Extending Flash Lifetime in Secondary Storage

Chengjun Wang, and Sanjeev Baskiyar

Abstract—Unlike magnetic disks, NAND flashes can be written a limited number of times. As flash memory densities increase and cell sizes shrink, further decreases in write endurance is expected. Although some mitigation is achieved by wear leveling, write endurance remains a concern for write intensive applications. In this research, we use a DRAM cache to filter write traffic to flash by coalescing and merging overwrites. To handle integrity of data upon power failure, we use a supercapacitor to provide short duration backup power during which DRAM data can be retired to flash memory. The effectiveness of such a mechanism is not obvious considering that a large file-system cache already exists which also merges overwrites. We investigated: (i) a DRAM and a flash disk cache combo within a magnetic disk controller and (ii) a DRAM only cache when flash is a full secondary storage. Our simulations show that using a medium sized DRAM cache, flash lifetime doubles with lazy updates compared to early update policy. Moreover, miss ratio and average response times decrease as well. With little effort, our technique can be extended to improve the usable life of other emerging non-volatile memories, such as PCM and MRAM.

Index Terms—Disk cache, flash lifetime, secondary storage.

1 Introduction

The performance gap between the processor and storage of computers is widening with approximately 60% and 10% annual improvement in the processor and hard disk drives (HDDs) respectively [1]. The trend is becoming more marked with the advent of multi-socket, multi-core processor architectures. The I/O performance, especially I/O operations per second (IOPS), has not caught up with the corresponding improvements in processor performance. The processor utilization stays low due to the need to wait for the data being fed from the storage system [2]. Therefore, storage performance is becoming the bottleneck of a computer system.

Fortunately, there are emerging memory technologies that try to bridge the growing performance gap, such as flash memory, Phase Change RAM (PCM), and Magnetic RAM (MRAM). Noticeable among them is flash memory, which has been the most widely used nonvolatile memory. There are two types of flash: NOR flash and NAND Flash. NOR flash is byte addressable while NAND flash is page addressable. In this paper, we are only concerned about NAND flash, which is referred to as flash for short hereafter. Not only has flash been widely used on portable devices as storage media, but also flash-based Solid State Drives (SSDs) are being installed into data centers.

SSDs can provide as much as 3,300 write IOPS and 35,000 read IOPS, consuming 2.5 watts of power, whereas even the best HDDs (15K RPM drives) can only offer 300-400 IOPS while consuming 15-20 watts of power[2]. In comparison, the processor can offer 1,000,000 IOPS. Performance in term of IOPS is critical for enterprise applications serving a large number of users, like web servers, email servers, cloud computing, and cloud storage. The common method used to close the gap is to deploy multiple HDDs working in parallel to support peak workloads. In order to meet the IOPS requirement, more HDDs are added, which results in environments that have underutilized storage (well below 50% of their useful storage capacity). The extra storage in turn will incur power and cooling waste.

However, flash can only be written a limited number of times, ranging from 10K to 100K depending on the type of flash used. Traditional solutions to limited lifetime of flash focused on the algorithms used within Flash Translation Layer (FTL), which spread the writes evenly across the medium. It is referred to as wear leveling, in which no single cell fails ahead of others. Although wear leveling mitigates the lifetime issue to some extent, it remains a concern for write intensive applications.

In this paper, we focus on:

- Using DRAM as a cache for flash memories rather than HDD unlike previous works. Since DRAM does not have read penalty unlike HDD, the performance needs investigation. Read penalty impacts response time and throughput. By introducing HDDs into SSDs, the device as a whole will lose its advantages in terms of acoustic levels, mechanical reliability, susceptibility to environmental factors, weight and size, and power consumption.
- Determining whether a small DRAM (15-60MB) can effectively reduce write traffic. Although earlier works [8] have used relatively larger size (91-2,728MB) HDD as a cache for SSDs, the effectiveness in using smaller size caches have not been investigated. Using a small size cache is important. Upon power failure, DRAM contents need to be retired

- *C. Wang is with the Department of Computer Information Systems, Vermont Technical College, Randolph Center, VT, 05061.*
  E-mail: cwang@vtc.edu
- *S. Baskiyar is with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849.*
  E-mail: baskisa@auburn.edu

to permanent store using a supercapacitor, whose size will be small for retiring the contents of a small DRAM. Thus, the super-capacitor can be practically implemented in a reasonable space.
- Comparing the retirement policy for entries of DRAM, i.e. early vs. lazy retirement, in terms of traffic, response time, miss ratios and DRAM sizes.
- Determining the effectiveness of employing DRAM in increasing lifetime of the flash cache used in HDD controllers. We see that with a medium sized DRAM cache, flash lifetime doubles with lazy updates compared to early updates. Moreover, miss ratio and average response times decrease as well.

The rest of the paper is organized as follows: Section 2 presents the the motivation. Section 3 briefly reviews related work. In Section 4, we talk about the system architecture. Section 5 discusses the methodology we used. In Section, 6, we show the simulation results when flash acts as disk cache. In Section 7, we present the simulation results when flash is used as major store. Finally, in Section 8, we conclude the paper.

2 Motivation

2.1 Flash Memory

As mentioned above, one of flash drawbacks is write endurance. The endurance issue stems from cell degradation caused by each burst of high voltage (10V) across the cell. The problem shows no imminent sign of vanishing. As flash goes into Multi-Level Cell (MLC), write endurance becomes worse compared with Single Level Cell (SLC). For example, the write cycles of 2X MLCs drop to 10,000 from 100,000 (SLC). Furthermore, write endurance becomes worse as cells become smaller.

In order to mitigate the flash drawbacks, a Flash Translation Layer (FTL) has been proposed to manage how the flash resources are used [3]. FTL keeps mapping tables between logical and physical address spaces. When an update to a file is issued, FTL writes the file to a blank page and marks the old page as invalid. FTL updates the mapping tables accordingly. A technique called wear leveling attempts to evenly use entire memory cells.

In spite of all these efforts, endurance is still a concern for flash based storage systems. The flash lifetime over I/Os per second is shown in Fig. 1 [4]. The lifetime drops as the number of I/Os increases. As Kim et al. [4] put it, “Although MTTFs for HDDs tend to be of the order of several decades, recent analysis has established that other factors (such as replacement with next, faster generation) imply a much shorter actual lifetime and hence we assume a nominal lifetime of 5 years in the enterprise.” As we can see from the figure, when the number of I/Os exceeds approximately 50 IOPS, the lifetime of flash is less than 5 years. For example, the OpenMail workloads we used have 98 I/Os per second, which corresponds to a lifetime less than 2 years. Therefore, the endurance issue is one of the factors that would hinder the further applications of flash-based storage systems to a heavy write-intensive workload environment like some embedded or enterprise systems, where the storage system needs to work 24/7 with heavy write traffic. Therefore, enhancing the flash endurance is demanded.

2.2 Flash Trends

According to [5], flash trends can be expressed as: bigger, faster, and cheaper, but not better. Bigger: Flash component densities are doubling at a rate greater than Moore’s Law. Faster: A single operation is needed for programming an entire page or erasing an entire block. Cheaper: As the densities are going up, price is going down. Not better: “Better” means more reliable in terms of endurance and data retention. Data retention is defined as the length of time a charge remains on the floating gate after the last program. The cells are worn out over program/erasure and make it more difficult to keep the electrons in place. Therefore, there is an inverse relationship between endurance and data retention — the more program/erasure, the shorter the data retention. As NAND manufactures are struggling for lower cost per bit, they keep sacrificing endurance and data retention. The most renowned trade-off is in MLC vs. SLC, in which 2:1 or 3:1 cost benefit is obtained through 10:1 reduction in rated endurance.

According to Hutchby et al. [6], Flash has been classified as “mature nonvolatile memory”. Most importantly, flash will not disappear in the short run [7]. Although there are emerging memories, such as PCM (PCRAM), and MRAM, they have not reached the maturity level to replace flash. Flash will coexist with the emerging memories for 5-10 years. Therefore, the lifetime issue of flash deserves the efforts of research.

