GCaR: Garbage Collection aware Cache Management with Improved Performance for Flash-based SSDs

Suzhen Wu, Yanping Lin
Department of Computer Science
Xiamen University
suzhen@xmu.edu.cn

Bo Mao✉
Software School
Xiamen University
corresponding author
maobo@xmu.edu.cn

Hong Jiang
Department of Computer Science and Engineering
University of Texas at Arlington
hong.jiang@uta.edu

ABSTRACT

Garbage Collection (GC) is an important performance concern for flash-based SSDs, because it tends to disrupt the normal operations of an SSD. This problem continues to plague flash-based storage systems, particularly in the high performance computing and enterprise environment. An important root cause for this problem, as revealed by previous studies, is the serious contention for the flash resources and the severe mutually adversary interference between the user I/O requests and GC-induced I/O requests. The on-board buffer cache within SSDs serves to play an essential role in smoothing the gap between the upper-level applications and the lower-level flash chips and alleviating this problem to some extend. Nevertheless, the existing cache replacement algorithms are well optimized to reduce the miss rate of the buffer cache by reducing the I/O traffic to the flash chips as much as possible, but without considering the GC operations within the flash chips. Consequently, they fail to address the root cause of the problem and thus are far from being sufficient and effective in reducing the expensive I/O traffic to the flash chips that are in the GC state.

To address this important performance issue in flash-based storage systems, particularly in the HPC and enterprise environment, we propose a Garbage Collection aware Replacement policy, called GCaR, to improve the performance of flash-based SSDs. The basic idea is to give higher priority to caching the data blocks belonging to the flash chips that are in the GC state. This substantially lessens the contentions between the user I/O operations and the GC-induced I/O operations. To verify the effectiveness of GCaR, we have integrated it into the SSD extended Disksim simulator. The simulation results show that GCaR can significantly improve the storage performance by reducing the average response time by up to 40.7%.

1. INTRODUCTION

Hard Disk Drives (HDD) have been the primary storage deices in large-scale storage systems for a few decades now. The performance of HDDs has been improved rather slowly relative to other layers/components of the computer systems stack, making them the performance wall of storage systems. With advancements in the semi-conductor technology, flash-based SSDs have emerged as an appealing alternative or supplement to HDDs owing to their many attractive features such as light weight, high random-access performance and shock resistance. As a result, they have received a great deal of attention from both academia and industry. Besides the deployment on mobile devices and desktop/laptop PCs, they have been widely deployed in the high performance computing and enterprise environments [1, 5, 8, 15, 24, 33].

Typically, an SSD consists of multiple flash chips with each containing a large amount of flash blocks. Each block consists of a fixed number of pages [1]. Besides the normal read and write operations, block erase is also a frequent operation within flash-based SSDs because the pages can only be written once before the entire block is erased. Erase operations in flash memory are nearly an order of magnitude slower than write operations. Therefore, flash-based SSDs use out-of-place writes instead of the in-place writes used on HDDs. To reclaim invalid pages and to create free space for writes, SSDs use a Garbage Collection (GC) process [22]. The GC process is a time-consuming operation since it copies valid pages in blocks into the free storage pool and then erases the blocks that do not store valid data. A block erase operation takes approximately 2 milliseconds [1]. Considering that valid pages in the victim blocks need to be copied and then erased, GC overhead can be quite significant.

Usually, GC operations can be executed when there is sufficient idle time (i.e., no incoming I/O requests to SSDs) to minimize the impact to the user performance. However, workloads in server-centric enterprise data centers and High Performance Computing (HPC) environments often have bursts of requests with very short inter-arrival times. These workloads do not exhibit sufficiently long idle times [4, 6] to accommodate GC operations. Examples of enterprise workloads that exhibit this behavior include online-transaction processing applications, such as OLTP and OLAP [5, 26]. Furthermore, it has been found that HPC file systems are routinely stressed with write requests of frequent and periodic checkpointing and journaling operations [27]. In a study of HPC I/O workload characterization of the Spider storage system at Oak Ridge National Laboratory, it was observed that the bandwidth distributions are heavily long-tailed [20]. Thus, in the real environment, most GC operations are triggered on-demand. Our preliminary experiment

ICS '16, June 01-03, 2016, Istanbul, Turkey

DOI: http://dx.doi.org/10.1145/2925426.2926263

results, detailed in Section 2.1, and previous studies [21, 22] observed that the GC operations can significantly degrade the user performance for bursty write-dominant workloads. All these studies have implied or shown that GC is a significant performance concern for flash-based storage systems in enterprise and HPC environments.

There are many studies in the literature addressing the GC issues within flash-based SSDs, including different Flash Translation Layer (FTL) designs [6, 18] and sophisticated data layout schemes that exploit the workload characteristics and internal parallelism [13, 17, 21, 23]. However, since they only try to optimize the random write traffic to alleviate the GC-induced overhead, they all fail to consider the contention for resources and adversary interference between the user I/O traffic and the GC-induced I/O traffic. During an ongoing GC process of a flash chip, any incoming user I/O request to this chip will be delayed until the completion of the GC. Thus, the GC process can significantly impede user I/O performance by increasing their read and write queuing delays.

On the other hand, in order to bridge the performance gap between the upper-level host memory and the lower-level flash storage, the flash-based SSDs are usually embedded with an on-board buffer cache to smooth the user I/O traffic. In other words, the on-board buffer cache has the capability of changing the user accesses seen by the flash chips by absorbing bursty traffic and leveraging access locality with an appropriate cache replacement policy. The existing cache replacement policies, such as LRU and LAMA [11], are well optimized to maximally reduce the buffer cache miss rate by reducing the I/O traffic to the backend devices. The variants of these policies designed to improve GC efficiency, such as BPLRU [19] and GC-ARM [10], only consider how to replace data blocks to improve the GC efficiency and are not aware of the underlying GC operations within flash chips. Thus, none of them takes the GC operations within flash chips into consideration, making them insufficient and ineffective in reducing the expensive I/O traffic to the flash chips that are in the GC state. GC state indicates that a flash chip is conducting garbage collection: coping valid data from a block and erasing the block to free up the flash space for subsequent write data. We argue that, to alleviate the aforementioned contention and interference between the user I/O traffic and GC-induced I/O traffic, the management of the on-board buffer cache should be made aware of the real-time GC activities within the flash chips.

In this paper, we propose a garbage collection aware cache management, named GCaR, to substantially reduce the contention and interference between the user I/O operations and the GC I/O operations. The basic idea is to give a higher priority to caching the blocks on the flash chips that are in GC state. When replacing or destaging a data block in the cache line, we will check whether the data block belongs to the flash chips that are in or will soon be in the GC state. If so, the data block will be kept in the buffer cache for a longer time until the GC completes. Otherwise, it will be replaced or destaged as usual. Different from the existing goal of the cache management aiming to reduce as many user I/O requests as possible, the objective of GCaR is to reduce as many "right" kind of I/O requests as possible to significantly alleviate the contention and interference between the user I/O operations and the GC-induced I/O operations. We have integrated GCaR into the SSD extended Disksim simulator and conducted extensive evaluations. The performance results show that GCaR can significantly improve the storage performance by reducing the average response time by up to 40.7%.

