DOI:10.1145/1498765.1498791

Integrating NAND Flash Devices onto Servers

By David Roberts, Taeho Kgil, and Trevor Mudge

Abstract

Flash is a widely used storage device in portable mobile devices such as smart phones, digital cameras, and MP3 players. It provides high density and low power, properties that are appealing for other computing domains. In this paper, we examine its use in the server domain. Wear-out has the potential to limit the use of Flash in this domain. To seriously consider Flash in the server domain, architectural support must exist to address this lack of reliability. This paper first provides a survey of current and potential Flash usage models in a data center. We then advocate using Flash as an extended system memory usage model—OS managed disk cache—and describe the necessary architectural changes. Specifically we propose two key changes. The first improves performance and reliability by splitting Flash-based disk caches into separate read and write regions. The second improves reliability by employing a programmable Flash memory controller. It changes the error code strength (number of correctable bits) and the number of bits that a memory cell can store (cell density) in response to the demands of the application.

1. INTRODUCTION

Data centers are an integral part of today's computing platforms. As cloud computing initiatives provide IT capabilities that incorporate software as a service, it requires internet service providers such as Google and Yahoo to build large-scale data centers hosting millions of servers. Energy efficiency becomes a first-class citizen to address the increasing cost of operating a data center. Data centers based on off-the-shelf general-purpose processors are unnecessarily power hungry, require expensive cooling systems, and occupy a large space. In fact, the cost of power and cooling these data centers contributes to a significant portion of the operating cost. Figure 1 breaks down the annual operating cost for data centers. It clearly shows that the cost of power and cooling servers increasingly contributes to the overall operating costs of a data center.

System memory power (DRAM power) and disk power contribute as much as 50% to the overall power consumption in a data center. Further, current trends suggest that this percentage will continue to increase at a rapid rate as we integrate more memory modules (DRAM) and disk drives to improve throughput.

Fortunately, there are emerging memory devices in the technology pipeline that may address this concern. These devices typically display high density and consume low idle power. Flash, Phase Change RAM (PCRAM) and Magnetic RAM (MRAM) are examples.

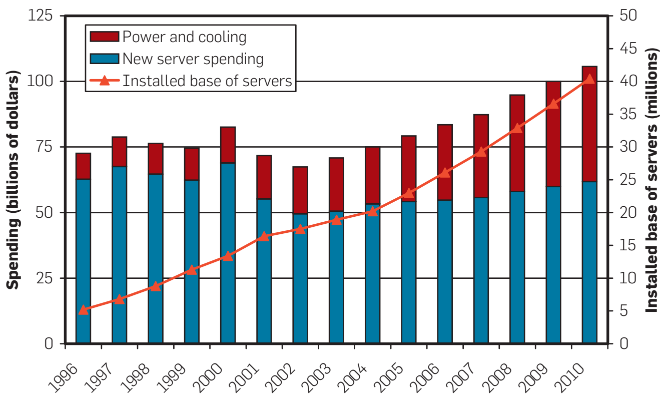
In particular, Flash is an attractive technology that is already deployed heavily in various computing platforms. Today, NAND Flash can be found in handheld devices such as smart phones, digital cameras, and MP3 players. This has been made possible because of its high density and low power properties. These result from the simple structure of Flash cells and its nonvolatility. Its popularity has meant that it is the focus of aggressive process scaling and innovation.

The rapid rate of improvement in density has become the primary driver to consider Flash in other usage models. There are several Flash usage models in the data center that are currently being examined by industry and academia that address rising power and cooling costs, among other things. Two common usage models are disk caches or storage devices. Some efforts have lead to product development,[8, 19] while others have influenced storage and memory device standards.[16, 18]

This paper provides an overview of the benefits of integrating Flash onto a server. Specifically, in this paper:

1. We provide an analysis of current and potential Flash usage models for servers.
2. We argue that the extended system memory model[10] is the best usage model to reduce data center energy when the contribution of system memory power exceeds the contribution of disk power.
3. We review two architectural modifications to improve NAND-based disk caches.[11] First, we show that by splitting Flash-based disk caches into read and write regions, overall performance and reliability can be improved.

Figure 1: IDC estimates for annual cost spent on powering and cooling servers and purchasing new servers.[17]



A previous version of this paper, entitled "Improving NAND Flash-based Disk Caches" was published in *Proceedings of the International Symposium on Computer Architecture* (ISCA 2008).

Second, we show that a programmable Flash memory controller can improve Flash cell reliability and extend memory lifetime. The first programmable parameter is error correction code (ECC) strength. The second is the Flash cell density—changing from multilevel cells (MLC) to single-level cells (SLC).

2. BACKGROUND

2.1. Properties of a NAND Flash device

Flash memory is a nonvolatile memory device that can be electrically read, written, and erased. Flash memory cells in NAND Flash are connected in series to maximize cell density. Further, to improve Flash density, each Flash memory cell can use multiple threshold voltage levels to store more than one bit per cell. NAND Flash supporting MLC is called MLC NAND Flash. NAND Flash using a single threshold voltage level (technically two levels) is called SLC NAND Flash. Cutting-edge MLC NAND Flash supports 4 bits per cell. There are significant differences in the access time and lifetime of the two types. Although MLC Flash is cheaper and bit density is higher relative to SLC, MLC is slower to read and write and has shorter lifetime by a factor of 10 or more. Typical latencies for read, write, and erase are 25 µs, 250 µs, and 0.5 ms for SLC and 50 µs, 900 µs, and 3.5 ms for 2-bit MLC. The gap between performance and lifetime is getting worse as the number of bits per Flash cell is increased. This may be perfectly acceptable for some applications; for example, a tune in an MP3 player may only be replaced every few days. A disk cache, however, may have all of its locations rewritten several times a day depending on the amount of disk traffic and size of the cache.