2.3 Limitations of Existing Solutions

There are two basic methods to solve or mitigate the short flash lifetime 1) efficiently use cells, 2) reduce the write traffic to flash. Most research uses the first method, e.g., wear leveling. Since flash lifetime is directly related to cycles of program/erase, reducing write traffic to flash will extend the flash lifetime. However, little research has employed the second method. We did find Soundararajan et al.’s paper [8] using the second method. However, they use disk-based write cache instead of DRAM cache to save write traffic. Their research was based on a much larger cache size (90MB-2728MB). The cache size of 15MB to 60MB was not investigated. Therefore, their conclusions do not apply to a DRAM cache. In addition, they used a log file, which results in read penalty and affects performance and response time. By using HDDs in SSDs, SSDs will lose their benefits in power consumption, weight and size, which will prevent its wide use. RAM buffer has been proposed to improve the flash performance. Park et al. [9] reduce writes by evicting clean data first. Jo et al. [10] propose a flash aware buffer management scheme to decrease the number of erase operations by choosing victims based upon page utilization. Kim et al. [11] use three key techniques, block-level LRU, page padding, and LRU compensation to improve random write performance. Gupta et al. [12] enhance random write performance by selectively caching page-level address mappings. However, the above studies except Soundararajan's work did not concentrate on the lifetime issue although Kim et al. mentioned the erase counts in their papers. Next, a common problem of using DRAM is the fault tolerance issue inherent in the volatile memory. Although Kim et al. [11] suggested using a small battery or capacitor to delay shutdown until the RAM content is backed up to flash, both batteries and capacitors have their limitations. Batteries have short lifetime, which require maintenance and replacements during the lifetime of devices while capacitors of a reasonable size do not have enough energy sustaining enough time, during which the data is transferred to flash. Finally, flash as a disk cache has not been studied in their research.

In this paper, we focus on solving the flash lifetime issue. We propose to use flash as a victim device of the DRAM cache with a supercapacitor backup power to prolong the flash lifetime. Unlike traditional solutions, our method uses the second method to decrease the traffic to flash.

As an integrated part of HDDs, flash will affect the whole device in terms of lifetime, reliability, and performance. Unlike DRAM and HDDs, flash lifetime is limited by the number of erases performed. Moreover, as the write cycles increases, the flash becomes less reliable. Therefore, error correction code (ECC) must be employed to correct increasing errors. Lifetime is critical for write workloads since flash-based cache is sensitive to write traffic due to the endurance issue.

2.4 Contributions

Intuition tells us that DRAM disk cache can reduce writes by coalescing and merging multiple writes into a single write. **However, on second thought, we doubt that it would be effective given that there is a larger file system cache in main memory above the device level which already merges writes.** For example, typical DRAM disk cache size in HDDs is 16MB/32MB whereas file system cache can be 512MB or all free memory for a PC with 4GB main memory as shown in Fig. 2. Thus, it is far from clear whether such a small DRAM disk cache at device level could indeed reduce traffic to persistent storage.

Moreover, researchers are reluctant to use DRAM in the disk-cache for this purpose because [8] DRAM is volatile memory and thus could cause loss of data upon power failure. Yet, whether DRAM is suitable for reducing traffic depends on the following factors:

- How much write traffic would be reduced by using DRAM cache?
- What is the minimal size of DRAM cache to be effective as a traffic saver?
- How would flash size impact the traffic savings?
- What sort of update policy should be used?

In this paper, we attempt to answer these questions.

3 Related Work

Reducing Write Traffic: Usually, a non-volatile write cache or log region is employed in front of the flash medium to catch the write traffic so that less write traffic would reach the flash medium. Some works [13] [14] use the emerging memory technologies (e.g., PCM) as the non-volatile log region. Unlike flash, PCM supports in-place updating. In addition, PCM is faster than flash. Due to the front end PCM log region, traffic to the flash is decreased.

Soundararajan et al. [8] even use disk-based write caches to extend SSD lifetimes. Their design is based on the following observations. First, there are many overwrites of a small set of popular blocks at the block device level. Second, thanks to the file system cache, there will be not many immediate reads following a write. Finally, HDDs are excellent at sequential writes and reads. Therefore, HDDs fit log-structured write cache perfectly.

Qureshi et al. [15] have proposed using a DRAM buffer to lazy-write to PCM in order to improve write performance as well as lifetime of PCM. However, their target is at the system level rather than at the device level. Additionally, they did not deal with the power failure issue. Therefore, our research is complementary to theirs.

Using DRAM cache with supercapacitor backup power to save write traffic does not need many extra efforts since DRAM is an indispensable part in almost all flash controllers. A little larger DRAM might be needed to be used as a traffic saver. The prices of DRAM have been dropped to the point that adding, for example, 15-60 MB DRAM to the SSD drives does not cost much to the total prices of the devices.

4 System Architecture

There are three major usage models of flash memory:

1) System memory model
2) Disk caches
3) Storage devices (SSDs)

Since our focus is on secondary storage, the system memory model is beyond the scope of this paper. Therefore, we deal with two usages for flash: as disk caches in Hybrid Hard Drives (HHD) and as an independent Solid State Drive (SSD). For both categories, we propose to use a DRAM cache to filter write traffic. The use of DRAM in the first category, i.e. in hybrid hard drive, is shown in Fig. 3. It shows a DRAM interface to flash; the flash is used as a cache for the HDD in this case. The DRAM is used to minimize the traffic to the flash used as a disk cache. This design is further elaborated in Fig. 5. In Fig. 5, the evictions from the DRAM are retired to the flash through the controller. The functioning of this unit for reads and writes, with early and lazy updates, is further elaborated through the flowcharts in Figures 8, 9, 18 and 19. The second category is shown in Fig. 4. There is only one level of disk cache. Like the first category, flash is updated only when the data is evicted from the DRAM cache. The main purpose for doing that is to reduce write traffic to flash thereby extending the lifetime of flash.

4.1 Comparison of Traffic Mitigation Techniques

The idea of traffic mitigation is to have a non-volatile cache in front of the flash medium. Apart from our solution (DRAM with a supercapacitor backup power), such kind of cache can be:

- Battery-backed DRAM
- SLC as a write cache for MLC
- PCM as a write cache
- HDDs

Since a thorough comparative analysis of all the options is beyond the scope of this paper, we briefly describe a few other designs and compare them qualitatively with our solution.

4.1.1 Battery-backed DRAM as a write cache

The difference between battery-backed DRAM and our solution lies only in the way the backup power is supplied. As calculated in Section 4.2, a period of 10 seconds is long enough for the contents in DRAM to be transferred into flash. Therefore, we do not need a long last power supply in the presence of flash. A supercapacitor backup power is a perfect fit. It does not suffer from many drawbacks of batteries, such as regular maintenance or replacement, limited charge/discharge cycles, slow charge, degrade with shallow discharge.

4.1.2 SLC as a write cache for MLC

SLC can be a write cache for MLC due to the fact that each SLC block has more write cycles. However, SLC is expensive. To be feasible, the size of SLC must be small. The problem with SLC is that SLC also suffers from write endurance although write endurance is 10 times better than MLC. Therefore, SLC endurance should be taken into account as a design constraint along with MLC endurance. Since SLC has 10 times the endurance of MLC, to reach the same lifetime, SLC can be a tenth as large as MLC. According to Soundararajan et al. [8], if SLC receives twice as many writes as MLC (where MLC receives 50% write savings), SLC should be a fifth as large as MLC. Hence, the larger the backing MLC SSD, the larger the SLC cache. For example, a 80GB MLC SSD, 16GB SLC is needed. It is believed that 16GB SLC will continue to be expensive enough for a 80GB MLC SSD to afford. In contrast, the size of primary store in our solution has little impact on the size of cache, i.e., the size of cache alone determines the write savings no matter how large the primary store is.

4.1.3 PCM as a write cache

Phase-change memory (also known as PCM, PRAM, PCRAM, etc.) is a type of non-volatile computer memory. According to ITRS [7], it is in the status of development underway and at the border of qualification/pre-production. PCM is one of a number of new memory technologies competing in the non-volatile role with the almost universal flash memory. PCM is a potential choice to be a write cache for SLC/MLC due to the following characteristics [16]:

- Nonvolatile
- Fast read speeds: Access times comparable to DRAM.
- Fast write speeds: Significant write speed improvement over NOR and NAND flash and no erase needed.

However, The write speed of PCM is far less than that of DRAM. Additionally, PCM still suffers from write endurance issues (10^8) although it improves a lot over SLC/MLC. It would still be a concern for write-intensive applications. It is not yet clear that how the future of PCM will be in terms of price and the production readiness. Only time will tell. However, our study does not exclude the use of PCM.

4.1.4 HDDs as a write cache

HDDs have been proposed to be a write cache for flash to take advantage of the fact that a SATA disk drive can deliver over 80 MB/s of sequential write bandwidth. Two observed characteristics support the use of HDDs as a write cache. First, at block level, there are many overwrites of a small set of popular blocks. Second, reads do not follow writes immediately, which gives the write cache a grace period to flush the contents into flash before reads. There are two competing imperatives: On one hand, data should stay longer in write cache in order to catch more overwrites. On the other hand, data should be flushed into flash in advance to avoid expensive read penalty from HDDs. Therefore, many triggers to flush must be designed to achieve the goal of more overwrites with less read penalty, which makes it quite complicated. In contrast, data in our solution can stay as long as the capacity of the DRAM cache allows since there is no read penalty.

Flash-based SSDs have many advantages over HDDs in terms of:

- Acoustic levels: SSDs have no moving parts and make no sound unlike HDDs.
- Mechanical reliability: SSDs contain no moving parts thereby virtually eliminating mechanical breakdowns.
- Susceptibility to environmental factors: Compared to traditional HDDs, SSDs are typically less susceptible to physical shock. Additionally, they have wider temperature ranges.
- Weight and size: The weight of flash memory and the circuit board material are very light compared to HDDs.

- Power consumption: High performance flash-based SSDs generally require less power than HDDs.
- Magnetic susceptibility: SSDs are not concerned about magnets or magnetic surges, which can alter data on the HDD media.

By introducing HDDs into SSDs, the device as a whole will lose its advantages in the above aspects. Therefore, using HDDs as write cache of SSDs may hinder their wider applications.

Our experimental results show no less write savings using DRAM cache than using HDD cache although the comparison may not be precise since we use different workload traces from Soundararajan et al. [8].

4.1.5 *Summary*

The comparison between our research and several others is presented in Table 1. As we can see from the table, log-structure has been employed in [8] [13] [14]. As mentioned in [8], log-structure is good at writing but it incurs read-penalty.

4.2 Fault Tolerance Issue

There is a fault tolerance issue with DRAM cache since DRAM is volatile memory, which will lose data when the power is off. There is battery-backed DRAM. However, batteries have many issues, such as lifetime issue, maintenance issue, recharge-time issue. In this paper, we propose to use supercapacitors as backup power for the controller and DRAM as shown in Fig. 5 (flash as disk cache) and Fig. 6 (flash as major store).

Supercapacitor backup power has been used by Seagate [17] in their SSDs and Sun Oracle [18] in their storage systems, which approves that supercapacitors can fit well in the 2.5-inch form factor. Although the use of supercapacitors as a backup power source is not new, we have not found it being used to solve the flash lifetime issue.

4.2.1 *What are Supercapacitors?*

A supercapacitor (also known as ultracapacitor) is an electrochemical capacitor that offers very high capacitance in a small package. The amount of energy a capacitor can hold is measured in microfarads or μF. (1μF = 10^{-6} Farad). While small capacitors are rated in nano-farads (nF=10^{-9}F) and pico-farads (1pF = 10^{-12}F), supercapacitors come in farads.

4.2.2 *Why Supercapacitors not Batteries?*

Unlike the electrochemical battery, there is very little wear and tear induced by cycling and age does not affect the supercapacitor much. In normal use, a supercapacitor deteriorates to about 80 percent after 10 years, which is long enough for most applications whereas a battery needs many replacements during the lifetime of a device. Additionally, supercapacitors do not need a full charge detection circuit like rechargeable batteries. They take as

TABLE 1
COMPARISON OF TRAFFIC MITIGATION TECHNIQUES

Technique→	**Kim et al. [13]**	**Sun et al. [14]**	**Qureshi et al. [15]**	**Soundararajan et al. [8]**	**Ours**
Media	PCM	PCM	DRAM	Disk-based	DRAM (supercapacitor)
Traffic handled	Metadata	Metadata	All data	All data	All data
Update speed	Slow (PCM)	Slow (PCM)	Fast (DRAM)	Slow (HDD)	Fast (DRAM)
Role of flash	Main storage	Main storage	N/A	Main storage	Disk cache/main storage
Usage	Embedded devices	Secondary storage	Main memory	Secondary storage	Secondary storage
Workloads	Block device	Block device	Main memory	Block device	Block device
Read penalty	Yes	Yes	No	Yes	No
Data loss upon power failure?	No	No	Yes	No	No
Comparative added cost?	High	High	Low	High	Low
Matured technology	No (PCM)	No (PCM)	No(PCM)	Yes (HDD)	Yes (DRAM)

much energy as needed. When full, they stop accepting charges. There is no danger of overcharge or memory.

The supercapacitor offers high power density although the energy density is far below that of the battery as depicted in Fig. 7. Today, supercapacitors can store 5%-10% as much energy as a modern lithium-ion battery of the same size. What supercapacitors lack in range, they make up in the ability to rapidly charge and discharge. They can be charged in seconds rather than in minutes or hours. Supercapacitors are already all over the places. Millions of them provide backup power for the memory used in microcomputers and cell phones. As mentioned above, supercapacitors have been used in enterprise SSDs.

4.2.3 How to calculate the Capacitance of Supercapacitors?

The value of a supercapacitor can be estimated [56] by equating the energy needed during the hold-up period to the energy decrease in the supercapacitor, starting at V_{wv} and ending at V_{min}.

The energy (E_x) needed during the hold-up period (t):

$$E_x = I\frac{V_{wv} + V_{min}}{2}t \tag{1}$$

The energy decrease (E_y) as voltage drops from V_{wv} to V_{min}:

$$E_y = \frac{CV_{wv}^2}{2} - \frac{CV_{min}^2}{2} \tag{2}$$

Since $E_x = E_y$, the minimum capacitance value that guarantees hold-up to V_{min} is:

$$C = I\frac{V_{wv} + V_{min}}{V_{mv}^2 - V_{min}^2}t \tag{3}$$

When the main power is turned off for any reason, the supercapacitor backup power needs to provide the temporary power long enough for the controller to transfer the dirty data into flash.

Suppose we use 64MB of DRAM cache, which is large enough to have an effective write traffic savings. We use Intel X-25M SSD drives [19] to estimate the transfer time. The drive needs 5V (+/-5%) power and the active power is 150mW (current is 0.15/5=0.03A). The sustained sequential write speed is 70MB for 80 GB SSD drives. Therefore, in less than one second, 64MB will be moved into flash. We use 2 seconds to calculate the minimum capacitance value. Applying (3), we get C=0.12F. According to Maxwell [20], supercapacitor prices will be approaching \$0.01 per farad in production volumes of millions. Apart from supercapacitors, a power control circuit should be added. However, the total cost would not be high.

5 METHODOLOGY

Extensive experiments were conducted with a disk simulator. Our simulator was based on DiskSim 4.0 and Microsoft SSD add-on, which were implemented in C. In this Section, we describe the methodology we used in this paper.

TABLE 2
WORKLOAD CHARACTERISTICS

	Dev[a]	**TIOR**[b]	**Reads**	**IOPS**[c]	**ARS**[d]**(KB)**
OpenMail	080	51295	36373 (70%)		5.0/10.0/6.5
	096	363789	119400 (32%)	98.29	7.8/7.3/7.5
	097	366223	117530 (32%)		7.9/7.5/7.6
	098	421270	219553 (52%)		3.6/5.6/4.6
	099	424887	227744 (53%)		3.5/5.6/4.5
	100	295536	152372 (51%)		3.5/5.4/4.4
UMTR	0	1439434	1439434 (100%)	5.27	15.2/-/15.2
Web3	1	1410370	1409113 (99%)		15/26/15
	2	1410493	1410492 (99%)		15/8.0/15.5
	3	489	487 (99%)		15/8.0/15
	4	486	486 (100%)		15.1/-/15.1
	5	434	434 (100%)		16.3/-/16.3
Synthetic workloads	0	10000	6600 (66%)	40.05	6.4/6.2/6.3

[a] Device No
[b] Total I/O Requests
[c] I/O Per Second
[d] Average Request Size in KB

5.1 Simulator

We evaluated the proposed architecture using DiskSim 4.0 [21] with flash extension. DiskSim is a widely used disk drive simulator both in academia and industry alike. DiskSim is an event-driven simulator. It emulates a hierarchy of storage components such as buses and controllers as well as disks. In 2008, Microsoft research implemented an SSD module [22] on top of DiskSim. Based on that, we made some changes to the code to reflect our architecture.

5.2 Workloads and Traces

Many studies of storage performance analysis use traces of real file system requests to produce more realistic results. The traces we used are from HP [23] [24] [25]: OpenMail. In addition, we used disk traces from University of Massachusetts Trace Repository (UMTR) [26] to test the impact of different update policies on the disk behavior of enterprise level applications like web servers, database servers, and web search. Since the traces were recorded from the device level, they already include the impact of the file system cache in the main memory shown in Fig. 2. We also generate synthetic workloads that represent typical access distributions and approximate real disk usage. The characteristics of workloads used in this paper are shown in Table 2. We selected the first device (which can represent the workload characteristics in class) within each trace to report simulation results.

OpenMail [23]: It was a one-hour trace from five servers running HP's OpenMail, collected during the servers' busy periods. The trace was collected in 1999.

UMTR [26]: There are two kinds of traces: OLTP application I/O and search engine I/O. The former includes two I/O traces (Financial1.spc and Financial2.spc) from OLTP applications running at two large financial institutions. The later includes three traces (Websearch1.spc, Websearch2.spc, and Websearch3.spc) from a popular search engine. These traces are made available courtesy of Ken Bates from HP, Bruce McNutt from IBM and the Storage Performance Council.

Synthetic workloads [21]: The synthetic workloads were generated using DiskSim 4.0 workload generator. The generator can be configured to generate a wide range of synthetic workloads. In addition, probability-distribution parameters can be set to uniform, normal, exponential, Poisson, or twovalue.

5.3 Performance Metrics

Response time and *throughput* are generally two important metrics in measuring I/O performance. Response time starts from a request being issued until the request is served. Throughput measures the ability of a system in a form of the number of I/Os per second. The difference between *Response time* and *Throughput* is whether we measure one task (*Response Time*) or many tasks (*Throughput*). According to Hsu and Smith [1], throughput is hard to be quantified for trace-driven simulation in that the workloads are constant. However, they maintain that throughput can be estimated by taking the reciprocal of the average service time, which tends to be optimistic estimate of the maximum throughput. Additionally, we examine the *miss ratio* of the read cache and the write cache. The miss ratio is the fraction of I/Os that need to access physical devices (HDDs). Finally, flash endurance is measured using the write traffic (Bytes) to the flash. To compare different policies (early update policy and lazy update policy), we introduce *Relative Traffic* γ, which is defined as:

$$\gamma = \frac{\beta}{\alpha} \tag{4}$$

where

- β is the write traffic to flash.
- α is the write traffic to device.

The ideal flash lifetime, which does not take write amplification into consideration (write amplification will be discussed later), can be expressed as:

$$\eta = \frac{\delta\varepsilon}{\lambda} \tag{5}$$

where

- η is the flash lifetime in days.
- δ is the maximum of the flash write cycles (10K-100K) depending on the type of flash.
- ε is the capacity of the device.
- λ is the write traffic per day.

As we can see from (5), flash lifetime is related to rated write cycles δ and the capacity of the device ε. Since we will use the same kinds of flash and same size of flash to compare early update policy to lazy update policy, we would like to eliminate these two factors from the equation. In order to do so, we introduce *relative lifetime* φ:

$$\varphi = \frac{\eta 1}{\eta 2} = \frac{\frac{\delta\varepsilon}{\lambda 1}}{\frac{\delta\varepsilon}{\lambda 2}} = \frac{\lambda 2\mu}{\lambda 1\mu} = \frac{\beta 2}{\beta 1} = \frac{\frac{\beta 2}{\alpha}}{\frac{\beta 1}{\alpha}} = \frac{\gamma 2}{\gamma 1} \tag{6}$$

where

- $\eta 1$ is the flash lifetime for early update policy.
- $\eta 2$ is the flash lifetime for lazy update policy.
- μ is the time in days the flash is used.
- $\lambda 1$ is the write traffic to flash per day for early update policy.
- $\lambda 2$ is the write traffic to flash per day for lazy update policy.
- $\beta 1$ is the write traffic to flash for early update policy.
- $\beta 2$ is the write traffic to flash for lazy update policy.
- α is the total traffic to the device.
- $\gamma 1$ is relative traffic for early update policy.
- $\gamma 2$ is relative traffic for lazy update policy.

From (6), we see that relative lifetime φ can be expressed using relative traffic γ. The advantage of using relative traffic to compare relative lifetime is that the rated write cycles δ and the capacity of the device ε are taken out of the equation.

The flash lifetime in practice is smaller than the ideal flash lifetime due to a factor, called write amplification. Write amplification is referred to as the phenomenon that n bytes of write traffic from a file system will be translated into nm ($m > 1$) bytes of write traffic to flash. There are several factors [27] that contribute to write amplification. First, write amplification is caused by wear leveling, where cold/hot data swapping consumes extra write cycles. Second, garbage collection contributes to write amplification. In garbage collection, in order to reclaim invalid pages, the valid pages within the reclaiming blocks need to be relocated, which consumes extra write cycles. Write amplification varies for different FTL. Usually, simpler FTLs own larger write amplification while more complex FTLs have smaller write amplification. Many efforts have been made to minimize write amplification to extend flash lifetime. According to Soundararajan et al. [8], flash lifetime can be an order of magnitude worse than the optimum even for advanced FTLs, such as Intel X25-M MLC SSD.

Due to the write amplification, it is not straightforward to map between reduced write traffic and increased lifetime. However, decreasing the write traffic in half will at least double its lifetime because of reduced write amplification [8].

5.4 DiskSim 4.0 modifications

In doing our research, we made some modifications on DiskSim4.0. First, we added two trace formats: SRT 1.6 and SPC. In addition, we added a trace filter to control what types of requests will be fed into our simulator.

Next, we added secondary flash cache on top of the primary disk cache. Last, we fixed bugs with regard to the incompatibility issue of Type 3 Smart Controller for Microsoft SSD add-on.

5.5 Validation

DiskSim 4.0 comes with a set of validation tests (runvalid), which are used to test the simulator. Runvalid uses a series of validation data to validate the simulator. The validation data were from a logic analyzer attached to the SCSI bus. Validation was achieved by comparing measured and simulated response time distributions. Our modified version of DiskSim 4.0 produces the same values as DiskSim 4.0 for all the validation tests.

6 Traffic Savings for Flash as a Victim Disk Cache

In this Section, we focus our research on flash being as a victim disk cache, whose architecture was shown in Fig. 3. There are two levels of disk caches: primary DRAM disk cache and secondary flash disk cache. We concentrated on the relationship between the two levels of disk caches: when is the right time to update the flash disk cache?

6.1 Early Update vs. Lazy Update

Early update means the flash is updated as soon as the DRAM cache is updated. Lazy update is referred to as the policy that the flash is updated only when the data is evicted from DRAM cache. In other words, flash acts as a victim device of the DRAM cache. Traditionally, early update policy is employed. For examples, Bisson et al. [28] and Kgil et al. [29] [30] [31] use early update policy.

6.2 Lazy Update Policy for Reads

When a read request arrives, DRAM cache is first checked as shown in Fig. 8. If found in DRAM cache, then the request can be served through DRAM cache. If not, then check with flash. If it has an entry in it, the request can be satisfied by flash. The data will also be copied into DRAM cache. If the data is not in flash either, then the data is fetched from HDDs and it is cached in DRAM cache. But flash is not updated until the data is evicted.

6.3 Lazy Update Policy for Writes

When a write request arrives, DRAM cache is first checked as shown in Fig. 9. If found in DRAM cache, the request will be merged. If not, allocate an entry in DRAM cache. If there is no space, the least used clean data will be evicted. The evicted data will be copied into flash. If all data in DRAM cache are dirty, the write request has to wait for the dirty data to be retired into HDDs.

6.4 The Benefits of Lazy Update Policy

Lazy update policy benefits from the fact that block devices receive many overwrites of a small set of popular blocks. One cause for overwrites is that many file systems enforce a 30-second rule [25], which flushes buffered writes to disks every 30 seconds for the sake of data integrity. Soundararajan et al. [8] have found that, on average, 54% of the total overwrites occur within the first 30 seconds, which confirms the role that 30-second rule plays. Some file systems, such as ext2, flush metadata more frequently (e.g., every 5 seconds) [32]. With lazy update policy, overwrites can be coalesced thereby reducing the write traffic to flash significantly.

6.5 Experimental Results

We ran OpenMail, Synthetic, and Websearch3 workload traces against early update policy and lazy update policy, during which relative traffic, miss ratio, and average response time were observed. We especially watched the impact of DRAM size and flash size on relative traffic savings. The simulation results are shown in Fig. 10 through Fig. 17. Several observations can be made from these figures.

6.5.1 Write Traffic Savings

OpenMail (Fig. 10 and Fig. 11): relative traffic with lazy update policy is 50% of that with early update policy when DRAM size reaches 25MB (0.33 for lazy update policy vs. 0.68 for early update policy).

Synthetic workload (Fig. 12 and Fig. 13): relative traffic with lazy update policy is 50% of that with early update policy when DRAM size reaches 60MB (roughly 0.5 for lazy update policy vs. 1 for early update policy).

Websearch3 : relative traffic does not show improvement with Websearch3 because Websearch3 is a read-only workload. For read-only workloads, there is no overwrite that can be coalesced using lazy update policy.

6.5.2 DRAM Size Need Not to be Very Large to be Effective

DRAM size ranges from 15MB to 60MB depending on the workloads, which is shown in Fig. 10 and Fig. 12.

6.5.3 Flash size has little effect on Write Traffic Savings

As can be seen from Fig. 11 and Fig. 13, flash size has little impact on write traffic savings when compared to varying DRAM size.

We see that the write traffic savings depend mainly on DRAM size rather than flash size. This is a great characteristic since a moderate size of DRAM cache ($<$ 60MB) can extend the flash lifetime significantly no matter how large the flash is.

6.5.4 *Summary*

As we can see from the results, the write traffic savings depend upon the DRAM size as well as the workloads characteristics. The lazy update policy improvement over early update policy for OpenMail and synthetic workloads is presented in Fig. 14, Fig. 15, Fig. 16, and Fig. 17 respectively.

Miss Ratio: we notice that miss ratio with lazy update policy is slightly better than that with early update policy.

Response Time: we observe that response time with lazy update policy remains unchanged.

Traffic Reducing: although we see no big difference in terms of miss ratio and average response time with lazy update policy, the write traffic is significantly reduced.

7 Traffic Savings for Flash as a Major Storage Medium

In this Section, we focus our research on flash used as a primary store instead of a victim disk cache, whose architecture was shown in Fig. 4. Unlike a disk cache, read requests would not entail write traffic to flash. Moreover, as a major storage medium, we assume the flash size is much larger than a disk cache. We still concentrated on the impact that update policies have on the lifetime of flash. Compared with the early update policy, our study shows that using flash as a victim device (which corresponds to lazy update policy) can extend a lot of lifetime. At the same time, the performance in terms of response time improves as well.

7.1 Early Update for reads

When a read request arrives, DRAM cache is first checked. If found, then the request is satisfied. Otherwise, data is read from flash. Meanwhile, DRAM cache is updated upon receiving data from flash. There is no need for DRAM eviction when doing read in early update policy, as there is no dirty data. If the data is not found in the DRAM, it is read from the flash and the DRAM data is simply overwritten without eviction.

7.2 Early Update for writes

When a write request arrives, search in DRAM cache first. If found, merge the data. Otherwise, DRAM cache is updated. Next, update flash with Write Counter incremented.

7.3 Lazy Update for reads

The flowchart of lazy update for reads is shown in Fig. 18. When a read request arrives, search in DRAM cache. If found, return the data. If not, read the data from flash. Next, update DRAM cache with the new data. If a dirty data entry is evicted from DRAM cache, update flash with the Write Counter incremented.

7.4 Lazy Update writes

The flowchart of lazy update for writes is shown in Fig. 19. When a write request arrives, search in DRAM cache. If found, merge the data. Otherwise, DRAM cache is updated. If dirty data is evicted from DRAM cache, update the flash with the Write Counter incremented.

7.5 Experimental Results

We ran OpenMail, Synthetic, and Websearch3 workload traces against early update policy and lazy update policy, during which relative traffic and average response time were observed. We especially watched the impact of DRAM size on relative traffic savings. The simulation results are shown in Fig. 20 through Fig. 23. Several observations can be made from these figures.

7.5.1 *Write Traffic Savings*

OpenMail (Fig. 20): relative traffic with lazy update policy is roughly 33% of that with early update policy when DRAM size reaches 10MB (0.33 for lazy update policy vs. 1 for early update policy).

Synthetic workload (Fig. 21): relative traffic with lazy update policy is roughly 40% of that with early update policy when DRAM size reaches 10MB (roughly 0.4 for lazy update policy vs. 1 for early update policy).

Websearch3 : relative traffic does not show improvement with Websearch3 because Websearch3 is a read-only workload. For read-only workloads, there is no overwrite that can be coalesced using lazy update policy.

7.5.2 *DRAM Size Needs not to be Large to Be Effective*

DRAM size larger than 10MB is effective to reduce write traffic significantly, which is shown in Fig. 20 and Fig. 21.

7.5.3 *Summary*

The lazy update policy improvement over early update policy for OpenMail and synthetic workloads has been presented in Fig. 22 and Fig. 23 respectively.

Response Time: performance in terms of average response time shows marked improvement.

Traffic Reducing: lazy update policy reduces write traffic significantly with read/write workloads.

8 Conclusions

This paper introduced the following main contributions that aim to extend the lifetime of flash:

- **Extend the lifetime of flash in a new way**: DRAM cache has been used to improve performance in terms of response time. Due to its volatile nature, researchers are reluctant to use it to save write traffic. Soundararajan et al. [8] put it this way, "RAM can make for a fast and effective write cache, however the overriding problem with RAM is that it is not persistent (absent some power-continuity arrangements)." The potential of DRAM cache being a write traffic saver has been overlooked.

- **Solve the data integrity**: An Achilles' heel is that the data will be lost in case of power failure. A common way to solve this issue is to have a battery backup power. However, batteries have limited charge/discharge cycles and need regular maintenance or replacements. Instead, we proposed to use a supercapacitor backup power. Supercapacitors are perfect at supplying short period of power in this scenario.

We have demonstrated through simulation that a medium-sized DRAM cache can save up to 50% write traffic to flash, which is translated into at least doubling the lifetime of flash. Meanwhile, performance in terms of response time and miss ratio shows improvement as well. Furthermore, our findings can be applied to computer main memory to enhance data integrity of computer systems.

9 Future Work

Our work in this paper is at device level. However, we realize that the concept of supercapcitor backup power can be applied to computer main memory to enhance data integrity of computer systems as well.

A conventional practice to alleviate data loss due to unexpected power failure is to enforce a 30-second flush rule, as Unix operating systems do. The negative side of the rule is that disk fragmentation is increased, which will decrease its performance. Some important computer systems are even equipped with Uninterruptible Power Supply (UPS) to protect its data.

With a supercapacitor backup power, backing flash, and controller, the content of main memory can be backed up into flash on power loss. The data can be recovered on power resuming. In this context, 30-second flush rule is no longer needed, which would reduce disk fragmentation and improve data integrity as well as performance improvement.

In addition, future research should address the size of DRAM cache for PCM/MRAM memories, retirement policies and the size of super-capacitor needed for effective fault tolerance upon power failure.

Acknowledgments

This work was supported in part by grants DARPA/AFRL FA8750-09-1-0163 and NSF # 0966278

References

[1] W. Hsu and A. J. Smith, "The performance impact of I/O optimizations and disk improvements," *IBM J. Res. Dev.*, vol. 48, no. 2, pp. 255–289, 2004.

[2] Solaris^{TM} ZFS^{TM} enables hybrid storage pools–Shatters economic and performance barriers. [Online]. Available: http://download.intel.com/design/flash/nand/SolarisZFS_SolutionBrief.pdf

[3] "Understanding the flash translation layer (FTL) specification," Intel Corporation, Tech. Rep., 1998.

[4] Y. Kim, A. Gupta, and B. Urgaonkar, "MixedStore: An enterprise-scale storage system combining Solid-state and Hard Disk Drives ," The Pennsylvania State University, Tech. Rep., 2008.

[5] "NAND evolution and its effects on Solid State Drive (SSD) useable life," Western Digital, White paper WP-001-01R, 2009.

[6] J. Hutchby and M. Garner. Assessment of the potential & maturity of selected emerging research memory technologies. Workshop & ERD/ERM Working Group Meeting (April 6-7, 2010). [Online]. Available: http://www.itrs.net/Links/2010ITRS/2010Update/ToPost/ERD_ERM_2010FINALReportMemoryAssessment_ITRS.pdf

[7] (2010) Process integration, devices & structures. The International Technology Roadmap for Semiconductors. [Online]. Available: http://www.itrs.net/Links/2010ITRS/Home2010.htm

[8] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending SSD lifetimes with disk-based write caches," in *Proceedings of the 8th USENIX conference on File and storage technologies*, ser. FAST'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 8–8. [Online]. Available: http://portal.acm.org/citation.cfm?id=1855511.1855519

[9] S.-y. Park, D. Jung, J.-u. Kang, J.-s. Kim, and J. Lee, "CFLRU: a replacement algorithm for flash memory," in *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, ser. CASES '06. New York, NY, USA: ACM, 2006, pp. 234–241. [Online]. Available: http://doi.acm.org/10.1145/1176760.1176789

[10] H. Jo, J.-U. Kang, S.-Y. Park, J.-S. Kim, and J. Lee, "FAB: flash-aware buffer management policy for portable media players," *Consumer Electronics, IEEE Transactions on*, vol. 52, no. 2, pp. 485 – 493, May 2006.

[11] H. Kim and S. Ahn, "BPLRU: a buffer management scheme for improving random writes in flash storage," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, ser. FAST'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 16:1–16:14. [Online]. Available: http://portal.acm.org/citation.cfm?id=1364813.1364829

[12] 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*. New York, NY, USA: ACM, 2009, pp. 229–240.

[13] 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 *Proceedings of the 8th ACM international conference on Embedded software*, ser. EMSOFT '08. New York, NY, USA: ACM, 2008, pp. 31–40. [Online]. Available: http://doi.acm.org/10.1145/1450058.1450064

[14] G. Sun, Y. Joo, Y. Chen, D. Niu, Y. Xie, Y. Chen, and H. Li, "A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, 9-14 2010, pp. 1 –12.

[15] 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 annual international symposium on Computer architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 24–33. [Online]. Available: http://doi.acm.org/10.1145/1555754.1555760

[16] "The basics of phase change memory (PCM) technology," Numonyx White Paper. [Online]. Available: www.numonyx.com/Documents/WhitePapers/PCM_Basics_WP.pdf

[17] Pulsar. Seagate. [Online]. Available: http://www.seagate.com/staticfiles/support/disc/manuals/ssd/100596473a.pdf

[18] Sun Storage F5100 flash array. Sun Oracle. [Online]. Available: http://www.oracle.com/us/products/servers-storage/storage/disk-storage/043970.pdf

[19] Intel® X18-M/X25-M SATA Solid State Drive-34 nm product line. Intel Corporation. [Online]. Available: http://download.intel.com/design/flash/nand/mainstream/322296.pdf

[20] A. Schneuwly, G. Sartorelli, J. Auer, and B. Maher. Ultracapacitor applications in the power electronic world. Maxwell Technologies. [Online]. Available: http://www.maxwell.com/ultracapacitors/white-papers/power_electronic_applications.pdf

[21] J. S. Bucy, J. Schindler, S. Schlosser, G. R. Ganger, and Con-

tributors, *The DiskSim simulation environment version 4.0 reference manual*, Carnegie Mellon University, Pittsburgh, PA, 2008.

[22] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, “Design tradeoffs for SSD performance,” in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2008, pp. 57–70. [Online]. Available: http://portal.acm.org/citation.cfm?id=1404014.1404019

[23] HP open source software. [Online]. Available: http://tesla.hpl.hp.com/opensource/

[24] Block I/O traces from SNIA. [Online]. Available: http://iotta.snia.org/traces/list/BlockIO

[25] C. Ruemmler and J. Wilkes, “Unix disk access patterns.” in *USENIX Winter'93*, 1993, pp. 405–420.

[26] University of Massachusetts trace. [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage

[27] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, “Write amplification analysis in flash-based solid state drives,” in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, ser. SYSTOR '09. New York, NY, USA: ACM, 2009, pp. 10:1–10:9. [Online]. Available: http://doi.acm.org/10.1145/1534530.1534544

[28] T. Bisson and S. A. Brandt, “Reducing Hybrid Disk write latency with flash-backed I/O requests,” in *Proceedings of the 2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 402–409. [Online]. Available: http://portal.acm.org/citation.cfm?id=1474555.1475519

[29] T. Kgil and T. Mudge, “Flashcache: a NAND flash memory file cache for low power web servers,” in *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, ser. CASES '06. New York, NY, USA: ACM, 2006, pp. 103–112. [Online]. Available: http://doi.acm.org/10.1145/1176760.1176774

[30] T. Kgil, D. Roberts, and T. Mudge, “Improving NAND flash based disk caches,” in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ser. ISCA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 327–338. [Online]. Available: http://dx.doi.org/10.1109/ISCA.2008.32

[31] D. Roberts, T. Kgil, and T. Mudge, “Integrating NAND flash devices onto servers,” *Commun. ACM*, vol. 52, pp. 98–103, April 2009. [Online]. Available: http://doi.acm.org/10.1145/1498765.1498791

[32] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, “Analysis and evolution of journaling file systems,” in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 8–8. [Online]. Available: http://portal.acm.org/citation.cfm?id=1247360.1247368

Chengjun Wang received the BS degree in Radio Electronics from Shandong University in 1988, the ME in Computer Engineering from Chinese Academy of Sciences in 1992, the Ph.D. degree in Computer Science from Auburn University in 2011. Currently, he is an assistant professor in the Department of Computer Information Systems at Vermont Technical College. His research interests are in computer architecture, storage systems, and embedded computing.

Sanjeev Baskiyar received the BS degree from Indian Institute of Science, Bangalore, MSEE and PhD degrees from University of Minnesota, Minneapolis. Currently he is Associate Professor in the department of Computer Science and Software Engineering at Auburn University, Auburn, AL. His research interests are in Computer Architecture, Scheduling and Real-time and Embedded Computing.

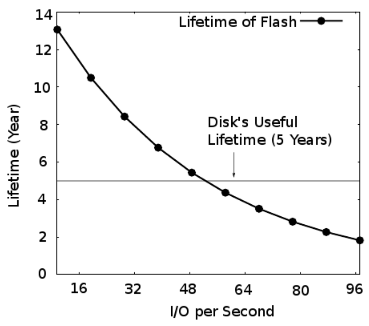

Fig. 1. Lifetime of Flash

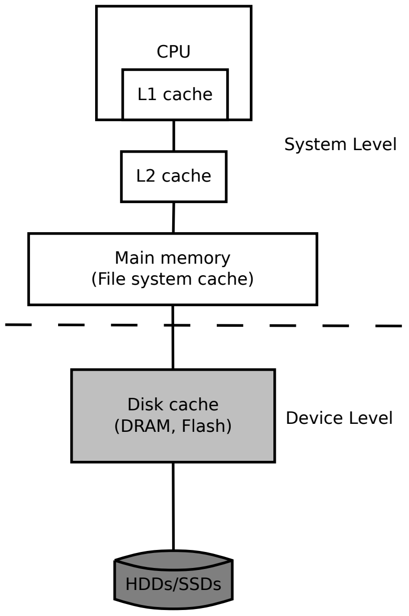

Fig. 2. Memory hierarchy of computers

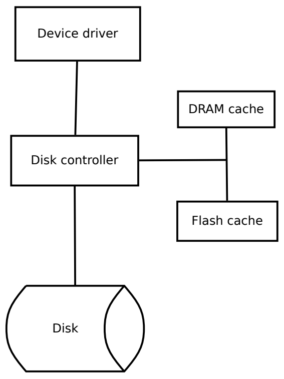

Fig. 3. System architecture of Hybrid Hard Drives (HHD)

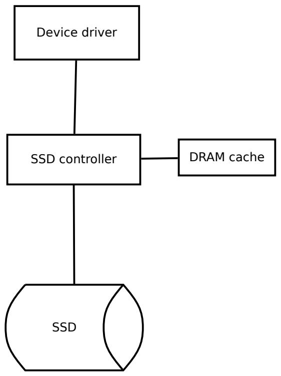

Fig. 4. System architecture of Solid State Drives (SSD)

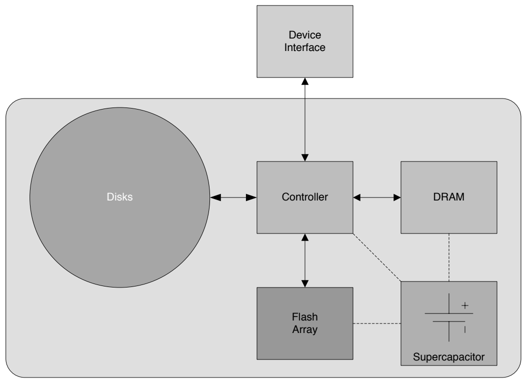

Fig. 5. Flash as disk cache with a supercapacitor backup power

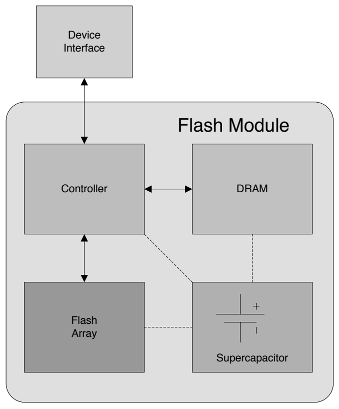

Fig. 6. Flash as major store with a supercapacitor backup power

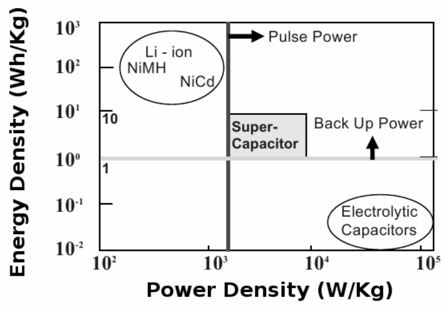

Fig. 7. Supercapacitor and rechargeable batteries

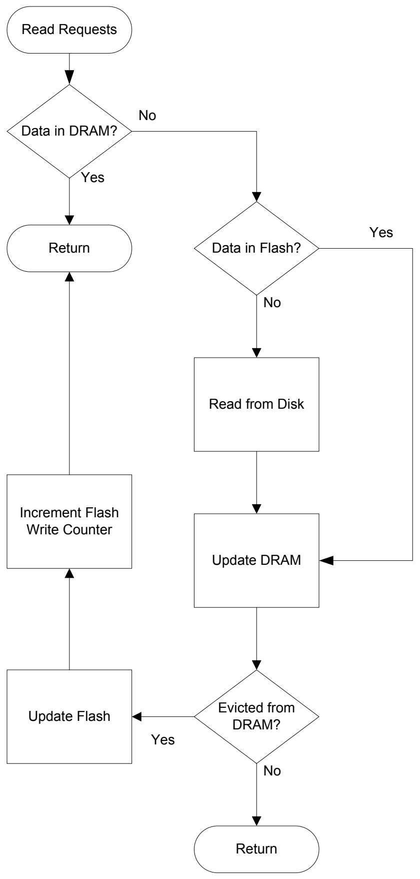

Fig. 8. Lazy update for reads (HHD)

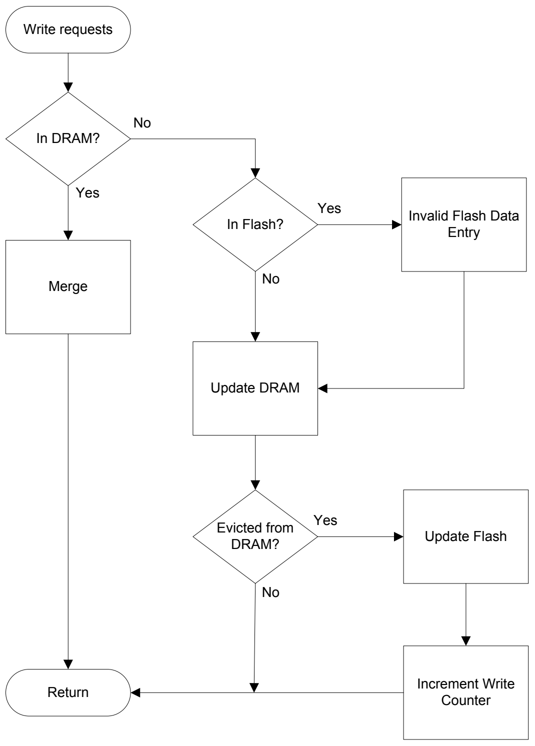

Fig. 9. Lazy update for writes (HHD)

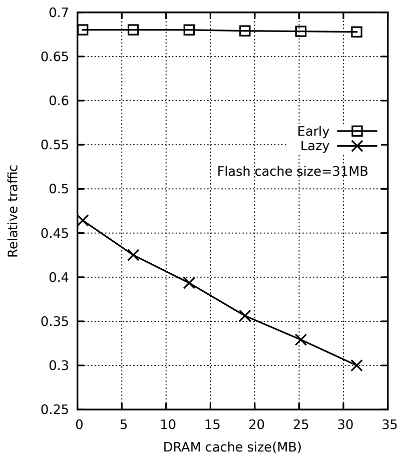

Fig. 10. Relative traffic for OpenMail (HHD)

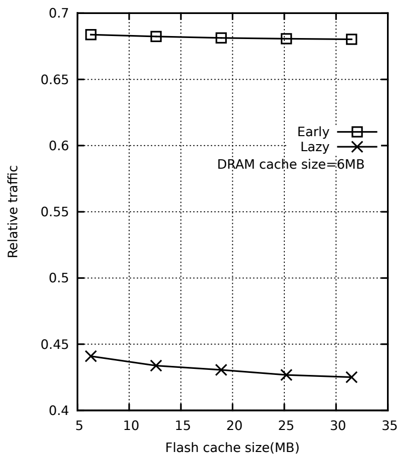

Fig. 11. Relative traffic for OpenMail (HHD)

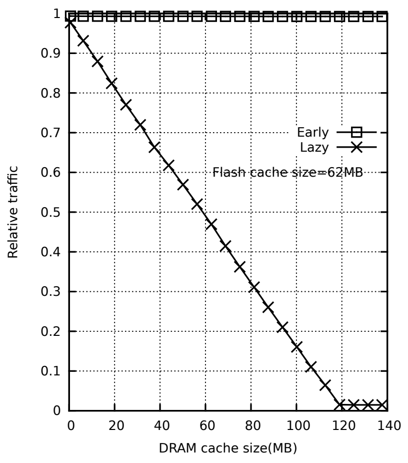

Fig. 12. Relative traffic for Synthetic workload (HHD)

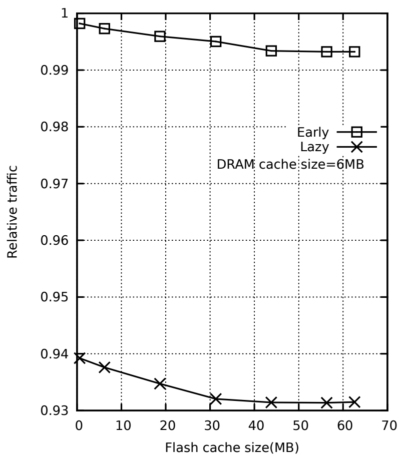

Fig. 13. Relative traffic for Synthetic workload (HHD)

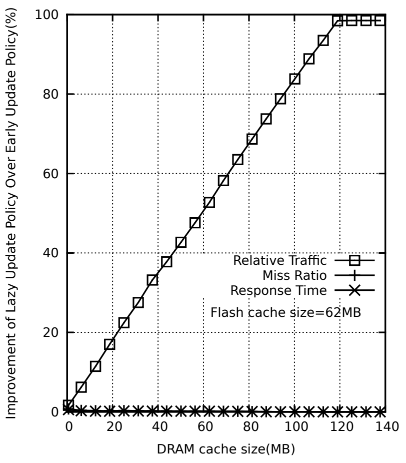

Fig. 16. Improvement of lazy update policy over early update policy for synthetic workload (HHD)

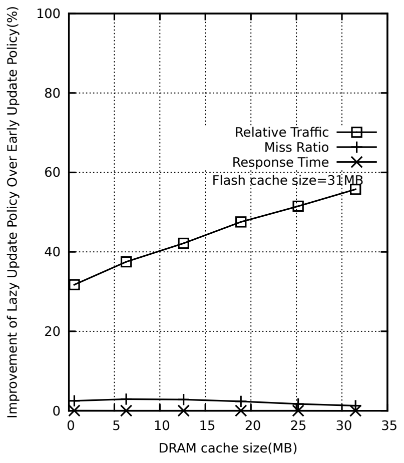

Fig. 14. Improvement of lazy update policy over early update policy for OpenMail (HHD)

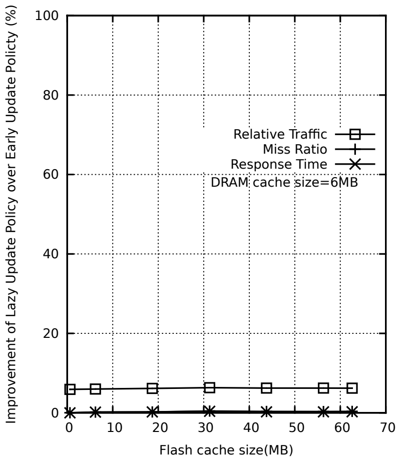

Fig. 17. Improvement of lazy update policy over early update policy for synthetic workload (HHD)

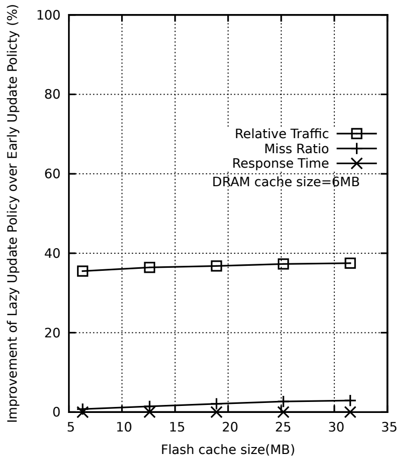

Fig. 15. Improvement of lazy update policy over early update policy for OpenMail (HHD)

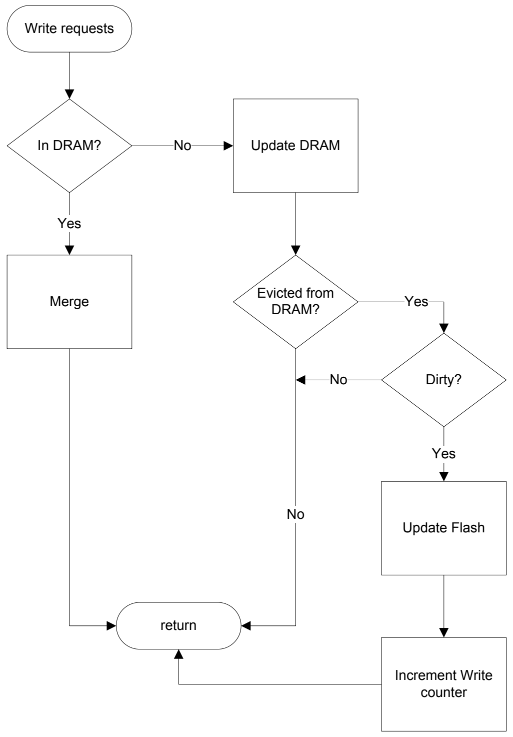

Fig. 19. Lazy update for writes (SSD)

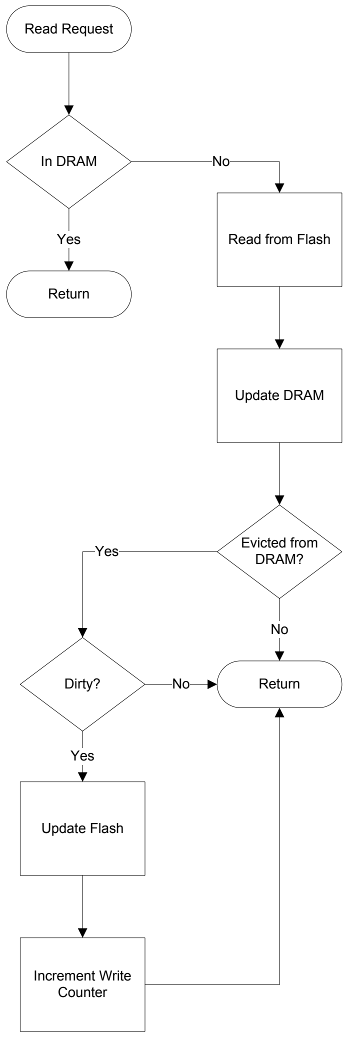

Fig. 18. Lazy update for reads (SSD)

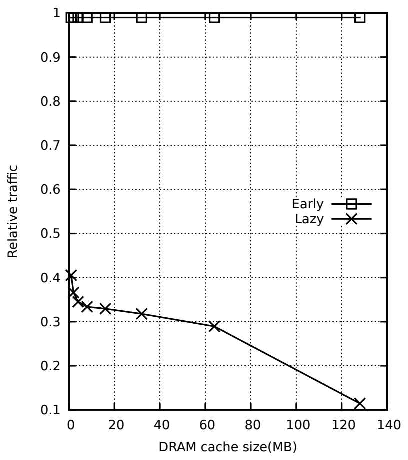

Fig. 20. Relative traffic for OpenMail (SSD)

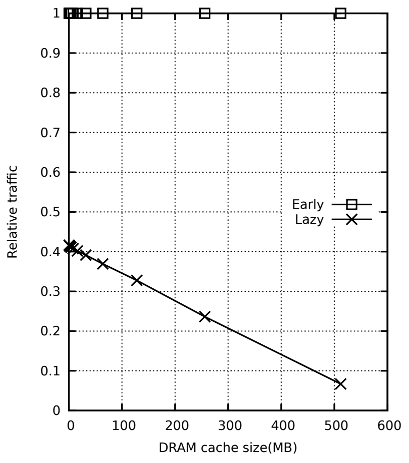

Fig. 21. Relative traffic for Synthetic workloads (SSD)

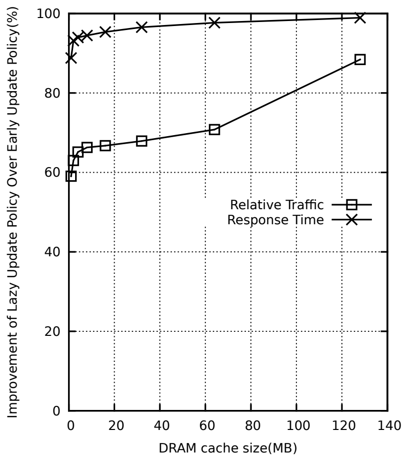

Fig. 22. Improvement of lazy update policy over early update policy for OpenMail (SSD)

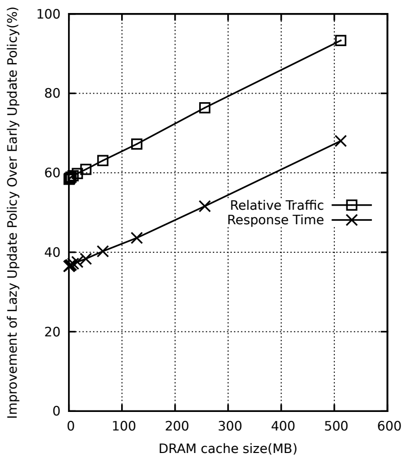

Fig. 23. Improvement of lazy update policy over early update policy for synthetic workload (SSD)