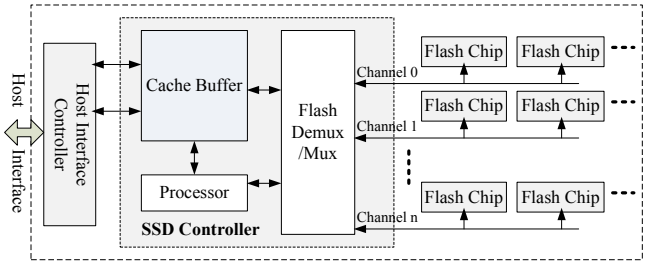


Figure 1: Overview of a typical SSD showing its internal cache buffer and flash chips.

The rest of this paper is organized as follows. Background and motivation are presented in Section 2. We describe the design details of GCaR in Section 3. The performance evaluation is presented in Section 4 and the related work is presented in Section 5. We conclude this paper in Section 6.

2. BACKGROUND AND MOTIVATION

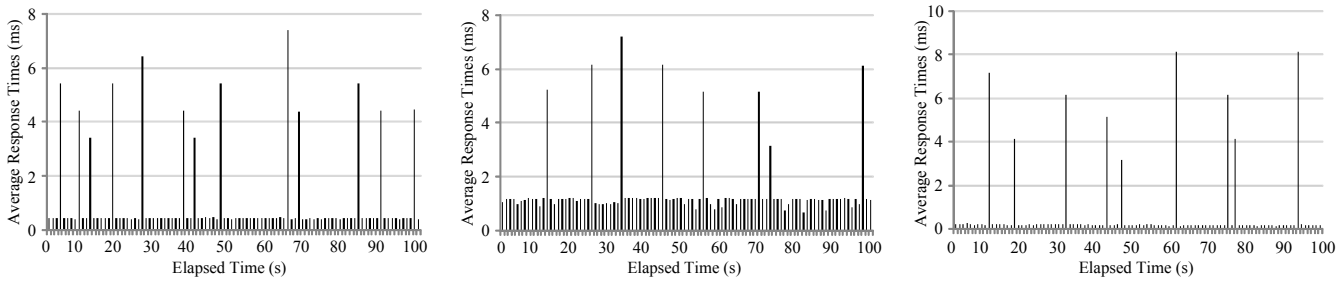
In this section, we first present the key performance characteristics of SSDs most relevant to this research. Then we discuss factors affecting and affected by cache efficiency to motivate our proposed GC-aware cache management for flash-based SSDs.

2.1 SSD basics

Unlike mechanical HDDs, flash-based SSDs are made of silicon memory chips and do not have moving parts (*i.e.*, mechanical positioning parts). Figure 1 illustrates a logical overview of a typical SSD with n independent channels, each channel is shared by multiple flash chips. Despite of the high energy-efficiency and high random-read performance advantages of flash-based SSDs, they have the following two main unique non-HDD characteristics.

First, flash-based SSDs have asymmetric read-write-erase performance [1, 16, 36]. Writing to NAND flash is a multi-step process. In general, to write to cells that have existing data, one must first read the cells, which is followed by erasing the cells and then programming (writing) to the cells. In particular, SSDs suffer from the poor performance of small random-write requests. This leads to erase/program (writing over existing data) operations that are 2-3 orders of magnitude slower than reads. Furthermore, there are at least two types of NAND flash cells: Single-Level Cell (SLC) and Multi-Level Cell (MLC). SLC can store a single bit of data while MLC can store two or more bits. MLC can store at least twice the amount of data compared to SLC. But the read performance of MLC is twice as slow as SLC as could be expected and the write performance is over 3 times slower. To better understand the performance of the NAND flash chips, Table 1 shows the concrete examples of the read-write-erase performance asymmetry and the performance differences between SLC and MLC.

Second, GC is an important but expensive background process in flash-based SSDs. GC eliminates the need to perform erasure of the whole block prior to every write. It accumulates data marked for erase as "Garbage" and performs



(a) Financial1 (b) Prn_0 (c) Prxy_0

Figure 2: The microscopic analysis of the average response times driven by the realistic traces.

Table 1: Read/Write/Erase times for SLC/MLC flash chips [1].

Operations	SLC chips	MLC chips
Random Read	25 us	50 us
Write	250 us	900 us
Erase	2 ms per block	2 ms per block

whole block erase as space reclamation in order to reuse the block. The required GC operations in SSD significantly affect the user I/O performance. The granularity of each read or write operation is a page (2KB-4KB) while that of an erase operation is a block (128KB-256KB). The execution time of an erase operation is one or two orders of magnitude more than that of a write or read operation respectively, as shown in Table 1. The reason is that, for flash-based SSDs, each block (128-256KB) must be erased in advance before any part of it can be written, which is the characteristic feature of flash chips and known as "erase-before-write". Due to the sheer size of a block, an erase operation typically takes milliseconds to complete. When the number of free blocks in an SSD is smaller than a predetermined threshold, the valid pages in the victim blocks (i.e., to be erased) must be copied to a different free block before the victim blocks are erased to create new free blocks (thus free pages), which is called garbage collection. The GC process can significantly degrade both read and write performance by increasing the queuing times of the user requests.

Figure 2 shows the microscopic analysis of the average response times driven by the three realistic traces on the Intel DCS3700 200GB SSD. Before the evaluations, the SSD is filled with the written data. We can see that larger latencies are occurred due to the frequent GC operations. The large latencies are orders of magnitude than that in the normal state without GC operations. Moreover, previous studies also have similar findings [21] and shown that GC can render the SSD performance significantly variable and unpredictable in high performance computing and enterprise environments [15, 22]. The Solid State Storage Initiative of SNIA has initiated a project named "Understanding SSD Performance Project" [31], which has found that, along with the performance evaluations, the performance of flash-based SSDs also degrades dramatically. All these studies have revealed that GC progress has a significantly impact on the system performance.

2.2 Cache efficiency

Different storage devices, such as register, RAM, flash and disk, have different performance characteristics. The buffer cache is widely used in the storage systems to bridge the performance gap between two neighboring layers of the storage stack that have storage devices with very different performance characteristics. The efficiency of the buffer cache can be assessed by the average memory access time that depends on the *Hit_Time*, *Miss_Rate* and *Miss_Penalty*. The Average Memory Access Time (AMAT) is represented by the equation 1:

$$AMAT = Hit\_Time + Miss\_Rate * Miss\_Penalty \quad (1)$$

Each of the three factors can be optimized to reduce the AMAT. As shown in Figure 1, the on-board buffer cache is made up of the DRAM whose access time is consistent, meaning that the *Hit_Time* is consistent. The existing optimizations are trying to reduce the *Miss_Rate* as much as possible by reducing the number of the user I/O requests actually passed to the backend flash chips by exploiting access locality. They assume that the *Miss_Penalty* is also consistent because the direct access times from the flash chips are almost the same. While this assumption is valid for HDDs, it may not be fully validated for flash-based SSDs, as we discussed in Section 2.1. The access times of the flash chips can be quite different and varying depending on the presence or absence of the GC operations. The access time to the flash chip that is in the GC state will be much longer than that to one not in the GC state, by up to orders of magnitude. Thus we argue that *for the on-board buffer cache within flash-based SSDs, not only the* Miss_Rate*, but also the* Miss_Penalty *should be considered in the buffer cache management.*

Previous studies on the power-aware cache management scheme PB-LRU [37] and availability cache management schemes VDF and Shaper [32, 35] also found that under some conditions, the *Miss_Penalty* to the backend HDDs are different. For example, in PB-LRU [37], the access times on the active disks and sleep disks are different. In disk arrays with a faulty disk, the access times on the surviving disks and the faulty disk are also different. Thus, PB-LRU, Shaper and VDF take the *Miss_Penalty* as an important factor into their optimization considerations for the management of the buffer cache.

For the on-board buffer cache within the flash-based SSDs, the access times on the flash chips with GC and without GC are also different and more significantly so. However, the existing cache replacement schemes, such as BPLRU [19] and GC-ARM [10], only consider how to replace data blocks to improve the GC efficiency and are not aware of the un-

derlying GC operations within flash chips. Inspired by the previous studies on the power-aware cache management [37] and availability cache management [32, 35], we propose a GC-aware replacement scheme for buffer cache management, called GCaR, to improve the efficiency of the on-board buffer cache within the flash-based SSDs. By considering not only the *Miss_Rate*, but also the *Miss_Penalty*, the AMAT with GCaR will be shown to significantly improve.

3. GCaR

In this section, we first outline the main principles guiding the design of GCaR. Then we present a system overview of the GCaR, followed by a description of the the cache replacement and the buffer destaging algorithms in GCaR.

3.1 Design Principles

The design of GCaR aims to achieve high user performance and high GC efficiency, as explained below.

- **High user performance**. Most of the existing on-board buffer cache management schemes are designed to reduce the miss rate by reducing the user I/O requests delivered to the flash chips, oblivious of the GC activities in these chips. Our GCaR buffer cache management scheme is designed to be made garbage collection-aware in that it not only considers the miss rate, but also considers the GC-dependent miss penalty. As a result, GCaR will be able to reduce the user average response time by keeping as many of the expensive user I/O requests in the memory as possible, thus avoiding the long waiting time on the flash chips that are in the GC state.

- **High GC efficiency**. GCaR is designed to reduce the contention and interference between the user I/O requests and the GC-induced requests by keeping the interfering user requests in memory. This enables the GC operations to perform in the flash chips unimpeded to significantly improve the GC efficiency.

3.2 System Overview

Figure 3 shows a system overview of our proposed GCaR within the flash translation layer (FTL) of a flash-based SSD on the system I/O path. The host consists of the applications, file system and block device driver. The SSD device receives the read and write requests from the block device driver through the host interface. The on-board buffer cache management will check whether the requested data is in the buffer cache. If so, they will be serviced by the on-board buffer cache. Otherwise they will be issued to the backend flash chips.

GCaR has three main functional modules: The cache replacement module, the buffer destaging module and the GC scheduler module. The cache replacement and buffer destaging modules will interact with the GC scheduler module to determine what data should be replaced or destaged to reduce the miss penalty, as shown in Figure 3. The GC scheduler module has access to the real-time information about whether a given flash is in the GC state or not by its interaction with the FTL. There are three operations that are issued from the FTL to flash chips: read, write and erase. The GC process will invoke all the three operations within the involved flash chips, which significantly and adversely affects the user read and write performance. Thus the miss

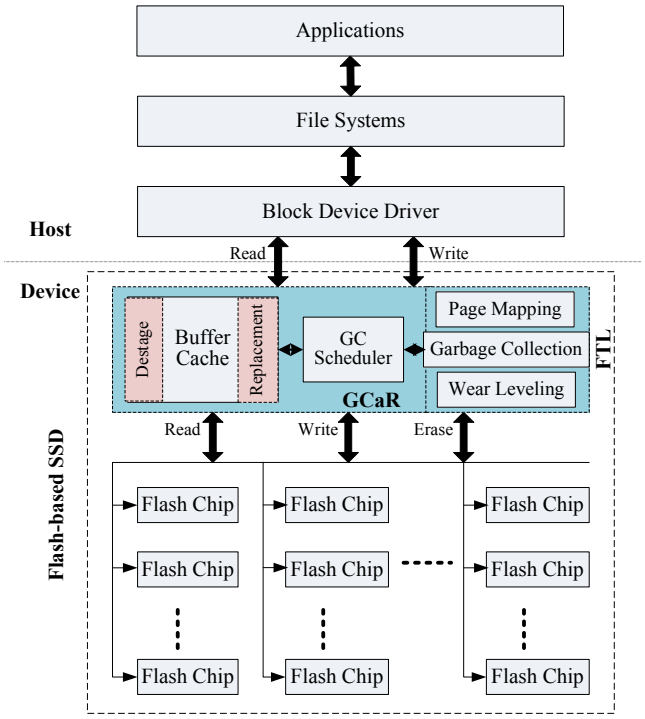


Figure 3: Overview of GCaR on the I/O path.

penalty to a flash chip in the GC state will be waiting for a much longer time than that to one not in the GC state and thus it is very expensive.

Different from the traditional buffer cache management schemes for flash-based SSDs, GCaR not only exploits the locality of workloads, but also takes the miss penalty as an important design factor in the cache replacement and buffer destage. Figure 4 shows the data structure in GCaR. We can see that all the data blocks in the buffer cache are associated with an LRU list to capture the workload locality. However, our scheme also can integrate with the data structures of schemes with other replacement algorithms, such as CFLRU [28] and LAMA [11]. Moreover, the conversion from the original cache algorithms to the GCaR-based algorithms will be smooth, because GcaR takes effect only when any of the flash chips are in the GC state. In other words, the buffer cache is managed by the original algorithms in the GC-free mode, and the GCaR policy becomes effective only when GC operations occur.

Figure 4 shows an example of the LRU scheme with flash chip 2 being in the GC state. Based on the LRU scheme, block D7 at the end of LRU list should be evicted from the buffer cache. However, by checking the logical-block-address to physical-block-address (LBA-PBA) mapping information, block D7 is in flash chip 2 that is in the GC state, the GCaR scheme will thus keep it longer in the buffer cache and instead evict block D5 to free buffer space. Furthermore and importantly, when a flash chip is in the GC state, the buffer space allocated to that flash chip is also increased to further reduce the miss rate because of the significantly increased miss penalty in a chip in the GC state. In the following two subsections, we will illustrate the cache replacement and buffer destaging schemes in GCaR in more detail.

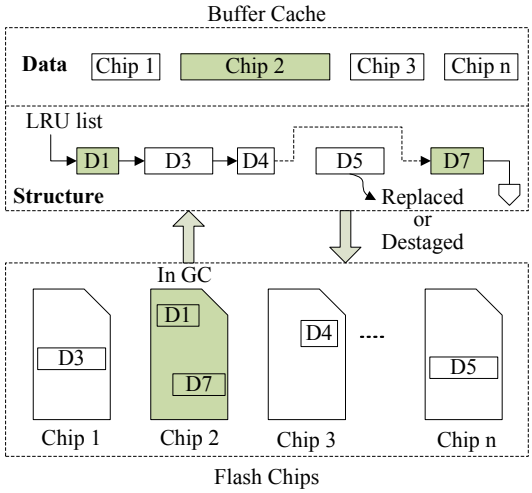


Figure 4: The data structure in GCaR.

3.3 Cache replacement

Traditional cache replacement algorithms by and large are designed to reduce the miss rate on the cache by exploiting request access locality while assuming that the penalty of each miss at the same level is of the same constant value. However, in flash-based SSDs with GC operations, the penalty of a miss to the missing data in the flash chips that are in the GC state are likely to be much more expensive than that of a miss to the flash chips not in the GC state. Therefore, from the aspect of an SSD device, the buffer cache performance should not be simply evaluated by the traditional metrics such as hit ratio or miss ratio. To address this issue, the cache replacement in GCaR takes not only the hit rate, but also the miss penalty as an important design parameter. A detailed description of the cache replacement algorithm in GCaR is given in Algorithm 1.

Upon receiving a read request, the GCaR scheme will check whether it hits the cache. If yes, the cached data will be returned. Otherwise, it will be fetched from the flash and the free cache space availability will be checked. If the cache is full, GCaR will replace data blocks to make free space for the new fetched data. The data blocks at the LRU tail will be checked to make sure that the evicted data blocks do not belong to the flash chips that are in the GC state. Once determined, the new fetched data from flash will be kept in the cache and the LRU list is updated. If the cache has free space, the new fetched data from flash will be written to the cache directly and the LRU list is updated. After the requested data is fetched from the flash to the cache, it will be returned to the upper layer.

Since the miss penalty on the flash chips that are in the GC state is orders of magnitude larger than the miss penalty on the flash chips not in the GC state, the data blocks belonging to the former will be given a higher priority of being retained in cache. When encountering a read miss and in need of replacing an existing data block in the cache, the data block from the tail of the LRU list not given the higher priority (i.e., not belonging the a chip in the GC state) will be selected as a victim data block. As shown in Figure 4, since data block D7 belongs to the flash chip 2 that is in the GC state, data block D5 belonging to flash chip n (not in the GC state) will be evicted from the cache to free buffer space.

Algorithm 1 Cache replacement algorithm in GCaR

```
1: Input: the user read requests R_1, R_2, ... R_i, ...
2: procedure GCaR_LRU_Replace(R_i)
3:     if R_i is LRU_List then
4:         /*A cache hit case*/
5:         Return_from_cache(R_i)
6:     else
7:         /*A cache miss case*/
8:         if Cache is full then
9:             repeat Get bottom item in LRU_List: L_x
10:                if L_x belongs to the GC-flash-chip then
11:                    x = x-1
12:                else
13:                    Delete(LRU_List, L_x)
14:                    Return_from_flash(R_i)
15:                    Update(LRU_List, R_i)
16:                    break
17:                end if
18:            until x == 0
19:        else
20:            Return_from_flash(R_i)
21:            Update(LRU_List, R_i)
22:        end if
23:    end if
24: end procedure
```

The GC operations are controlled and scheduled by the FTL within flash-based SSDs. The GC scheduler module in GCaR is responsible for getting the GC state information of each chip. Based on the GC state of a given chip, the GCaR can check whether a cache block to be replaced belongs to the flash chips that are in the GC state. In a real environment, the GC operations are triggered dynamically for each flash chips. Thus the priority of a data block in the cache also changes dynamically according to the underlying GC operations. Adding and updating a priority tag in the metadata for each of the cached data blocks can introduce a significant performance overhead. In our current design, the check for the GC state only happens when a data block needs to be replaced to avoid the metadata update overhead.

3.4 Buffer destaging

Buffer is an important storage pool to absorb and filter the write traffic from the upper level before going to the flash chips. For performance purposes, the buffer is usually set up with a write-back scheme. If the buffer space is unlimited, then the write performance will be the same as the memory speed. However, due to the cost and power issues, the buffer space is limited. The existing buffer management schemes concentrate on first merging random writes into sequential ones to improve both the write performance when destaging to the chips and the GC efficiency by reducing the GC-induced I/O traffic [19, 10]. However, these schemes, being unaware of the GC state of the involved chips, are only beneficial when writing to chips not in the GC state and are unable to improve the user write performance during GC operations.

The buffer destaging scheme in GCaR takes the GC operations into consideration by first detaging data blocks belonging to the flash chips that are not in the GC state. To make free space for the subsequent write data on the flash

Algorithm 2 Buffer destage algorithm in GCaR

```
1: Input: the user write requests W_1, W_2, ... W_i, ...
2: procedure GCaR_LRU_Destage(W_i)
3:     if W_i is LRU_List then
4:         /*A buffer hit case*/
5:         Write_to_buffer(W_i)
6:     else
7:         /*A buffer miss case*/
8:         if Buffer is full then
9:             repeat Get bottom item in LRU_List: L_x
10:                if L_x belongs to the GC-flash-chip then
11:                    x = x-1
12:                else
13:                    Write_to_flash(D_x)
14:                    Delete(LRU_List, L_x)
15:                    Write_to_buffer(W_i)
16:                    Update(LRU_List, W_i)
17:                    break
18:                end if
19:            until x == 0
20:        else
21:            Write_to_buffer(W_i)
22:            Update(LRU_List, W_i)
23:        end if
24:    end if
25: end procedure
```

chips that are in the GC state, the GCaR scheme will allocate much more buffer space to temporally store all the write data blocks belonging to the flash chips that are in the GC state. Once a flash chip completes its GC process, the corresponding data blocks belonging to that flash chip will be destaged. By avoiding issuing the user write traffic to the flash chips that are in the GC state, both the GC efficiency and the user write performance can be improved.

Algorithm 2 provides a detailed description of the buffer destaging scheme in GCaR. Once a write request arrives, the GCaR scheme will check whether it hits the buffer. If yes, the buffered data will be updated. Otherwise, the buffer free space availability will be checked. If the buffer is full, GCaR will destage data blocks to make free space for the new write data. The data blocks at the LRU tail will be checked to make sure that the destaged data blocks do not belong to the flash chips that are in the GC state. Once determined, the evicted data blocks will be written to the flash chips that are not in the GC state. The new write data will be kept in the buffer and the LRU list is updated. If the buffer has free space, the new write data will be written to the buffer directly and the LRU list is updated.

4. PERFORMANCE EVALUATIONS

In this section, we first describe the experimental setup and methodology. Then we evaluate the performance of GCaR through both benchmark-driven and trace-driven evaluations.

4.1 Experimental setup and methodology

To evaluate the efficiency of our proposed GCaR scheme, we have implemented a prototype of the GCaR scheme by integrating it into an open-source SSD simulator developed by Microsoft Research (MSR) [1]. The MSR SSD simulator, an extension of Disksim from the Parallel Data Lab of CMU [3], has been released to the public and widely used to evaluate the performance of the SSD-based storage systems in many studies [1, 22, 34]. In this paper, we extend the original Disksim and the MSR SSD simulator to implement our proposed GCaR scheme. The values of the SSD specific parameters used in the simulator are shown in Table 2.

Table 2: The default SSD model parameters.

Parameter	Value
Total Capacity	28GB
Reserved Free Blocks	15%
Minimum Free Blocks	5%
Cleaning Policy	Greedy
Flash Chip Elements	7
Planes Per Package	8
Blocks Per Plane	1024
Pages Per Block	64
Page Size	4KB
Page Read Latency	25 us
Page Write Latency	200 us
Block Erase Latency	1.5 ms

We use both the synthetic traces and the realistic enterprise-scale workloads to study the performance impact of the different buffer cache schemes. The synthetic workloads allow us to flexibly vary parameters such as request size, inter-arrival time of requests, read access probability, and sequentiality. The default values of the parameters that we use in our experiments are shown in Table 3. The three realistic enterprise-scale workloads were collected from the Microsoft Cambridge Research [2]. The main workload parameters of these traces are summarized in Table 4.

Table 3: The default parameters of synthetic traces.

Parameter	Value
Request size	32KB
Inter-arrival time	4ms
Probability of sequential access	0
Probability of read access	0.2

In the evaluations, we have integrated our proposed GCaR scheme with the LRU, CFLRU [28], and BPLRU [19] schemes, with the resulting GCaR-based schemes being labeled GCaR-LRU, GCaR-CFLRU, and GCaR-BPLRU respectively. We compare the performances of these GCaR based schemes with LRU, CFLRU, and BPLRU schemes in terms of the average response time. The LRU scheme is the baseline for all the other schemes. The main features of the CFLRU and BPLRU schemes are summarized below.

- **CFLRU** (Clean First LRU) [28] is a buffer cache management algorithm for flash storage. It was proposed to exploit the asymmetric performance of flash memory read and write operations. It attempts to choose a clean page rather than a dirty one as a victim because writing cost of the latter is much more expensive.

- **BPLRU** (Block Padding LRU) [19] aims to improve the random-write performance of SSDs. It combines three key techniques, block-level LRU management, page padding, and LRU compensation to convert random write requests to sequential ones.

Table 4: The realistic workload characteristics.

Traces	Write Ratio	Total I/Os	Average Request Size (KB)
Financial1	76.8%	5,334,981	8.5
Prn_0	89.2%	5,585,886	22.2
Prxy_0	97.1%	5,000,000	6.8

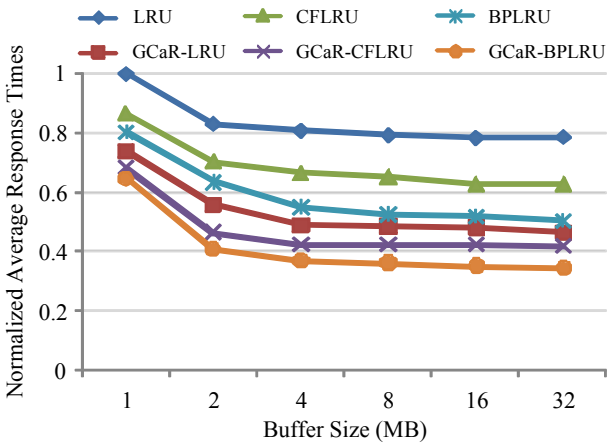


Figure 5: The average response times, normalized to that of the LRU scheme with a 1MB buffer, under the synthetic workloads.

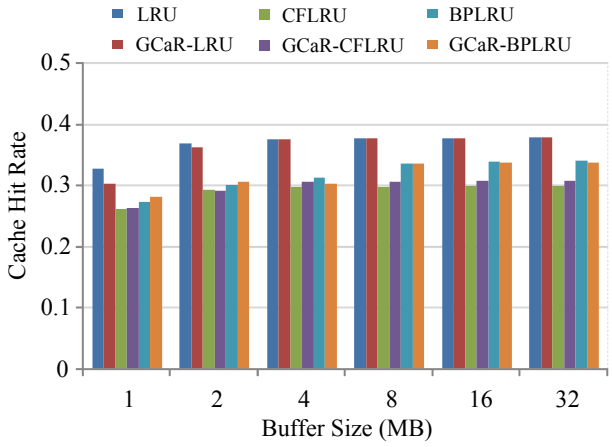


Figure 6: The cache hit rate of the different cache schemes as a function of the buffer size, driven by the synthetic workloads.

4.2 Performance results

Synthetic workloads: Figure 5 shows the average response times, normalized to that of the baseline with a buffer size of 1 MB, of the different schemes as a function of the buffer size, driven by the synthetic workloads. We can see that GCaR-based schemes reduce the average response times by 35.9%, 32.1%, and 30.6% on average compared with the LRU, CFLRU, and BPLRU schemes respectively. It is clear that, by avoiding issuing GC-conflicting user requests to the flash chips in the GC state, all the GCaR-based schemes are able to significantly reduce user response times. The results indicate that GC operations have a significant impact on the user response times. On the other hand, as expected, the average response times decrease as the buffer size increases. When the buffer size is larger than 4 MB, the response times become stable. However, the improvements of the GCaR-based schemes over their original ones are also consistent. For example, GCaR-LRU reduces the average response time of the LRU scheme by 40.7% when the buffer size is 32 MB. With a larger buffer size, GCaR-LRU scheme can keep data blocks belonging to the flash chip in the GC state in memory without scarifying too much data access locality of other blocks.

To better understand the reasons behind the above results, we examine the cache hit rate and GC count measures collected in the simulation study, as shown in Figure 6 and Figure 7. Figure 6 shows that the LRU scheme has the highest cache hit rate under the different buffer sizes, though the differences are not significant. The reason is obvious. Among all the schemes, the LRU scheme is the only one that tries exclusively to exploit the access locality to improve the cache hit rate. The other schemes try to improve the GC efficiency or reduce the miss penalty. Thus, the access locality may be sacrificed to some extend. However, their designs are based on the LRU scheme, the overall hit rates are not reduced significantly. On the other hand, the cache hit rates are not consistent with the average access times, as shown in Figure 5. This phenomenon further implies that cache hit rate is not the only influential factor affecting the overall performance. It is also the reason why cache hit rate should not be the only design objective of an effective cache scheme for flash storage.

Figure 7 shows the GC count of the different cache schemes as a function of the buffer size, driven by the synthetic workloads. First, compared with the LRU scheme, both the CFLRU and BPLRU schemes reduce the GC count significantly under the different buffer sizes, especially when the buffer size is small. BPLRU reduces the total GC count of the LRU scheme by up to 11.2%. It confirms that reducing GC count can significantly improve the overall system performance. In other words, the GC operations have a significant impact on the overall system performance. Second, it is interesting to notice that, while the GCaR-based schemes have similar GC counts to their non-GCaR counterparts, the former's average response times are much better than the latter's. The reason is that the GCaR-based schemes do not try to further improve the GC efficiency, but instead try to reduce the interference between the user I/O requests and GC-induced I/O requests. Although the GC efficiency is not further improved by the GCaR-based schemes, the responsiveness to the user I/O requests has been significantly improved. The overall system performance in terms of the average response time is also improved accordingly. Third, the GCaR-based schemes achieve their superiority in overall performance without sacrificing advantages of their non-GCaR counterparts, as shown in Figure 5. In other words, the GCaR scheme is orthogonal and complementary to these existing schemes. This is because the GCaR scheme only

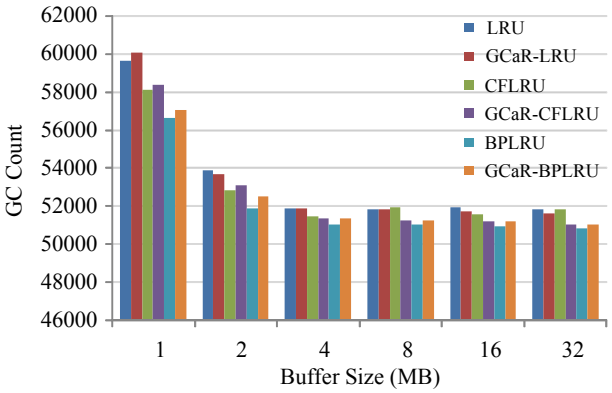


Figure 7: The GC count of the different cache schemes as a function of the buffer size, driven by the synthetic workloads.

takes the GC activities into the buffer cache design, without changing the original cache algorithm in the normal operational (non-GC) state. The extra resource overhead is minimal. Thus the other schemes can be easily extended to be the GCaR enhanced schemes to improve system performance.

Real application traces: Figure 8 shows the average response times of the different schemes driven by the three traces when the buffer size is 1 MB. The GCaR-based cache schemes perform much better than the their non-GCaR counterparts, especially for the Prn_0 and Prxy_0 workloads for which the improvements are on average 25.9% and 15.0% respectively. The reason is that for the Prn_0 and Prxy_0 workloads, the write ratio is high and the request size is large, which means that much more data is written to the flash chips than workloads with low write ratios and small request sizes. Also note that the GC operations are much more frequent than those in the Financial workload. As a result, the GCaR-based schemes, capable of issuing much fewer GC-conflicting user requests to the chips in the GC state, are able to significantly reduce the overall user response times.

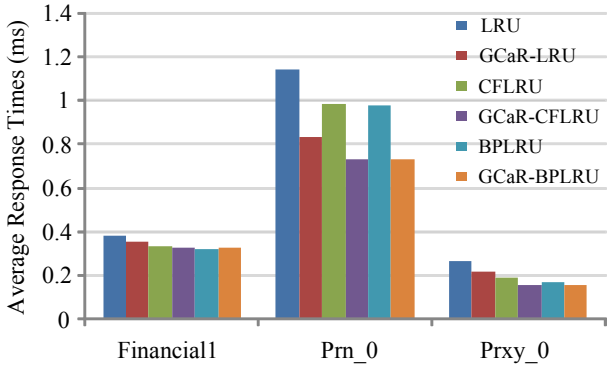


Figure 8: The average response times under the realistic trace-driven evaluations.

We now examine the cache hit rates of the different schemes driven by the three traces, normalized to that of the baseline with the LRU scheme. Figure 9 shows that, similar to the case of synthetic workloads, the LRU scheme outperforms

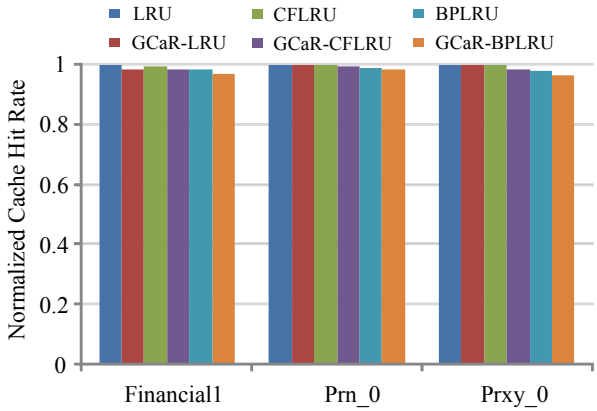


Figure 9: The cache hit rates, normalized to that of the baseline scheme (LRU), under the trace-driven evaluations.

all the schemes in the cache hit rate measure, although its advantage is only marginal. On the other hand, compared with the results of the average response times in Figure 8, the cache hit rate is not consistent with the average response times. The reason is obvious in that the miss penalty is not consistent for flash chips, e.g., much higher penalty on chips in the GC state than on non-GC chips. On the other hand, we also collect the GC count during the experiments. Figure 10 shows that request access delays caused by the GC operations are an influential part of the average response times. Thus, with the real application workloads we obtain consistent results with and the same conclusion as with the synthetic workloads.

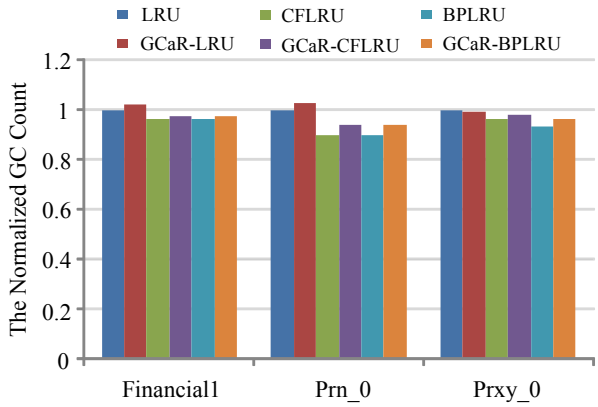


Figure 10: The GC counts, normalized to that of the baseline scheme (LRU), under the trace-driven evaluations.

4.3 Sensitivity study

The cache performance in the presence of GC operations is affected by various factors, such as the high/low watermark (defined below) and the request size. In order to evaluate their impact on the system performance, we carry out several sensitivity studies driven by the synthetic workloads. Here we only report results that compare GCaR-CFLRU with CFLRU for simplicity, since all the other schemes share the same trend.

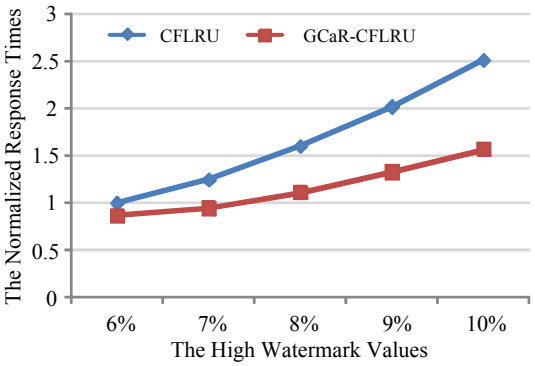


Figure 11: The average response time, normalized to that of the baseline with CFLRU scheme under 6% high watermark, as a function of the high watermark value, driven by the synthetic workloads.

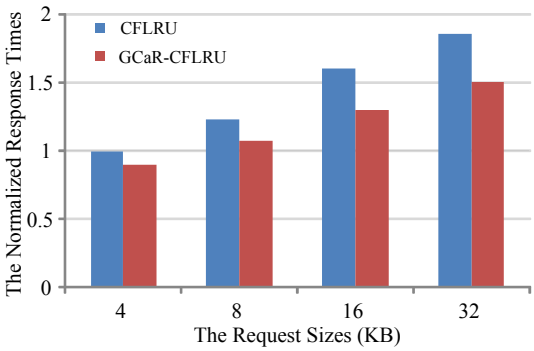


Figure 12: The average response time, normalized to that of the baseline with CFLRU scheme with a request size of 4KB, as a function of the request size, driven by the synthetic workloads. *Note: CFLRU under 4KB request size is the baseline scheme.*

High watermark. The high watermark is the threshold on the number of available free blocks in a flash chip, below which a GC operation within chip will be triggered. It is set by default at 5% of the total number of blocks in a chip, as shown in Table 2. Figure 11 shows the average response time, normalized to that of the baseline with CFLRU scheme under 6% high watermark, as a function of the high watermark values driven by the synthetic workloads. It shows that the overall response time increases with high watermark value. The reason is that with a higher watermark, the GC operations will be triggered more frequently and thus more blocks will be erased to free up the invalid blocks, which in turn increases the interference between the user IO requests and GC-induced IO requests. However, the GCaR scheme is shown to be much less affected by the change in the high watermark value than the CFLRU scheme. The reason is that the GCaR scheme tries to alleviate the effect of GC operations on the user performance. Thus the GC operation overhead on the system performance in the GCaR-based scheme is much less than the CFLRU scheme.

Request size. Figure 12 shows the average response time, normalized to that of the baseline with the CFLRU scheme with a request size of 4 KB, as a function of the request size, driven by the synthetic workloads. It shows that as request size increases, so does the response time. The reasons are two-fold. First, with a larger request size, the transfer time is also longer since the response time is linearly increased with the increased request size [25]. Second, with a larger request size, the overall written data size is increased. Thus the available free space is also decreased much faster, which causes the GC operations to be triggered more frequently. Nevertheless, the GCaR-CFLRU scheme consistently outperforms the CFLRU scheme in overall response time regardless of the request size.

5. RELATED WORK

Buffer cache is one of the most important and effective performance optimizations in storage systems [7]. Its management is well studied in the literature and a large number of cache algorithms have been proposed. Most of them aim to improve the HDD-based storage systems. However, due to the different performance characteristics, the cache schemes for HDDs are not suitable for flash-based SSDs [10, 19, 28].

Based on the asymmetric performance of read and write operations of flash memory, CFLRU [28] divides the LRU list into the working region and the clean-first region, and adopts a policy that evicts clean pages preferentially in the clean-first region until the number of pages hit in the working region reaches a suitable level. Their experiments show that CFLRU is able to reduce the average replacement cost by 26% in the buffer cache compared to the traditional LRU algorithm. CFLRU reduces the number of writes by trading off the number of reads. However, this is irrelevant when only write requests are involved. The Flash Aware Buffer policy (FAB) [14] groups the data pages belonging to the same erasable block together and manages with an LRU list. A group is moved to the beginning of the list when a buffer in the group is read or updated, or a new buffer is added to the group. When replacing a cache line out of the memory, a group that has the largest number of data blocks is selected as victim. All the dirty data blocks in the victim group are flushed, and all the clean data blocks in it are discarded. The main use of FAB is in portable media player applications in which the majority of write requests are sequential. However, both CFLRU and FAB are not useful for enhancing random write performance.

To address the random-write performance issue, BPLRU [19] is proposed. It manages a LRU list in the unit of NAND-flash erasable block and pads a log block with some clean pages from the data block to reduce the number of full merges. However, it fails to consider the cost of extra padding operations, which can degrade the performance significantly and become a performance bottleneck particularly when the buffer is much smaller than the working set. PUDLRU (Predicted average Update Distance LRU) [9] partly solves this problem by carefully considering the average update distance and the fullness of the buffered blocks and then selecting one as victim. The extended GC-ARM [10] dynamically destages either contiguous pages in a block as a whole or a single page from the write buffer based on the benefit to improve GC efficiency.

Some studies also try to dynamically allocate buffer space between mapping table and user data to improve system performance. Hyotaek et al. [29] propose an adaptive RAM partitioning scheme for SSDs, which adaptively tunes the

Table 5: The related studies with GCaR and their differences.

Schemes	Devices	Objective	Comments
PB-LRU [37]	HDDs	Power efficiency	Miss penalties on the active and sleeping disks are different
VDF [32]	HDDs	Availability	Miss penalties on the active and faulty disks are different
BPLRU [19]	SSDs	Performance	Improving GC efficiency proactively but unaware of run-time GCs
GC-ARM [10]	SSDs	Perf. & Reliability	Improving GC efficiency proactively but unaware of run-time GCs
GCaR	SSDs	Performance	Miss penalties in run-time GC-active and GC-inactive chips are different

space ratio of the data buffer and mapping table according to the workload characteristics. Hu et al. propose PASS [12], which judiciously and actively writes an appropriate amount of the buffered data back to flash by exploiting the light-traffic periods of workloads and the parallelism inside SSDs to avoid real-time flash operations and buffer overflow during bursty-traffic periods. These space management schemes only adaptively change the space allocation ratio between the mapping table and the write data according the workload characteristics. Thus the cache management algorithms for the user data are not affected.

All these schemes work well in the normal state when there are no GC operations in process within the flash chips. Whenever a GC operation is triggered on a particular flash chip, most of the assumptions made for these schemes are no longer valid. For example, the response time of a read request on an ongoing GC-based flash chip is much longer than that of a write request on a flash chip without GC operations. Thus, the CFLRU scheme, for example, will lose its key efficiency. Other optimizations, such as BPLRU, PUDLRU, and GC-ARM, only proactively improve the GC efficiency by reducing the amount of random write data blocks. Their main and shared objective is to improve the GC efficiency by optimizing the GC workflow. However, GCaR improves the GC efficiency by avoiding the contention between user requests and GC-induced requests. Dynamic allocation scheme [30] is trying to avoid the resource contention between the user I/O requests and the GC-induced requests. However, dynamic allocation needs extra memory to store the mapping information and changes the data layout on flash chips, which may affect the GC efficiency. Moreover, our GCaR scheme works not only for write requests, but also for read requests. Thus, our proposed GCaR scheme is orthogonal to and can be easily incorporated into the most existing cache management algorithms to further improve system performance.

Since the main objective of most cache algorithms proposed for HDD-based storage systems is performance, once the objective is changed, their management is also changed. For example, PB-LRU [37] takes the power saving rather performance as the design objective and dynamically allocates different buffer space for different disks based on the power state, because the miss penalties to the active disk and sleeping disk are different. VDF and Shaper [32, 35] are cache management schemes for disk arrays under disk failure. In this condition, the miss penalties to the active disks and the faulty disk are different. Thus they give higher priority for the data blocks belonging to the faulty disk in buffer cache to alleviate the disk contention between the user I/O requests and the reconstruction-induced I/O requests. Our proposed GCaR is inspired by the PB-LRU, VDF and Shaper schemes, but works for the flash-based SSDs with different design objectives. Table 5 presents a summary comparison among the related studies and GCaR.

6. CONCLUSION

With the widening gap between the speeds of the processor/memory and the HDDs, the I/O access latency has become the system performance wall. Flash-based SSDs have emerged to be a promising technology to bridge this gap to reduce the access latency in the high performance computing environment and enterprise data centers. However, GC is a significant performance concern for flash-based storage systems in enterprise and HPC environments. In this paper, we propose a Garbage Collection aware Replacement policy, called GCaR, to improve the performance of flash-based SSDs. The basic idea is to give higher priority to caching the data blocks belonging to the flash chips that are in the GC state. This substantially lessens the contentions between the user I/O operations and the GC-induced I/O operations. We have integrated GCaR into the SSD extended Disksim simulator and conducted extensive evaluations. The performance results show that GCaR can significantly improve the storage performance by substantially reducing the aforementioned contentions.

Our proposed GCaR is an ongoing research project and we are currently exploring several directions for the future work. First, we will build a hardware platform to incorporate our GCaR scheme within the FTL design in flash-based SSDs and conduct many more experiments on the prototype. Second, we will extend the GCaR scheme into the semi-preemptive garbage collector [22] to increase the GC efficiency. By improving the efficiency of both the GC-induced I/O operations and user I/O requests, the overall system performance will be further improved.

7. ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China under Grant No. 61100033, No. 61472336 and No. 61402385, US NSF under Grant No. NSF-CNS-1116606 and NSF-CNS-1016609, Fundamental Research Funds for the Central Universities (No. 20720140515).

8. REFERENCES

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy. Design Tradeoffs for SSD Performance. In *Proceedings of the 2008 USENIX Annual Technical Conference (USENIX'08)*, Boston, MA, Jun. 2008.

[2] Block I/O Traces in SNIA. http://iotta.snia.org/tracetypes/3.

[3] J. Bucy, J. S. Schindler, S. W. Schlosser, and G. R. Ganger. The DiskSim Simulation Environment Version 4.0 Reference Manual. May 2008.

[4] A. M. Caulfield, J. Coburn, T. Mollov, A. De, A. Akel, J. He, A. Jagatheesan, R. K. Gupta, A. Snavely, and S. Swanson. Understanding the Impact of Emerging Non-Volatile Memories on

High-Performance, IO-Intensive Computing. In *Proceedings of the 2010 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, New Orleans, LA, Nov. 2010.

[5] F. Chen, D. A. Koufaty, and X. Zhang. Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems. In *Proceedings of the 25th International Conference on Supercomputing (ICS'11)*, Tucson, AZ, USA, Jun. 2011.

[6] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09)*, Washington, DC, USA, Mar. 2009.

[7] W. W. S. Hsu and A. J. Smith. The Performance Effect of I/O Optimizations and Disk Improvements. *IBM Journal Research and Development*, 48(2):255–289, 2004.

[8] J. Hu, H. Jiang, and Prakash. Understanding Performance Anomalies of SSDs and Their Impact in Enterprise Application Environment. In *Proceedings of the 12th Joint ACM SIGMETRICS/Performance Conference (SIGMETRICS'12)*, London, UK, Jun. 2012.

[9] J. Hu, H. Jiang, L. Tian, and L. Xu. PUD-LRU: An Erase-Efficient Write Buffer Management Algorithm for Flash Memory SSD. In *Proceedings of The 18th Annual Meeting of the IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'10)*, Aug. 2010.

[10] J. Hu, H. Jiang, L. Tian, and L. Xu. GC-ARM: Garbage Collection-Aware RAM Management for Flash based Solid State Drives. In *Proceedings of the 7th IEEE International Conference on Networking, Architecture, and Storage (NAS'12)*, Jun. 2012.

[11] X. Hu, X. Wang, Y. Li, L. Zhou, Y. Luo, C. Ding, S. Jiang, and Z. Wang. LAMA: Optimized Locality-aware Memory Allocation for Key-value Cache. In *Proceedings of 2015 USENIX Annual Technical Conference (USENIX'15)*, Santa Clara, CA, USA, Jun. 2015.

[12] Y. Hu, H. Jiang, D. Feng, H. Luo, and L. Tian. PASS: A Proactive and Adaptive SSD Buffer Scheme for Data-Intensive Workloads. In *Proceedings of The 10th IEEE International Conference on Networking, Architecture, and Storage (NAS'15)*, Aug. 2015.

[13] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang. Performance Impact and Interplay of SSD Parallelism through Advanced Commands, Allocation Strategy and Data Granularity. In *Proceedings of the 25th International Conference on Supercomputing (ICS'11)*, Tucson, AZ, USA, Jun. 2011.

[14] H. Jo, J. Kang, S. Park, J. Kim, and J. Lee. FAB: Flash-Aware Buffer Management Policy for Portable Media Players. *IEEE Transactions on Consumer Electronics*, 52(2):485–493, 2006.

[15] M. Jung, W. Choi, J. Shalf, and M. Kandemir. Triple-a: A non-ssd based autonomic all-flash array for scalable high performance computing storage systems. In *Proceedings of 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*, Salt Lake City, Utah, Mar. 2014.

[16] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. Kandemir. Hios: A host interface i/o scheduler for solid state disks. In *Proceedings of the 41st International Symposium on Computer Architecture (ISCA'14)*, Minneapolis, MN, Jun. 2014.

[17] M. Jung and M. Kandemir. Sprinkler: Maximizing resource utilization in many-chip solid state disks. In *Proceedings of the 20th IEEE International Symposium On High Performance Computer Architecture (HPCA'14)*, Orlando, Florida, USA, Feb. 2014.

[18] M. Jung, R. Prabhakar, and M. Kandemir. Taking Garbage Collection Overheads off the Critical Path in SSDs. In *Proceedings of the ACM/IFIP/USENIX 13th International Conference on Middleware (Middleware'12)*, Montreal, Quebec, Canada, Dec. 2012.

[19] 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 (FAST'08)*, San Jose, CA, