writes and erasures that the Flash must perform and improving SSD endurance.
- As the byte-addressable NVM performs significantly better than Flash for both reads and writes, we explore a number of policies for managing the contents of the merge cache to improve performance. We also investigate how the capacity of the merge cache affects the performance of the best management policies.
- We show that our general purpose merge cache improves performance by over 75%, on average, for enterprise-level workloads. Additionally, the use of the merge cache reduces the number of erasures needed by more than 20%, on average.

We next provide an overview of the nature of Flash, FTLs, and existing auxiliary NVM designs in Section II. In Section III, we characterize our enterprise workload traces and describe why our design works well for them. The details of our design and the merge cache management policies that we explore are given in Section IV, Section V describes our evaluation framework and results, and Section VI offers our conclusions and opportunities for future work.

## II. BACKGROUND AND RELATED WORK

We now describe the architecture and limitations of Flash-based SSDs in Section II-A, the operation of the most common types of Flash Translation Layers in Section II-B, while Section II-C finished with a discussion of the related work.

### A. Flash-based SSDs Primer

Flash-based SSDs use a Flash Translation Layer (FTL) to provide the block-based interface of a hard disk. Reads and writes take roughly the same time on a hard disk, while writing to Flash is almost an order-of-magnitude slower than reading. Furthermore, a region of Flash must be erased before it may be written, an operation two orders-of-magnitude slower than reading, and each region may only be erased a limited number of times. The purpose of the FTL is to mask this behavior from the host system.

Flash has a minimum write size of a page, which is typically 4 kB for the large-block NAND Flash used in SSDs. These pages are grouped together to form erase units around 256 kB in size, which forms the smallest erasable unit. The FTL uses copy-on-write semantics to mask the behavior and latency of writes and erasures by changing the mappings for each logical block as needed. The old versions of a page continue to occupy space until their units are erased. However, any valid pages still in the erase unit must first be *merged* onto a new one during *garbage collection*. As each erase unit has a 10,000–100,000 program/erase cycle limit, it is necessary for the FTL to perform wear-leveling.

### B. Designing the Flash Translation Layer

We now summarize the relevant FTL designs here, but Gal and Toledo have reviewed many other proposals [8]. The two traditional schemes for storing the logical to physical page mappings are page-based and block-based. A page-based FTL can map any logical page to any physical page, a strategy reminiscent of fully associative caches, simply requiring a large amount of fast storage to hold the mapping table. A block-based FTL performs the same translation but restricts which erase units a logical block can map to, similar to

a set-associative cache. This map is at least an order-of-magnitude smaller than a page-based map. However, block-based schemes experience a much higher number of merges as each datum can only go in a limited number of locations.

Hybrid FTL schemes attempt to bridge the gap by partitioning Flash into data blocks and log (or update) blocks [5], [17], [18]. Data blocks are mapped using a block-based scheme while log blocks are mapped using a page-based scheme, reducing the total size of the mapping. The log blocks receive all of the updates, minimizing the amount of data that must be copied, and keeping garbage collection overheads low. When the device exhausts the set of free log blocks, the garbage collector must merge the log and data blocks and rewrite the result to already erased blocks, an expensive operation.

DFTL provides a page-based FTL variant that requires only the scratchpad SRAM within the SSD’s controller [10]. Since the capacity of the scratchpad SRAM is extremely limited, it is only used to cache the entries of the page-based map. Rather than reconstruct the missing FTL entries when a cache miss occurs, DFTL writes complete segments of the page-based map to Flash on a cache eviction. Gupta et al. showed that DFTL significantly reduces the number of expensive merge operations required over hybrid FTL schemes, though it does not perform as well as a simple page-based FTL, as it requires extra reads and writes to Flash to save and restore segments of its map.

### C. Augmenting SSDs with NVM

Despite the large number of publications on non-volatile memory (NVM) technologies, few have investigated SSDs that combine multiple technologies. Designed for magnetic disks, the HerMES file system uses magnetoresistive RAM (MRAM) to buffer writes and hold all metadata [20]. Doh et al. proposed a file system for Flash memory, MiNV, that stores all metadata in NVM and all file data in Flash, showing that it improved performance while reducing the number of Flash erase operations needed [6]. Kim et al. [14] proposed using phase-change memory (PCM) to hold metadata along with a page-based FTL called hFTL. They showed that moving both FTL and file-system metadata updates off of Flash significantly improves performance while reducing the number of Flash writes and erasures. However, these implementations require modifying the block-device interface and software to identify the metadata.

Though effective, the amount of space required to store all of the file-system metadata in NVM quickly becomes prohibitive—roughly 300 MB for a 30 GB

Flash device in the best case. Additionally, validating a file system is a difficult task required before a file system will be adopted in enterprise systems. Significant device interface changes hinder adoption as they prevent devices from being used as drop-in replacements for magnetic hard disks. The recently added SSD `trim` command improves performance by allowing the OS to indicate blocks belonging to erased files, thus reducing garbage collection overheads. However, the command does not change the underlying device abstraction and SSD manufacturers currently do not rely on it for good performance, as software support is still limited.

Sun et al. designed a hybrid FTL that stores its log blocks in PCM [30]. Because PCM is byte-addressable, updates may be performed in-place although care must be taken to ensure the log region does not wear out before the underlying Flash. Their dynamically allocated log provides superior performance to both in-place logging (where the log is written to Flash) and static log allocation by associating an arbitrary number of log blocks with each Flash erase unit. However, freeing log entries requires merging an entire log for some erase unit back to Flash, dramatically increasing the cleaning overhead. To mitigate this, Sun et al. use 1 GB of PCM to support 32 GB of Flash.

We build on the performance and flexibility of page-based FTLs shown by Gupta et al. [10], though we assume there is enough auxiliary memory (DRAM or NVM) to hold the entire page-based map. Our design requires neither file system modifications nor does it distinguish between metadata and ordinary file data. Our merge cache is general enough to improve the performance of any small write pattern and may be used as a drop-in replacement. Additionally, none of these designs have used the auxiliary memory to buffer reads as well as writes.

## III. WORKLOADS

To evaluate our system, we selected four enterprise server traces collected from production systems in Microsoft data centers [13]. The **Exchange** server is a corporate mail server with more than 5000 users. The **MSN** file server supplies the files requested by various Live data services. The Live **Maps** trace comes from the Virtual Earth servers responsible for retrieving map imagery and combining it other information before presenting it to users. **RADIUS** is an authentication server responsible for worldwide corporate remote access and wireless authentication.

Because many of the original traces touch far more than 32 GB of storage, we create a subset of each trace by using all of the accesses within a specified time slice. Even within this time slice, the original addresses may

Workload	Duration	Read / Write Ratio	Small Writes	Overwritten Pages
Exchange	1 hour	0.35 : 1	21.2%	78.4%
Maps	2.5 hours	4.21 : 1	2.1%	46.2%
MSN-FS	20 minutes	2.08 : 1	15.0%	34.3%
Radius	16 hours	0.12 : 1	39.4%	82.8%

Table I  
CHARACTERISTICS OF THE WORKLOAD TRACES

not be confined to 32 GB so we remap the addresses in the original trace to compressed addresses that fit within the device we simulate. Our remapping function provides the following three guarantees: (i) sequential accesses in the original trace are guaranteed to be sequential in the compressed trace, (ii) the transformation of addresses at a page granularity ensures that the number of Flash pages accessed is unchanged, and (iii) the transformation will not create new sequential accesses. Unlike disks where the position of the disk arm and the address play a significant role in latency, SSDs have constant latencies for sequential and random accesses. Hence, our compression of sparse addresses into a dense address space will not significantly alter the performance of the simulated system, which would not be true for disks.

Table I shows the characteristics of our workload traces, with a small write being any write less than a page (4 kB) in total size. Between the small writes and the large fraction of overwritten data, our proposed merge cache may prove extremely useful on an SSD. The non-volatile merge cache can potentially coalesce these small writes, providing better performance and reducing the wear on the Flash memory. We note that the merge cache will be most beneficial for workloads with a large number of small writes (e.g., Radius and Exchange) because this access pattern is the worst for Flash-based SSDs.

#### IV. DESIGN

In this section, we describe the design of our system and how it can improve performance. Our designs are based on the enterprise SSDs investigated by Polte et al., as these devices display the superior performance and reliability that is critical in server environments. We start with our selection of a secondary NVM technology to augment the Flash already present in the SSD, though we note that our design is not dependent on a particular NVM technology. Next, we describe our system architecture (shown in Figure 3) and the structures stored in the NVM. Finally, we discuss a number of policies that may be used to manage the merge cache.

##### A. Auxiliary NVM Selection

The design of the merge cache necessitates the ability to write (and easily rewrite) logical block (512 B) sized datum into the auxiliary NVM. This NVM augments the Flash already present in the SSD; we simply use the NVM as a replacement for the DRAM that is part of enterprise SSDs [21]. Replacing DRAM with NVM reduces the idle power consumption of the SSD while providing similar access times. Density is not a significant concern since we require only a small amount ( $\approx 64$  MB) of auxiliary NVM. Potential candidates include NOR Flash, small-block NAND Flash, PCM, and MRAM. NOR Flash permits word-level writing but requires erasing the entire logical chip at once as well as having write latencies up to 900 ms. Small-block NAND Flash uses a page size matching that of a logical block [19], but it still requires erasing at a larger granularity than a page, complicating management.

Both PCM and MRAM are byte-addressable NVM technologies, but PCM suffers from limited write endurance (currently in the range of  $10^4$ – $10^8$  set-reset cycles [16], [26], [28]) and has write speeds an order of magnitude slower than DRAM [15]. In contrast, MRAM has practically unlimited endurance but is significantly lower in density and higher in cost than Flash. In an effort to provide an SRAM-compatible interface, most MRAM chips equalize the speeds of reads and writes even though writes are slightly slower in practice. Spin torque transfer RAM (STT-RAM) is a new variant of MRAM under active industrial development [9] that has the potential to solve the two traditional drawbacks of MRAM: high write currents and large cell sizes [4], [15], [32]. Many other technologies are also in development, but they have yet to reach the maturity of the technologies listed above. Though it is presently more expensive than competing technologies, we have chosen to model MRAM in our device because it allows us to focus on the design of the merge-cache itself.

##### B. Auxiliary NVM Use

The auxiliary NVM in our design is split into two segments: the first holds the page-based FTL and the second the merge cache itself. Both segments also contain auxiliary structures to manage the FTL and the

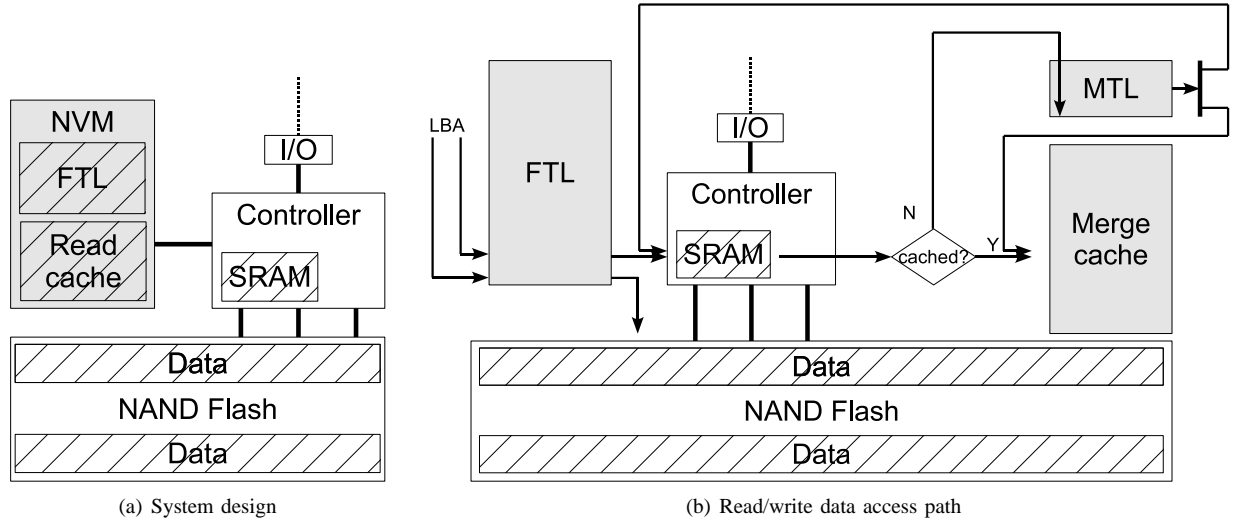


Figure 3. Conceptual architecture – the NVM contains the page-based FTL, merge cache, and auxiliary structures related to the merge cache’s bookkeeping.

merge cache. The FTL data need not reside in NVM, but it does obtain the side benefit of reducing startup time. The merge cache and its management structures do require non-volatility to ensure the data is persistent.

Like other page-based FTLs, our in-memory direct map is indexed by the logical page number and translates it to a physical Flash page. Given our assumed Flash page size of 4 KB, we require 23 b per entry for the physical page in the  $\approx 30$  GB SSD we model. The entry also stores a bit indicating whether any sectors within the page currently reside in the merge cache. As we use 32 b to hold each entry, this design can support significantly larger amounts of Flash without modification, though it requires more NVM to hold the complete map. We reserve the physical page number of all 1’s to represent an invalid mapping, supporting software’s use of the `trim` command. This allows the OS to tell the SSD that the corresponding data blocks may be immediately reclaimed.

We use an inverted index to track all of the merge cache entries. The inverted index maps an NVM physical sector (512B) to a logical block address. If the inverted index contains an entry for a given logical block address, its location in the merge cache is given by the entry’s NVM physical sector. Given the temporal locality exhibited by many enterprise workloads [10], our design caches a portion of the inverted index’s entries within a small (256 kB) hash table stored in the disk controller’s SRAM scratchpad and indexed by the logical page number, which can hold more than  $\frac{1}{7}$  of all the entries in our default configuration. We exclude the cache from further discussion because it is very effective even with only these simple management policies.

Figure 3(a) shows the SSD design with NVM replac-

ing the existing DRAM, and Figure 3(b) shows the read and write access path for I/O operations. When an I/O operation enters the SSD, the FTL identifies whether or not the requested data is stored in Flash and if any updated blocks reside in the merge cache. If found, the SRAM cache is probed using the logical page address to find the entry for the page. On a miss, a linear search of the inverted index is performed to reconstruct the missing cache entry. Regardless, the entry identifies the NVM physical sectors that hold blocks from the page, which can then be accessed. For a write, misses and *partial hits*, where only some of the blocks have been allocated to the merge cache, are given to the write policy to decide whether to allocate them to the merge cache or bypass them to Flash. A write that fully hits in the merge cache can bypass the write policy completely. Reads are always forwarded to Flash on a miss or partial hit.

### C. Merge Cache Policies

The management of the merge cache provides a rich design space for exploration. The write policy determines whether to bypass a write to Flash or allocate new sectors in the merge cache. The optional choice of a read policy decides whether to cache or prefetch pages. We have also explored variations of least recently used (LRU) and most recently used (MRU) eviction policies, but the four workloads used here give nearly identical results regardless of the policy used. We use a traditional LRU policy to ensure the best overall performance.

### Write Policies

The write policy decides whether or not to allocate new entries on a miss or partial hit in the merge cache.

Writes to our MRAM NVM are almost two orders-of-magnitude faster than writes to Flash, and sub-page writes are even more expensive to perform on Flash, as they require merging the data within the page. Thus, write policies should buffer all sub-page writes and as many other writes as possible in the NVM.

- **Writebuffer:** The most straightforward policy is to allocate every write request to the merge cache. Sectors that already reside in the merge cache are simply updated, while the eviction policy ensures that there is enough free space in the merge cache to allocate the remainder. This policy leverages the reduction in write latency the auxiliary NVM provides to its fullest, though it may be unable to reduce the number of merge operations needed.
- **Sub-page Writebuffer:** This policy directly targets sub-page Flash merge operations by allocating merge cache entries only for sub-page writes. Writes that update an entire page bypass the merge cache and are sent immediately to Flash, silently evicting any sectors currently residing in the merge cache for that page. This policy conserves merge cache space while giving up the write latency benefits of the full writebuffer policy.
- **Saturating Sub-page Writebuffer** If an entry in the merge cache is subsequently overwritten, it is possible that it will be overwritten again. If a write covers all the sectors in a page in the merge cache, this policy allocates the remaining sectors to the merge cache as well. Thus, subsequent updates will hit completely in the merge cache. This conserves less space than the sub-page writebuffer policy, while attempting to better leverage the write latency benefits of the auxiliary NVM.

### *Read Policies*

Most read caching and prefetching policies are designed for use in the operating system, which is able to analyze the traffic for each application independently, or for magnetic disks, where rotational latency dominates access time and the incremental cost of reading subsequent disk blocks is extremely low. The shift from mechanical hard disk drives to Flash SSDs changes the nature of both the algorithms and the data that should be cached and prefetched. Reading one or all of the blocks within a page takes about the same time, but reading blocks in two different pages may take twice as long, regardless of where the pages are. As a miss in the merge cache for any block of a page requires accessing the entire page, all of our read caching and prefetching policies operate on full pages.

The NVM can perform a read an order-of-magnitude faster than Flash. However, the read policy should not

impede the write policy as writes to Flash are even slower. As data cached or prefetched is simply a copy of data residing in Flash, it is not necessary that it be held in NVM, and DRAM could be reintroduced to separately hold this data, though this mandates a fixed partitioning between the read and write policies. Instead, we allow the policies to transparently share the same pool of fast storage and simply use more conservative read policies to minimize the impact on the write policy.

The policies presented below require no extra book-keeping and only a negligible amount of computation. Though not mandatory, we add a dirty bit for each page in the merge cache to eliminate unnecessary writebacks. The conditions used by the policies below to selectively cache or prefetch data may seem somewhat arbitrary, but they maintain the simplicity of the read policies while significantly reducing their impact on the write policy.

- **Selective Read Caching:** With the ONFI 2.0 interface, the difference in transferring one block or an entire page from Flash is very small. The simplest approach is thus to cache all pages as they are read from Flash, though this pollutes the merge cache with much unused data. Instead, we choose to cache only the last page of a read request, and only if the last logical block within the page was not requested. This improves the performance for sequential access patterns that are not aligned to page boundaries. This policy will not pollute the merge cache as heavily as traditional read caching.
- **Selective Read Prefetching:** The selective read caching policy can work well for sequential access patterns, but it stops at the page boundary. This policy actively prefetches the page immediately following the end of a read request. Unlike for read caching, this policy may significantly increase latency as it sends extra read requests to Flash. To mitigate this problem and reduce the pollution of the merge cache, we only prefetch the page when the read request accessed the last logical block within the last page in the request. To prevent prefetching from interfering with the read request, the pages to prefetch are appended to a queue that is processed only after the request has been handled.
- **Selective Read Caching and Prefetching:** As the two policies above are mutually exclusive, it is trivial to combine the two together. This combined policy should improve performance for many sequential access patterns.

## V. EVALUATION

In this section, we describe our evaluation framework and the simulator we used to evaluate our designs.

Then, we discuss the results derived from simulating the four enterprise workloads described in Section III. We explore the various policies for managing the merge cache before briefly looking at the impact of NVM capacity on the performance of the SSD.

#### A. Simulation Model

Agrawal et al. previously developed a model for Flash-based SSDs [1], implemented as a module for DiskSim 4.0 [3]. It supports a wide range of realistic Flash device configurations using a page-based FTL with no latency. We added support for auxiliary NVM, our merge cache, and the policies from Section IV-C to this model. We model MRAM as using a SRAM interface with symmetric 10 ns access latency, based on values in the ITRS roadmap [2]. We assume there are enough chips to form a 32-bit word, that each chip has at eight internal banks, and that the data and address buses operate at 200 MHz. We assume the Flash chips utilize the Open NAND Flash Interface (ONFI) 2.0 specification, which significantly increases the bandwidth available to transfer data to and from the Flash devices [23]. Our baseline system is organized like Agrawal et al.'s but with the FTL data held in NVM, though this reduces performance by less than 1%

#### B. Results

We start out by modeling the policies described in Section IV-C using 64 MB of MRAM. As the FTL utilizes 29 MB, there is 35 MB left for the merge cache. We also utilize two perfect policies that assume that all requests of the stated type(s) will be satisfied by the merge cache. These policies show the maximum possible improvement that may be achieved from the latency reduction of the NVM. In practice, the difference between the two perfect policies is small, as the latency reduction has much more impact on writes than reads. A design approaching a perfect policy is meeting the locality needs of the workload and leaves little room for further improvement.

*Metrics:* To reduce the potential for confusion, we describe the primary metrics used to evaluate our designs here. These metrics are normalized against either the baseline system without a merge cache or another, stated point of comparison. Normalizing the results eases comparison and allows us to calculate the average improvement across all four workloads.

- **Response Time:** We principally use the average response time to measure the performance improvement of our designs. While I/O operations per second (IOPS) is the more conventional figure of merit, our use of enterprise workload traces that contain timestamps fixes the I/O rate to that of the

system the trace was collected from. The response time for a request is the total time from when the request is first sent by the host system until the entire response is received. As such, reducing the response time potentially allows for higher IOPS as it can service more requests in the same amount of time.

- **Access Time Ratio:** The device access time is a component of the response time; it counts only the time to access the NVM and Flash subsystems within the SSD. We take the ratio of the average device access time to the average response time. When it approaches unity, the response time is dominated by access time and the SSD is performing as efficiently as possible; when it approaches zero, the response time is dominated by queuing and transmission delays. Workloads that have many large requests will always have lower access time ratio due to transmission delays, and queuing delays become significant for workloads with a high request intensity.
- **Erase Operations:** Each Flash block can be erased a limited number of times and is significantly slower than even a write to Flash. As such, reducing the total number of erasures can improve performance and extend the device lifetime.

Section V-B1 and V-B2 evaluate the read and write policies, respectively. We also look at the capacity sensitivity of the best write-only and read-write policy choices in Section V-B3.

1) *Write Policies:* Figure 4 shows the average response time for each workload and write policy combination, normalized to that of the baseline configuration, along with the mean normalized response time across the workloads. All three writebuffer policies reduce the average response time, but the two sub-page writebuffer policies provide negligible improvements for the Maps workload, as expected from the extremely low fraction (2.5%) of sub-page writes. The writebuffer policy performs significantly better on every workload than the two sub-page writebuffer policies, indicating that the latency benefits of the NVM far exceed the slowdown caused by merge cache pollution.

The low write intensity of Radius allows the writebuffer policy to approach the performance of having perfect writes, while the write intensities of Exchange and MSN-FS are so high that the perfect writes policy is indistinguishable from zero. The average response time for Maps is above zero but still well ahead of the writebuffer policy. Despite the significant reduction in response time, only Exchange and Radius show significant hit rates (19.5% and 25.1%, respectively) for pages in the merge cache for writes, though neither shows

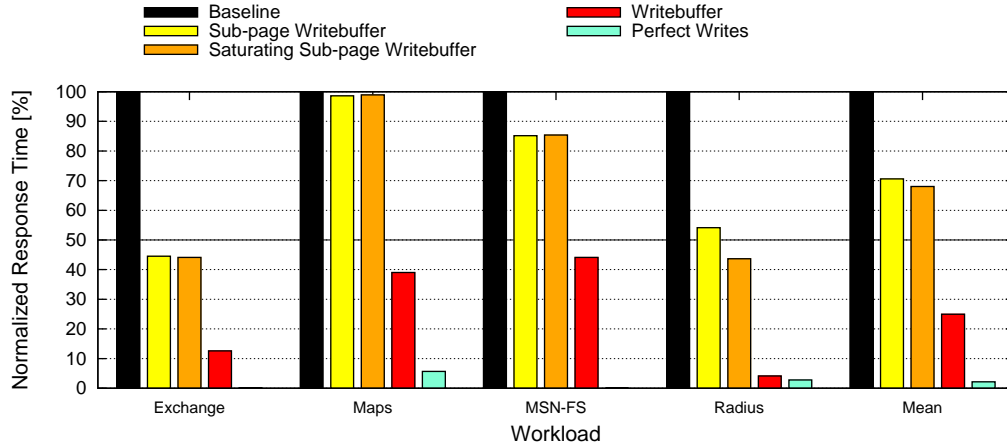


Figure 4. Average response time normalized to the baseline (lower is better) [64 MB MRAM]

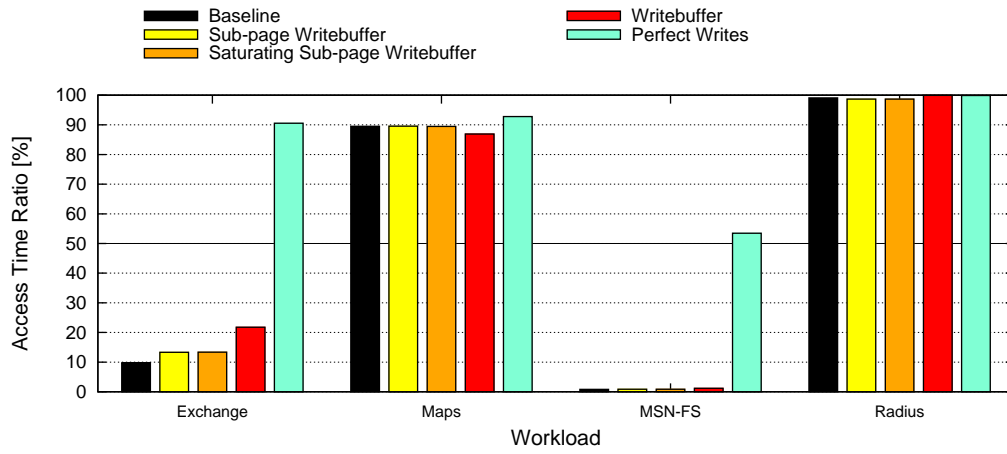


Figure 5. Average access time as a fraction of the average response time (higher is better) [64 MB MRAM]

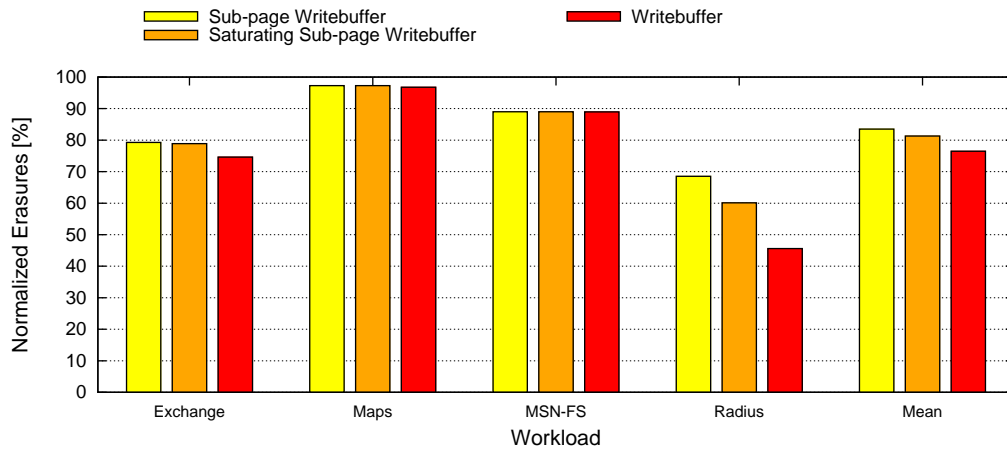


Figure 6. Number of erasures normalized to the baseline (lower is better) [64 MB MRAM with the writebuffer policy]

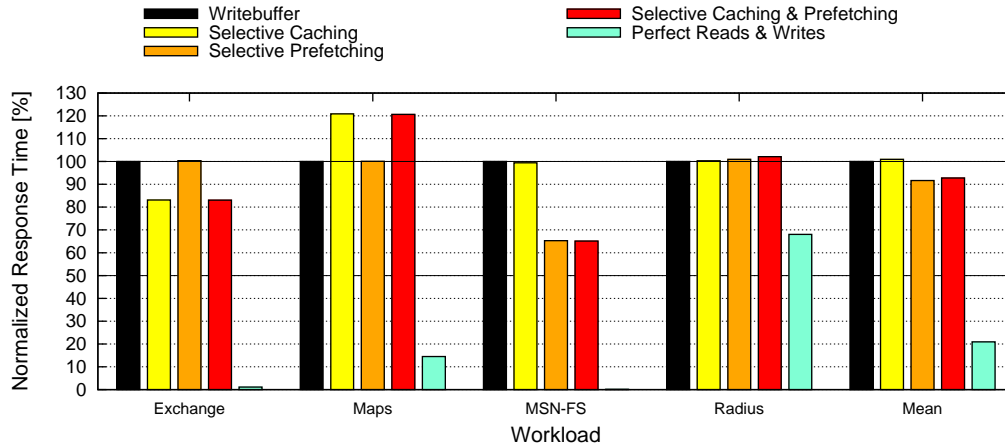


Figure 7. Average response time normalized to the writebuffer policy (lower is better)  
[64 MB MRAM using the writebuffer policy]

sensitivity to the policy used; and for all workloads, the hit rate for reads is extremely low and indifferent to the policy used. The decent write hit rate for Radius is reflected in the fact that it is the only workload to show a significant benefit from the saturating sub-page writebuffer policy over the sub-page writebuffer policy.

Figure 5 shows the average access time ratio, still normalized to the baseline. Maps and Radius have the lowest request intensity, giving them normalized access times near one; and Exchange shows access time ratios commensurate with the performance improvement given in Figure 4, approaching unity only with the perfect writes policy. Though it performs well, this Figure shows MSN-FS suffering from queuing delays caused by high request intensity, only alleviated by the perfect policy. The fact that its access time never approaches unity indicate the presence of a large number of large requests, which require a significant amount of time to transmit data to and from the SSD.

Figure 6 shows the number of erase operations normalized to that of the baseline. The write coalescing nature of the merge cache can truly eliminate erasures, as long as there are hits in the merge cache. As the merge cache has limited capacity, the reduction in erasures may simply be a deferral, as indicated by the low hit rates for writes for Maps and MSN-FS. However, the page write hit rates of 19.5% and 25.1% for Exchange and Radius (using the writebuffer policy) indicate that write coalescing eliminates a large number of erasures.

These results show that, though the sub-page policies provide significantly less performance improvement than the writebuffer policy, all three policies improve performance while reducing the number of Flash block erasures.

2) *Read Policies:* Figure 7 shows the response time for the three read policy variants, each using the write-

buffer write policy evaluated in Section V-B1, along with the perfect reads and writes policy for comparison. Each datapoint is normalized to the writebuffer policy operating without a read policy. Radius, which has an extremely low number of reads compared to writes, shows no significant change, and Maps is also unaffected by the selective prefetching policy as its requests are more random in nature. However, it performs poorly under the selective caching policy due to merge cache pollution, which also carries over into the combination policy, reducing the number of write hits by 13%. The simplest approach to mitigating this is to statically limit the number of pages that may be cached within a given time interval.

Exchange shows an almost 20% improvement in response time from the use of selective caching but sees no further improvement from selective prefetching. MSN-FS shows a 35% improvement using the selective prefetching policy but is insensitive to the selective caching policy. Neither policy is able to bring any of the workloads closer to the performance of having perfect reads and writes, indicating that further improvements require more aggressive read policies with adaptive control to minimize cache pollution.

Combining the selective caching and selective prefetching policies provides the best performance for Exchange and MSN-FS and performs well overall.

### 3) *Capacity Sensitivity:*

Figure 8 shows the normalized response time for MRAM capacities ranging from 32 MB to 256 MB using either the writebuffer policy by itself (Figure 8(a)) or the writebuffer policy in conjunction with the selective read caching and prefetching policy (Figure 8(b)). As the FTL data requires 29 MB of space, this means that the smallest capacity has only 3 MB for the merge

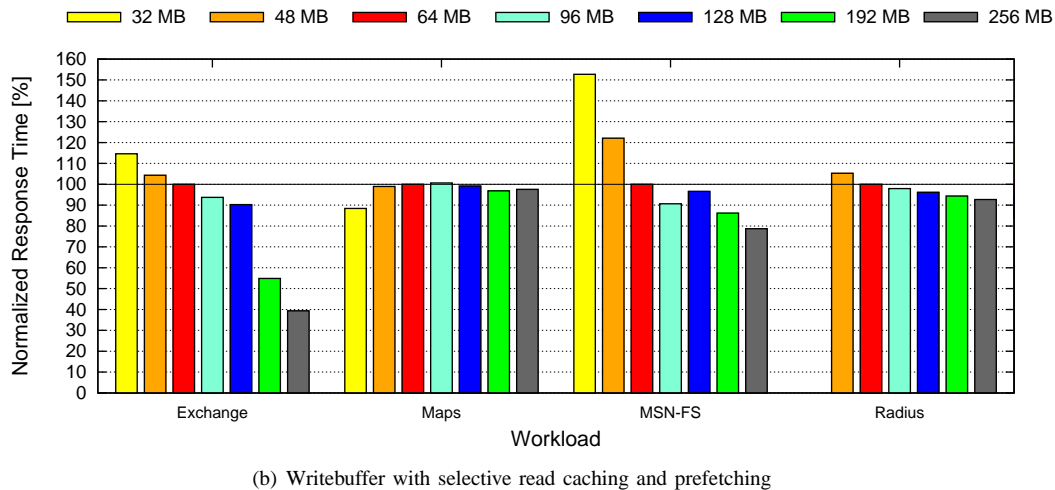
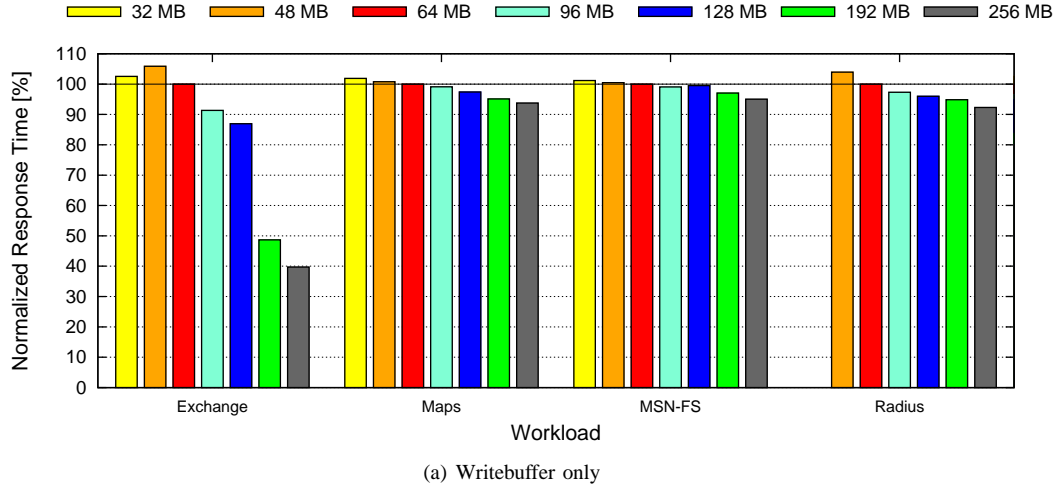


Figure 8. Average response time normalized to 64 MB capacity (lower is better)

cache. Despite this, the performance of the real-world workloads is not significantly reduced by shrinking the MRAM capacity.

In fact, Figure 8(a) shows that only Exchange performs significantly better as the NVM capacity is increased. The large jump in performance at 192 MB indicates that Exchange has a large working set for writes. Radius fails to operate with the 32 MB NVM as it issues write requests larger than the merge cache can handle when using the writebuffer policy. Modifications could be made to overflow such data to Flash, similar to what the sub-page writebuffer policy does.

Figure 8(b) shows nearly identical results for Exchange, Maps, and Radius. Exchange shows a larger reduction in performance going to 32 MB than it did with no write policy, and Maps continues to suffer reduced performance due to the selective caching, indicating that the cache pollution can only be mitigated by

unattainably large NVM sizes, reinforcing the need for either limits on the caching policy or adaptive control to improve selectivity. MSN-FS shows the same caching and prefetch efficiency regardless of capacity, but the write hit rate varies directly with capacity. This shows that, though the read policy is working well, it can limit the effectiveness of the writebuffer policy. Overall, lower NVM capacities interact poorly with this read policy, though the addition of adaptive control may be able to prevent the significant loss of performance due to cache pollution.

If the cost of MRAM is a significant concern, these results show that a design can achieve the majority of the performance benefits by continuing to use DRAM to hold the FTL data while using 4–16 MB of MRAM with the writebuffer policy for the merge cache. Increasing the amount of either DRAM or MRAM enables the use of selective read caching and prefetching, providing significant performance improvement for some workloads.

### C. Summary of the Results

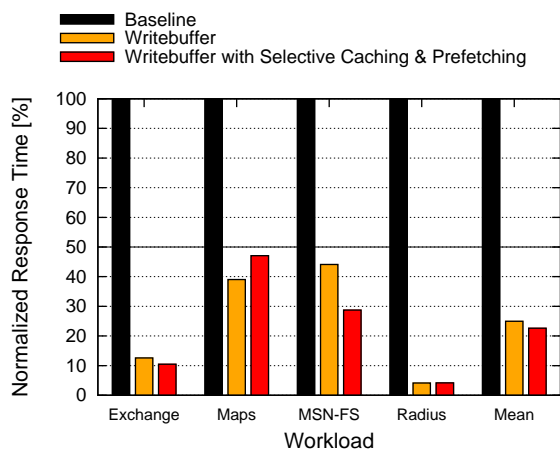


Figure 9. Average response time normalized to the baseline (lower is better)

[64 MB MRAM]

Figure 9 summarizes the performance benefits of our best-choice read and write policies. Using a write policy improved the performance of each workload, though the writebuffer policy is the most effective policy as it better masks the order-of-magnitude latency difference between Flash reads and writes. The simple read policies that we have presented improve the performance of some workloads while adding almost no overhead. Furthermore, the write coalescing performed by the merge cache reduces the number of moves, merges, and erase operations on Flash, all of which can improve the lifetime of the SSD.

### VI. FUTURE WORK AND CONCLUSION

The superior I/O and throughput performance of Flash-based SSDs compared to magnetic disk drives is driving their adoption in enterprise datacenters both to improve performance and reduce power consumption. However, the high latency for writes and the inability of Flash to perform in-place modifications limits the benefits for many enterprise workloads. We have shown that adding a merge cache made from an auxiliary NVM significantly improves the performance of real-world enterprise workloads. Our policies reduce the average response time of the SSD by more than 75% as well as reducing the number of Flash erasures by over 20%. This could help reduce the total number of drives needed, thereby reducing both power consumption and costs.

One of the most significant benefits of our architecture is its flexibility. The performance degradation of Maps under the caching read policies indicates that further improvements are needed. The addition of an adaptive control system could either tune policy parameters on the fly, or it could simply measure the

performance of different policies and choose the best candidate. Though we have used MRAM to model our NVM here, PCM is being actively researched and is predicted to become commercially available in the near future, providing significantly higher densities. A merge cache designed using PCM needs policies that leverage the increased density while mitigating the limited write endurance and low write performance. Similarly, the differences in device characteristics between STT-RAM and MRAM deserve a STT-RAM-specific model and policy optimization.

### ACKNOWLEDGMENTS

We would like to thank our shepherd, Dave Lowenthal, for his feedback and assistance with preparing the final draft of the paper. This research has been supported in part by NSF CAREER Award CCF-0643925 and gifts from Google and HP.

### REFERENCES

- [1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design Tradeoffs for SSD Performance. In *ATC '08: Proceedings of the 2008 USENIX Annual Technical Conference*, pages 57–70, 2008.
- [2] W. Arden, P. Coge, M. Graef, H. Ishiuchi, T. Osada, J. Moon, J. Roh, H.-C. Sohn, W. Yang, M.-S. Liang, C. H. Diaz, C.-H. Lin, P. Apte, B. Doering, and P. Gargini. *International Technology Roadmap for Semiconductors*. Semiconductor Industries Association, <http://www.itrs.net/>, 2007.
- [3] J. S. Bucy, J. Schindler, S. W. Schollosser, and G. R. Ganger. The DiskSim Simulation Environment Version 4.0 Reference Manual. <http://www.pdl.cs.cmu.edu/PDL-FTP/DriveChar/CMU-PDL-08-101.pdf>, May 2008.
- [4] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy. Overview of candidate device technologies for storage-class memory. *IBM Journal of Research and Development*, 52(4):449–464, 2008.
- [5] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song. System Software for Flash Memory: A Survey. In *Proceedings of the International Conference on Embedded and Ubiquitous Computing*, pages 394–404, August 2006.
- [6] I. H. Doh, J. Choi, D. Lee, and S. H. Noh. Exploiting Non-Volatile RAM to Enhance Flash File System Performance. In *EMSOFT '07: Proceedings of the 7th ACM & IEEE International Conference on Embedded Software*, pages 164–173, October 2007.
- [7] R. F. Freitas and W. W. Wilcke. Storage-Class Memory: The Next Storage System Technology. *IBM Journal of Research and Development*, 52(4):439–447, 2008.

- [8] E. Gal and S. Toledo. Algorithms and Data Structures for Flash Memories. *ACM Computing Surveys*, 37(2):138–163, 2005.
- [9] Grandis Inc. Announces the First 300 mm MTJ Fabrication Facility in the United States. [http://grandisinc.com/pdf/Feb2\\_2009\\_Grandis\\_300\\_mm.pdf](http://grandisinc.com/pdf/Feb2_2009_Grandis_300_mm.pdf), February 2009.
- [10] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In *ASPLOS '09: Proceeding of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 229–240, March 2009.
- [11] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. DRPM: Dynamic Speed Control for Power Management in Server Class Disks. In *Proceedings of the International Symposium on Computer Architecture*, pages 169–179, June 2003.
- [12] Hitachi Power and Acoustic Management - Quietly Cool. In *Hitachi Whitepaper*, March 2004. [http://www.hitachigst.com/tech/techlib.nsf/productfamilies/White\\_Papers](http://www.hitachigst.com/tech/techlib.nsf/productfamilies/White_Papers).
- [13] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda. Characterization of Storage Workload Traces from Production Windows Servers. In *IISWC '08: Proceedings of the IEEE International Symposium on Workload Characterization*, pages 119–128, September 2008.
- [14] J. K. Kim, H. G. Lee, S. Choi, and K. I. Bahng. A PRAM and NAND Flash Hybrid Architecture for High-Performance Embedded Storage Subsystems. In *EMSOFT '08: Proceedings of the 8th ACM International Conference on Embedded Software*, pages 31–40, October 2008.
- [15] M. H. Kryder and C. S. Kim. After Hard Drives—What Comes Next? *IEEE Transactions on Magnetics*, 45(10):3406–3413, October 2009.
- [16] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *Proceedings of the 36th International Symposium on Computer Architecture*, pages 2–13, 2009.
- [17] S. Lee, D. Shin, Y.-J. Kim, and J. Kim. LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems. *ACM SIGOPS Operating System Review*, 42(6):36–42, 2008.
- [18] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song. A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation. *ACM Transactions on Embedded Computing Systems*, 6(3), July 2007.
- [19] Small-Block vs. Large-Block NAND Flash Devices. Technical report, Micron, 2007.
- [20] E. L. Miller, S. A. Brandt, and D. D. E. Long. HeRMES: High-Performance Reliable MRAM-Enabled Storage. In *HOTOS '01: Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, pages 95–99, May 2001.
- [21] M. Moshayedi and P. Wilkison. Enterprise SSDs. *ACM Queue*, 6(4):32–39, 2008.
- [22] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating Server Storage to SSDs: Analysis of Tradeoffs. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 145–158, April 2009.
- [23] ONFI 2.0 specification. [http://onfi.org/wp-content/uploads/2009/02/onfi\\_2\\_0\\_gold.pdf](http://onfi.org/wp-content/uploads/2009/02/onfi_2_0_gold.pdf), 2008.
- [24] M. Polte, J. Simsa, and G. Gibson. Comparing Performance of Solid State Devices and Mechanical Disks. In *Proceedings of the 3rd Petascale Data Storage Workshop*, November 2008.
- [25] C. Preimesberger. Electricity Savings from Data Center SSDs Could Power an Entire Country, Researcher Says. *eWEEK*, May 2009. <http://www.eweek.com/c/a/Data-Storage/Researcher-Electricity-Savings-from-Data-Center-SSDs-Could-Power-an-Entire-Country-669508/>.
- [26] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *Proceedings of the 36th International Symposium on Computer Architecture*, pages 24–33, 2009.
- [27] A. Radding. *Energy-efficient storage technologies*. SearchDataCenter.com, 2007.
- [28] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung, and C. H. Lam. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development*, 52(4):465–479, 2008.
- [29] S. Sankar, S. Gurumurthi, and M. R. Stan. Intra-Disk Parallelism: An Idea Whose Time Has Come. In *Proceedings of the 35th International Symposium on Computer Architecture*, pages 303–314, June 2008.
- [30] G. Sun, Y. Joo, Y. Chen, Y. Xie, and H. Li. A Hybrid Solid-State Storage Architecture for the Performance, Energy Consumption, and Lifetime Improvement. In *HPCA '10: Proceedings of the 2010 International Symposium on High-Performance Computer Architecture*, January 2010.
- [31] F. Zheng. Leveraging HDD Strength with SSD Potential: Can Collaboration Generate Synergistic Benefits? *iSuppli Storage Space*, May 2009. <http://www.isuppli.com/Pages/Leveraging-HDD-Strength-with-SSD-Potential-Can-Collaboration-Generate-Synergistic-Benefits.aspx>.
- [32] J.-G. Zhu. Magnetoresistive Random Access Memory: The Path to Competitiveness and Scalability. *Proceedings of the IEEE*, 96(11):1786–1798, November 2008.
- [33] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes. Hibernator: Helping Disk Arrays Sleep through the Winter. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, pages 177–190, October 2005.