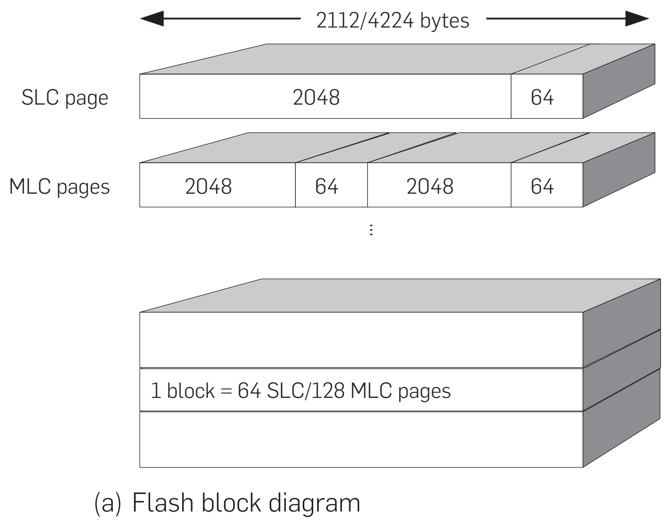
NAND Flash is organized in units of *pages* and *blocks*. A typical Flash *page* is 2KB in size and a Flash *block* is made up of 64 Flash *pages* (128KB). Random Flash reads and writes are performed on a *page* basis and Flash erasures are performed per *block*. A Flash must perform an erase on a *block* before it can write to a *page* belonging to that *block*. Each additional write must be preceded by an erase. Therefore *out-of-place* writes are commonly used to mitigate wear-out. These writes append new data to the end of the log while old data pages are invalidated.

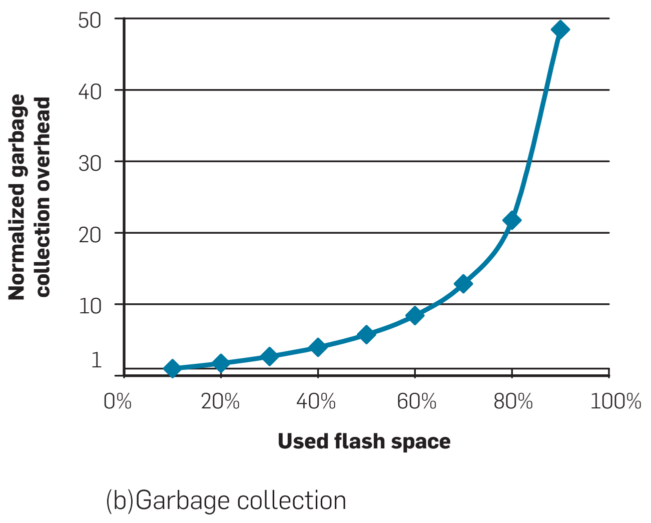
NAND Flash can also be dynamically configured to support multiple Flash memory cell types for each page or block. In fact, such devices are now commercially available, e.g., Samsung's Flex-OneNAND.[6] Figure 2(a) illustrates the organization of an SLC/MLC dual-mode device. Pages in SLC mode consist of 2048 bytes of data area and 64 bytes of "spare" data for ECC bits. When in MLC mode, an SLC page can be split into two 2048 byte MLC pages. Pages are erased together in blocks of 64 SLC pages or 128 MLC pages.

When the number of faulty bits per block exceeds the capabilities of an ECC, blocks are disabled, reducing the capacity of the memory. This is an instance of wear-out affecting system performance over time, especially, for file cache applications.

MLC Flash ages quicker than SLC Flash. An MLC Flash can support fewer reliable write/erase (W/E) cycles due to the smaller threshold voltage margins between bit values. New Flash architectures[6] can circumvent this problem by switching from high-density MLC to lower density or even single-level mode to counter wear-out. No policy currently exists to perform the mode selection, so we propose a mechanism for changing mode, tailored to a disk caching application.

Figure 2: (a) Example dual-mode SLC/MLC Flash bank organization and (b) time spent in garbage collection as a function of the Flash space in use.





Because Flash blocks have a limited number of erases before they develop faulty bits, a *wear-leveling* algorithm attempts to equalize the number of erases performed on each block.[3] This has to be achieved without performing more erases than necessary. The simplest method of wear-leveling is to treat the device as a circular log. New data is written to the next available page and the old page is invalidated. However, wear-leveling causes fragmentation problems. Fragmentation is addressed with garbage collection. The process of garbage collection reads valid pages from erase blocks containing some invalid pages, then writes them to a previously erased block.[4] Garbage collections free up pages that are ready to write new data. This process takes time and increases the amount of wear in the Flash blocks.

The overhead in garbage collection increases as less free space is available on Flash. This becomes a significant problem, because garbage collection generates extra writes and erases in Flash, reducing performance and endurance as the occupancy of the Flash increases. Figure 2(b) shows how the time spent garbage collecting increases as more Flash space is used. It is normalized to an overhead of 10% and is for a 2GB Flash memory. It can be seen that garbage collection becomes overwhelming well before all of the memory is used.

2.2. NAND Flash usage models in a server

Industry and researchers in academia are making strides to integrate Flash onto the data center. Industry has recently released several Flash-based products and Flash standards targeted for servers, while researchers in academia have recently published several papers proposing techniques for integrating emerging memory technologies including Flash.

NAND Flash usage models pursued by industry and academia can be categorized as follows:

1. Extended system memory usage model: A NAND Flash memory module is connected to the current system memory interface or to a dedicated Flash memory interface.
2. Storage accelerator usage model: A NAND Flash PCI express card is connected to the PCI express interface.
3. Alternative storage device usage model: A Solid State Drive (SSD) replaces or augments the hard disk drive. It is connected to the disk interface. An example would be a SATA SSD.

Each usage model presents a unique set of benefits and challenges. Table 1 qualitatively captures them. The "extended system memory" usage model presents Flash as a part of the system memory. It addresses the rising contribution of power consumed by DRAM in addition to the electrical constraints limiting the integration of more system memory. For example, to increase storage capacity without having to reduce the operating frequency of the memory channel, MetaRAM[14] packs more DRAM onto each DIMM module. Using denser memory such as Flash may serve a similar purpose. However, this usage model requires modification to the operating system kernel. Specifically, the current implementation in the kernel memory manager that supports nonuniform memory architectures needs to be aware of the unique organization and behavior of Flash. Flash reliability management can be performed by the kernel memory manager with the assistance of the Flash controller.

The "storage accelerator" usage model presents Flash as a PCI express device that can be directly managed by the user application. This usage model allows the server application to manage Flash directly as a cache that stores frequently accessed code and data. It reduces the number of accesses to the hard disk drive thereby reducing overall disk power. Further, it may also be used as a way to implement the "extended system memory" usage model but with several drawbacks such as higher latency, lower throughput and added complexity in managing Flash. Flash management is distributed across the user application, device driver stack and the Flash PCI express card firmware. To truly leverage Flash as a "storage accelerator," the user application should be Flash aware. A device driver stack needs to be implemented to support the PCI express device. The device driver stack needs to implement device sharing mechanisms such that other concurrent user applications and kernel components can make use of it simultaneously. In Fusion-io's Solid State Storage[7] they have also shown the "storage accelerator" usage model can expose the Flash PCI express device as an SSD by providing disk emulation features in the device driver stack.

The "alternative storage device" usage model presents Flash as an SSD that replaces a hard disk drive.[15] This usage model improves the latency and throughput to disk and reduces overall disk power consumption in a data center. With appropriate filesystems such as ZFS,[19] it improves storage device scalability in a data center. Industry has heavily adopted this usage model and has recently released several products.[8, 9] These solid state drives are used to implement a storage area network (SAN) or network attached storage (NAS) in a data center. They employ similar reliability features such as RAID, commonly found in a hard disk drive based SAN or NAS. Flash reliability management is performed by the Flash device controller in the SAN or NAS. However, this usage model also requires modification in the kernel, and a complex Flash device controller that is capable of performing intelligent Flash reliability management. A customized filesystem needs to be implemented to fully take advantage of the benefits of Flash.[12, 19] Further, this usage model ties itself to the non-Flash aware features that are found in a hard disk drive interface protocol such as SATA. For example, the device driver can only communicate to disk using SATA commands that are not defined with Flash in mind. On the other hand, the operating system in the "extended main memory" model has full visibility of memory page classification and activity statistics that can be used for more intelligent mapping of data to Flash.

Table 1: Comparison of Flash usage models in a server.

	Primary powersavings	Secondary powersavings	Hardware complexity	OS kernel modification	Application modification	Comments
Extended system memory	DRAM	Disk	Minimal	Medium	None	Extend kernel memory manager to manage Flash devices
I/O accelerator	Disk	DRAM	Medium	Medium	Yes	Need to build I/O accelerator driver stack
Alternative storage device	Disk	DRAM	High	Minimal	None	Need to implement filesystem for Flash

Servers clearly benefit from all three usage models that essentially integrate Flash as a faster hard disk or disk cache. All usage models help (1) reduce unnecessary standby power from hard disk drives and (2) improve overall throughput by reading and writing from disk cache instead of a hard disk drive.

In the remainder of this paper, we examine Flash-based disk cache architectures that improve Flash manageability and reliability in the extended system memory usage model. We believe this usage model is effective in addressing the increasing power consumption in system memory. Our studies on servers have revealed the system memory architecture to be the critical component in delivering high throughput in a data center.[10]

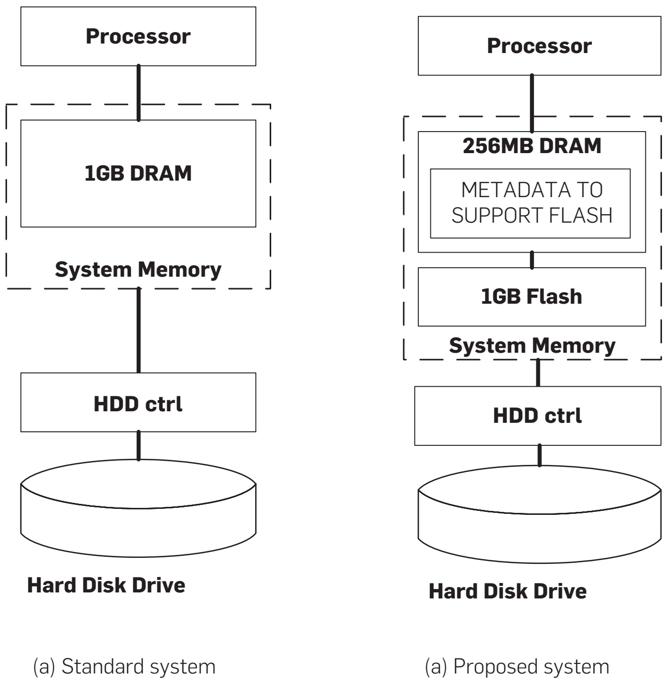
3. PROPOSED ARCHITECTURE

3.1. Architecture of the Flash-based disk cache

The right side of Figure 3 shows the Flash-based disk cache architecture for the extended system memory usage model. Compared to a conventional DRAM-only architecture shown on the left side of Figure 3, our proposed architecture uses a two level disk cache, composed of a relatively small DRAM in front of a dense Flash. The much lower access time of DRAM allows it to act as a cache for the Flash without significantly increasing power consumption. A Flash memory controller is also required for reliability management.

Our design uses a NAND Flash that stores 2 bits per cell (MLC) and is capable of switching from MLC to SLC mode using techniques proposed in Flex-OneNAND[6] and Cho.[5] Finally, our design uses variable-strength ECC to improve reliability while adding the smallest possible delay.

Figure 3: 1GB DRAM is replaced with a smaller 256 MB DRAM and 1GB NAND-based Flash. Additional components are added to control Flash.



Operating System Support: Our proposed architecture requires additional data structures to manage the Flash blocks and pages. These tables are read from the hard disk drive and stored in DRAM at run-time to reduce access latency and mitigate wear-out. Together, they describe whether pages exist in DRAM or Flash, and specify the various Flash memory configuration options for reliability. For example, the FlashCache Hash Table allows the operating system to quickly look up the location of a file page. The Flash Page Status Table keeps track of the ECC strength, MLC/SLC mode and access frequency for each page. Each Erase block has an entry in the Block Status Table to determine how worn out it is. Finally, the Global Status Table records how quickly the Flash-based disk cache is satisfying requests, and is the number we try to maximize while the system is running.

The storage overhead of the four tables are less than 2% of the Flash size. The FlashCache Hash Table and Flash Page Status Table are the primary contributors because an entry is needed for each Flash page. Our Flash-based disk cache is managed in software (OS code) using the tables described above. We found the performance overhead in executing this code to be minimal.

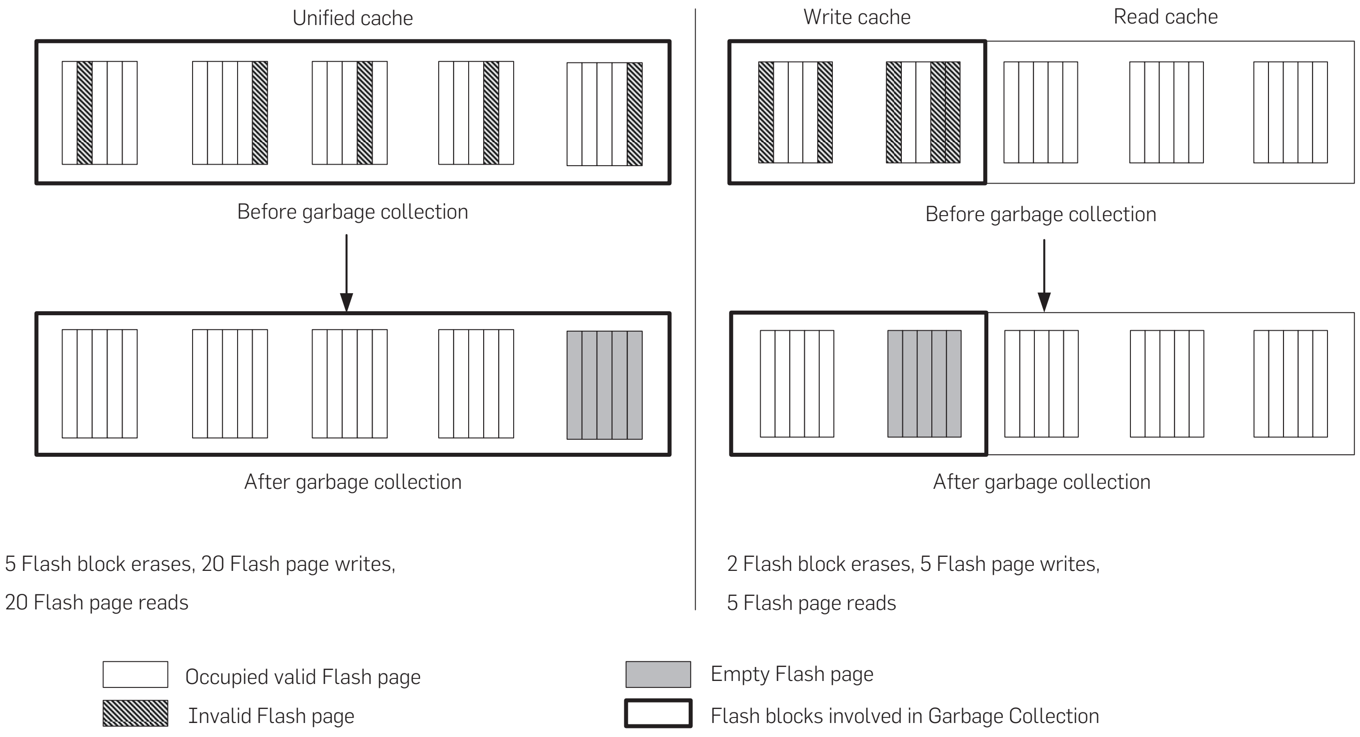
Splitting Flash into Read and Write Regions: We divide the Flash into a read disk cache and a write disk cache. Read caches are less susceptible to *out-of-place* writes, which reduce the read cache capacity and increase the risk of garbage collection. An *out-of-place* write happens when existing data is modified, because Flash has to be erased before it can be written to a second time. It is simple to invalidate the old data page (using the Page Status Table and modifying the Hash Table) then write new data into a previously erased page. However, the invalid pages accumulate as wasted space that will have to be garbage collected later. By splitting Flash into read and write regions, we were able cut down on time consuming garbage collections.

Figure 4 shows an example that highlights the benefits of splitting the Flash-based disk cache into a read and write cache. The left side shows the behavior of a unified Flash-based disk cache and the right side shows the behavior of splitting the Flash-based disk cache into a read and write cache. Figure 4 assumes we have five pages per block and five total blocks in a Flash-based disk cache. Garbage collection proceeds by reading all valid data from blocks containing invalid pages, erasing those blocks and then sequentially re-writing the valid data. In this example, when the Flash-based disk cache is split into a read and write cache, only two blocks are candidates for garbage collection. This dramatically reduces Flash reads, writes, and erases compared to a unified Flash-based disk cache that considers all five Flash blocks. Our studies also show that the overall disk cache miss rate is reduced substantially for online transaction processing (OLTP) applications by splitting the Flash.

3.2. Architecture of the Flash memory controller

Flash needs architectural support to improve reliability and lifetime when used as a cache. Figure 6 shows a high-level block diagram of a programmable Flash memory controller that addresses this need. Requests from the operating system

Figure 4: Example showing the benefits of splitting the Flash-based disk cache into Read and Write caches. In the five erase blocks, pages of data that have been evicted from the cache and invalidated are grayed out. On the left, a unified cache allows pages that are heavily read and written to be placed in any erase block. This results in scattered invalid pages. Our split read/write cache forces read and write-dominated data into two separate sets of erase blocks. As a result, invalid pages are clustered and fewer blocks have to be erased to prepare the invalid pages for another write.


Unified cache
Write cache
Read cache
Before garbage collection
Before garbage collection
After garbage collection
After garbage collection
5 Flash block erases, 20 Flash page writes, 20 Flash page reads
2 Flash block erases, 5 Flash page writes, 5 Flash page reads
Occupied valid Flash page
Empty Flash page
Invalid Flash page
Flash blocks involved in Garbage Collection

provide the address being accessed and any data to be written. In addition, the OS specifies the strength of ECC and whether the page is in MLC or SLC mode. The controller returns any data that was read along with information concerning the number of errors currently being corrected by the ECC logic.

Our architecture uses a BCH encoder and decoder to perform error correction and a CRC checker to perform error detection. The BCH code guarantees that a number of faulty bits can be corrected. However, as the number of faulty bits increases it takes longer to perform the correction. Doubling the number of correctable bits approximately doubles the time needed to decode the data and extract the correct value. Our system adapts the ECC strength to the appropriate number of faulty bits in each page to achieve graceful Flash wear-out. The relationship between number of correctable bits and erase count is shown in Figure 5. It shows that stronger ECC effectively improves page lifetime. The different lines on the graph show the effects of different levels of variability in the likelihood of bits being faulty. As the standard deviation increases, the number of tolerated erases decreases for any particular error correction strength. It shows that process variability has a negative impact on lifetime and requires more bits to be corrected for the same lifetime. As process technology advances and cells become smaller, the effect of variability will become even more pronounced.

Our programmable Flash memory controller also dynamically controls the density of a Flash page. Density control benefits Flash performance and endurance, because we are able to reduce access latency for frequently accessed pages and possibly improve endurance for aging Flash pages by

Figure 5: Maximum tolerable Flash Write/Erase cycles for varying code strength.

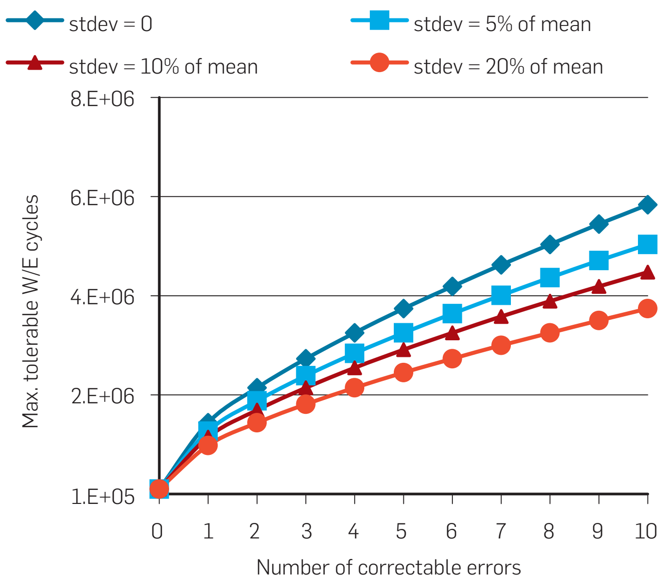
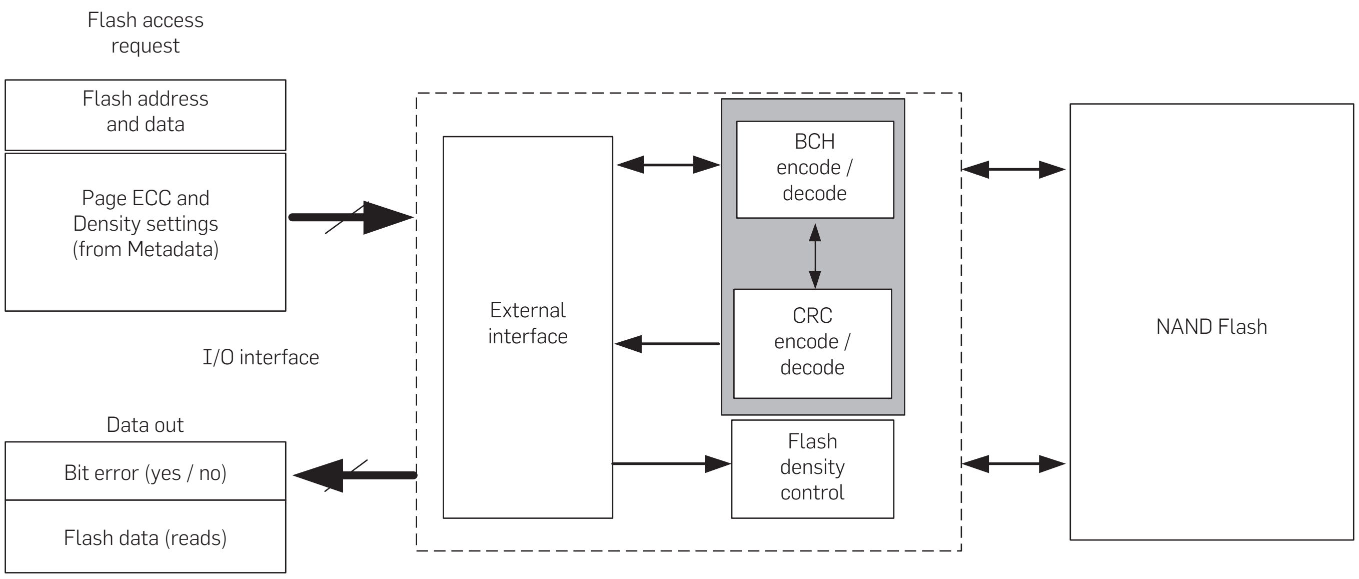


Figure 6: Flash memory controller architecture. The Flash disk cache device driver sends requests to the hardware interface. These requests also specify the ECC strength and density mode of the accessed page. In turn, the controller accesses the Flash chip after performing ECC encoding for a write, or decoding for a read. The device driver software receives any requested data along with an indication of the number of failing Flash bits.



changing MLC pages into SLC pages as needed. To show the potential improvement of Flash performance by controlling density, we present a study using real disk traces.

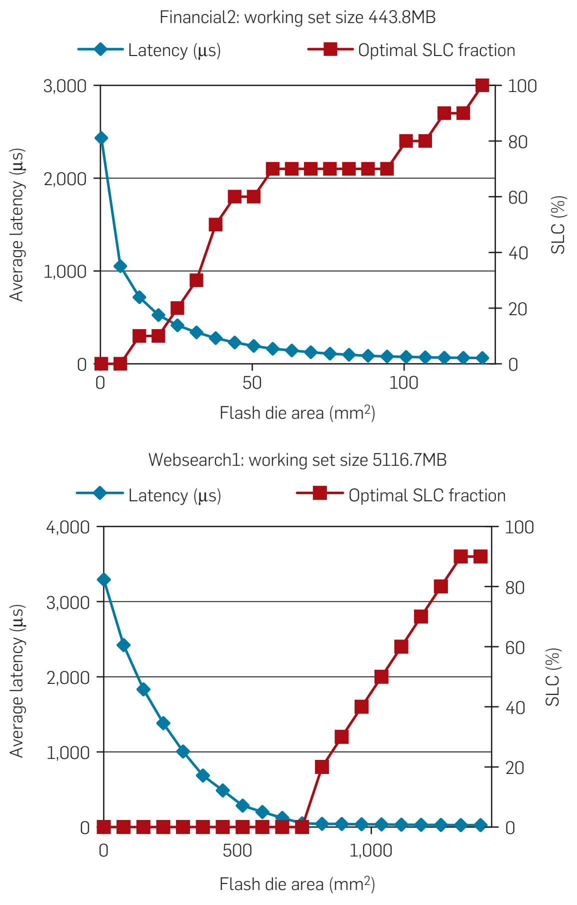
Using disk activity traces from the University of Massachusetts Trace Repository[20] for financial and web search applications, we analyzed the average access latency for different SLC/MLC partitions, for several Flash sizes.

A hybrid allocation of SLC and MLC Flash provides minimum access latency, because it is sometimes more effective to store heavily used data in a faster SLC page and lose one page of storage space. Figure 7 shows the average delay (left y-axis) achieved for an optimal partition (right y-axis) between SLC and MLC. The x-axis shows the Flash memory area and extends far enough to contain the entire working set. As expected, when the size of the cache approaches the entire workload, latency reaches a minimum using only SLC cells. Intermediate Flash sizes provide minimum latency through a combination of MLC and SLC with the most frequently accessed data in the faster SLC cells. The best division between SLC and MLC depends on the access frequencies of the data pages. For example, if there are hot pages that are significantly more active than other pages, the bias will be towards SLC. This type of behavior is exhibited by the Financial2 trace (Figure 7(a)) where the SLC allocation grows rapidly with Flash capacity. If the access frequencies are more uniform (e.g., Websearch1 in Figure 7(b)) it is better to have a bigger (MLC) cache to increase the number of accesses to Flash because going to disk is much slower.

3.3. Operation and dynamic reconfiguration

We use a typical software device driver interface to access the Flash memory controller. The driver specifies which Flash address is to be accessed along with the read or write mode. For a write, the driver also sends the new data to the

Figure 7: Optimal access latency and SLC/MLC partition for various multimode MLC Flash sizes.



controller. Using the configuration tables stored in DRAM, the driver tells the Flash controller what error correction strength and cell density to use for each access. In this section we provide more details on how the cache operates and how its settings are reconfigured on the fly.

Theory of Operation: In this section we summarize how the operating system and controller interact (see Kgil[11] for a full description). The concepts are similar to ordinary disk caches except that it is now a two-level cache. The first level of cache resides in DRAM, and the second level consists of Flash memory. In addition, the Flash portion of the cache has to be reconfigured on the fly to maximize performance and reliability. The DRAM, with fast, uniform read and write latency, no wear-out and no density modes, is easier to handle.

When a file read is performed, the OS searches for the file in the primary disk cache located in DRAM. If the page is found in DRAM, the file content is accessed directly from the primary disk cache—no access to Flash related data structures is required. Otherwise, the OS determines whether the requested file currently resides in the secondary (Flash) disk cache. If the requested file is found, then a Flash read is performed and the Flash content is transferred to DRAM.

If the data is not found in Flash, we first look for an empty Flash page in the read cache. If there is no empty Flash page available, we first select a block for eviction to disk, freeing Flash pages for the newly read data. The data being replaced is usually the "least recently used" (LRU) block so it is unlikely to be needed again. Such an access would have to go all the way to disk, increasing program execution time, so the LRU algorithm reduces the likelihood of this happening. Concurrently, a hard disk drive access is scheduled using the device driver interface. The hard disk drive content is copied to the primary disk cache in DRAM and also the read cache in Flash.

If we write to a file, we typically update/access the page in the primary disk cache and this page is periodically scheduled to be written back to the secondary disk cache and later periodically written back to the disk drive. When writing back to Flash, we first determine whether it already exists on Flash. If it is found in the write region, we update the page by doing an *out-of-place* write to the write cache. If it is found in the read cache, then we move it to the write cache. If it is not found in the Flash, we allocate a page in the write cache.

In the background, garbage collections are triggered when the Flash-based disk cache starts to run out of space. The cached data is also periodically flushed back to disk if it has been modified. Concurrent with the normal cache operation, the reliability management algorithms continuously try to adapt the Flash configuration to provide maximum benefit. We have already seen that the configuration changes with the application software. The next section describes the configuration policies enforced to achieve this.

Reconfiguring the Flash Memory Controller: The Flash Page Status Table (FPST) specifies the reliability control settings for each page of flash. When the OS reads and writes to/from the Flash controller, it also sends configuration bits specifying the various modes for the Flash page. Configuration policies are applied to select those modes, maximizing performance as the application demands change and the Flash eventually develops faulty bits.

There are two main triggers for an ECC strength or density mode change. These are (1) an increase in the number of faulty bits and (2) a change in access (read) frequency. Each trigger is explained below:

When new bit errors are observed and fail consistently due to wear-out, we reconfigure the page. This is achieved by enforcing a stronger ECC or reducing cell density from MLC to SLC mode. We choose the option with the minimum increase in latency using some simple heuristics. They take into account how active that particular page is to determine its impact on the system as a whole. It also considers the current level of wear-out for the page.

Some heavily accessed pages will benefit from being in SLC storage simply because of its lower latency. If a page is in MLC mode and the entry in the FPST field that keeps track of the number of read accesses to a page reaches a limit, we migrate that Flash page to a new empty page in SLC mode. If there is no empty page available, a Flash block is evicted and erased using our wear-level aware replacement policy. Reassigning a frequently accessed page from MLC mode to SLC mode improves performance by improving hit latency. Because many accesses to files in a server platform have a tailed distribution (Zipf) with hot and cold data, improving the hit latency to frequently accessed (hot) Flash pages improves overall performance despite the minor reduction in Flash capacity.

If a Flash page reaches the ECC strength limit and has already been set to SLC mode, the block is removed permanently and never considered when looking for pages to allocate in a disk cache.

4. METHODOLOGY

We evaluated the Flash memory controller and Flash device using a full system simulator called M5.[2] The M5 simulation infrastructure is used to generate access profiles for estimating system memory and disk drive power consumption along with published access energy data. We developed a separate Flash disk cache simulator for reliability and disk cache miss rate experiments where very long traces are necessary, because full system simulators are slow. Given the limitations in our simulation infrastructure, a server workload that uses a large working set of 100–1000's of gigabytes cannot easily be evaluated. We scaled our benchmarks, system memory size, Flash size, and disk drive size accordingly to run on our simulation infrastructure.

We also generated micro-benchmark disk traces to model synthetic disk access behavior. They represent typical access distributions and approximate real disk usage. To properly stress the system, some micro-benchmarks with uniformly random and exponential distributions were also generated.

We used disk traces from University of Massachusetts Trace Repository[20] to model the disk behavior of enterprise level applications like web servers, database servers, and web search. To measure performance and power, we used dbt2 (OLTP) and SPECWeb99 which generated representative disk/disk cache traffic.

5. RESULTS

5.1. System memory and disk energy efficiency

Figure 8 shows a breakdown of power consumption in the system memory and disk drive (left *y*-axis). Figure 8 also shows the measured network bandwidth (right *y*-axis). Throughput measured as network bandwidth is a good indicator of overall system performance as it represents the amount of data that the server can handle in each configuration. We calculated power for a DRAM-only system memory and a heterogenous (DRAM + Flash) system memory that uses a Flash as a secondary disk cache with hard disk drive support. We assume equal die area for a DRAM-only system memory and a DRAM + Flash system memory. Figure 8 shows the reduction in disk drive power and system memory power that results from adopting Flash. Our primary power savings for system memory come from using Flash instead of DRAM for a large amount of the disk cache. The power savings for disk come from reducing the accesses to disk due to a bigger overall disk cache made possible by adopting a Flash. We also see improved throughput with Flash because it displays lower access latency than disk.

5.2. Impact of BCH code strength on system performance

We have already mentioned that BCH latency incurs an additional delay beyond the initial access latency. We simulated the performance of the SPECWeb99 and dbt2 benchmarks to observe the effect of increasing code strength that would occur as Flash wears out. It is assumed that all Flash blocks have the same ECC strength applied. We also measured performance for code strengths (more than 12 bits per page) that are beyond our Flash memory controller's capabilities to fully capture the performance trends.

From Figure 9, we can see that throughput degrades slowly with ECC strength. dbt2 suffers a greater performance loss than SPECWeb99 after 15 bits per page. The disk bound property of dbt2 makes it more sensitive to ECC strength.

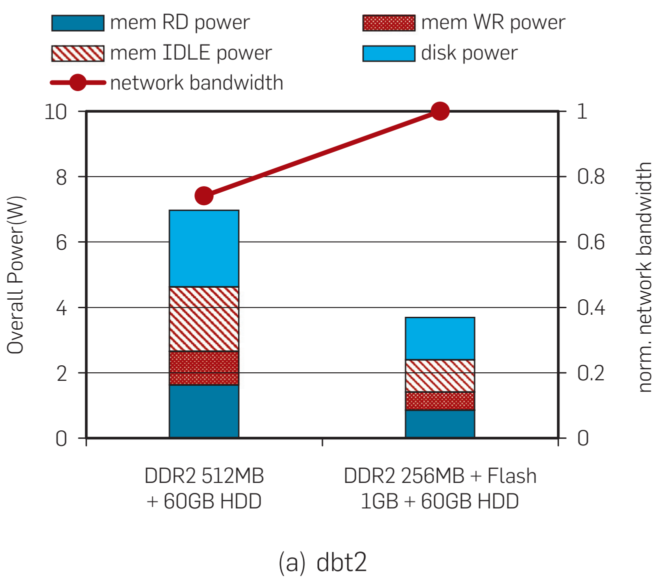
5.3. Improved Flash lifetime with reliability support in Flash memory controller

Figure 10 shows a comparison of the normalized number of accesses required to reach the point of total Flash failure where none of the Flash pages can be recovered. We compare our programmable Flash memory controller with a BCH 1-bit error correcting controller. Our studies show that for typical workloads, our programmable Flash memory controller extends lifetime by a factor of 20 on average. For a workload that would previously limit Flash lifetime to 6 months, we show it can now operate for more than 10 years using our programmable Flash memory controller. This was accompanied by a graceful increase in overall access latency as Flash wore out.

6. CONCLUSIONS AND FUTURE WORK

This paper presents the challenges and opportunities in integrating Flash onto a server platform. Flash is an attractive candidate for integration because it reduces power consumption in system memories and disk drives while improving overall throughput. This in turn can reduce the operating cost of a server platform, which is a growing concern in a data center. We presented three key usage models of Flash and examined an architecture for the "extended system memory" usage model. Our proposed architecture carefully manages the Flash and uses it as a secondary disk cache split into a separate read cache and write cache. We observed a dramatic improvement in power consumption and performance. In our simulation studies, a Flash-based disk cache improved the DBT2 database benchmark performance by over 25% while reducing memory and disk power by 44%. For a web server benchmark, performance improvement was around 11% with a power reduction of 73%. This does not account for potentially larger systemwide energy savings obtained from speeding up system response and increasing idle time. Assuming that a server can enter a low-power mode while

Figure 8: Breakdown in system memory and disk power and network bandwidth for architecture with/without a Flash-based disk cache.



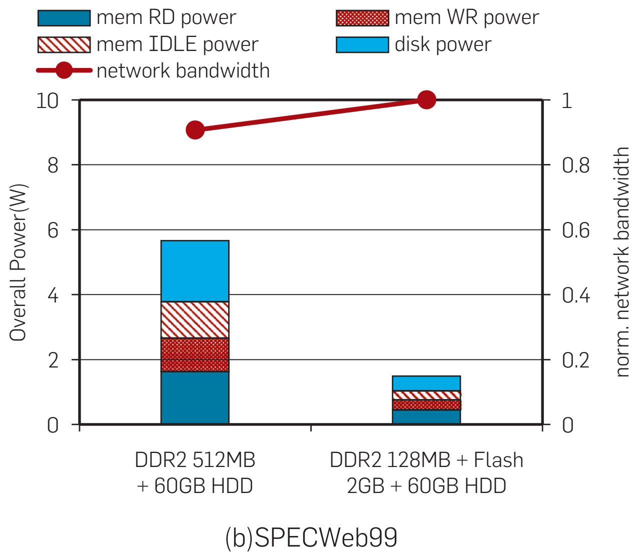
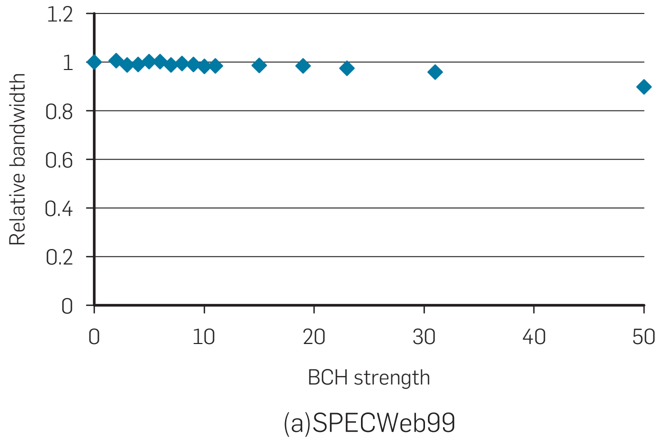


Figure 9: Average throughput as a function of ECC strength. The system used 256MB of DRAM and 1GB of Flash.



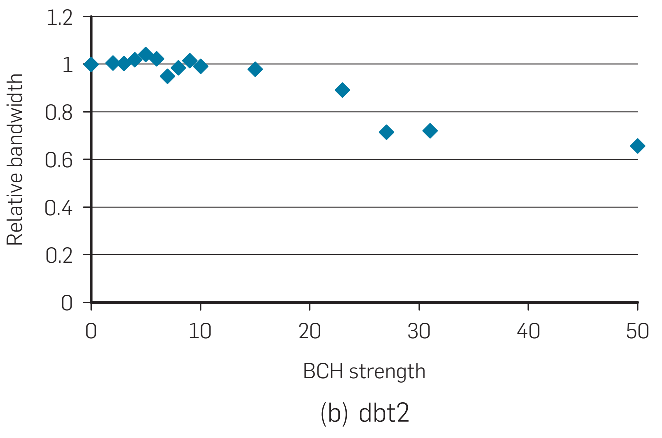
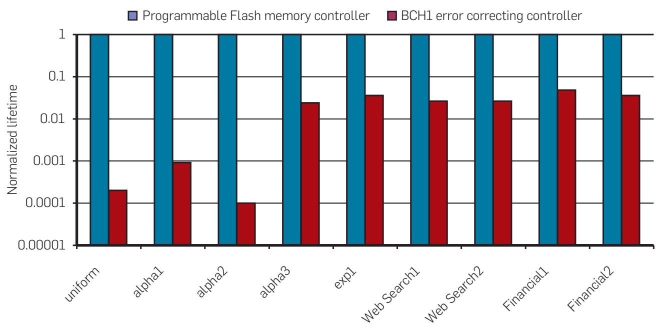


Figure 10: Normalized expected lifetime for a given access rate and the point of total Flash failure.



idle,[13] all components save energy rather than just memory and disk.

We also showed that a Flash memory controller with reliability support greatly improves Flash lifetime. We found that the best configuration of a Flash memory controller is largely dependent upon the access patterns resulting from the application. For example, we found that the typical workload with Zipf access behavior was best served by a Flash configured such that the heavily accessed contents would be located in regions composed of reliable low latency SLC. In general, we found that variable ECC strength gracefully extended Flash lifetime, and that the overhead of ECC is minimized with configurable density. Combining all of our techniques, we saw an average 20× lifetime improvement relative to a system using only a single ECC. We believe our findings are applicable not only to Flash but also to emerging memory technology devices such as PCRAM.[1]

New memory technologies are creating opportunities for increased performance and efficiency in a data center. These disruptive technologies are forcing architects to rethink the current system memory and storage hierarchy in a server. Together, with server virtualization, we believe these memory devices will help realize the objective of building greener data centers. C

References

1. Bedeschi, F. et al. A multi-level-cell bipolar-selected phase-change memory. In *Proceedings of the International Solid-State Circuits Conference* (Feb. 2008), 428–625.
2. Binkert, N., Dreslinski, R., Hsu, L., Lim, K., Saidi, A., Reinhardt, S. The M5 simulator: Modeling networked systems. *IEEE Micro 26*, 4 (Jul./Aug. 2006), 52–60.
3. Chang, L.-P. On efficient wear-leveling for large-scale flash-memory storage systems. In *22nd ACM Symposium on Applied Computing (ACM SAC)* (2007).
4. Chang, L.-P., Kuo, T.-W. Real-time garbage collection for flash-memory storage system in embedded systems. *ACM Trans. Embedded Computing Systems 3*, 4 (2004).
5. Cho, T. et al. A dual-mode NAND flash memory: 1-Gb multilevel and high-performance 512-mb single-level modes. *IEEE J. Solid State Circuits 36*, 11 (Nov. 2001).
6. Flex-OneNAND. http://www.samsung.com/global/business/semiconductor/products/fusionmemory/Products_FlexOneNAND.html.
7. Fusion-io's Solid State Storage—A New Standard for Enterprise-Class Reliability. http://www.fusionio.com/PDFs/Whitepaper_Solidstatestorage2.pdf.
8. Hutsell, W, Bowen, J., Ekker, N. Flash Solid-State Disk Reliability. http://www.texmemsys.com/files/f000252.pdf.
9. Intel X18-M/X25-M SATA Solid State Drive. http://download.intel.com/design/flash/nand/mainstream/mainstream-sata-ssd-datasheet.pdf.
10. Kgil, T., Mudge, T. Flashcache: A NAND Flash memory file cache for low power web servers. In *International Conference on Compilers, Architecture and Synthesis for Embedded Systems* (2006).
11. Kgil, T., Roberts, D., Mudge, T. Improving NAND Flash based Disk Caches. In *Proceedings of the International Symposium on Computer Architecture (ISCA)* (2008).
12. Leventhal, A. Flash storage today. *ACM Queue* (Aug. 2008).
13. Meisner, D., Gold, B.T., Wenisch, T.F. Powernap: Eliminating server idle power. *ASPLOS* (Mar. 2009).
14. MetaRAMs DDR3 MetaSDRAM Doubles Memory Capacity and Increases Frequency of Future Intel Systems. http://www.metaram.com/pdf/press/MetaRAM_DDR3_08_19_08.pdf.
15. Moshayedi, M., Wilkison, P. Enterprise ssds. *ACM Queue* (Aug. 2008).
16. ONFI: Open NAND Flash Interface. http://www.onfi.org/index.html.
17. Scaramella, J. Enabling Technologies for Power and Cooling. http://h71028.www7.hp.com/enterprise/downloads/Thermal_Logic.pdf.
18. Serial ATA 2.6 Specification. http ://www.sata-io.org.
19. *Solaris ZFS Administration Guide.* 2008.
20. University of Massachusetts Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage.

David Roberts (daverobe@umich.edu), Advanced Computer Architecture Lab, Department of CSE, University of Michigan.

Trevor Mudge (tnm@eecs.umich.edu), Advanced Computer Architecture Lab, Department of CSE, University of Michigan.

Taeho Kgil (taeho.kgil@intel.com), Intel Corporation.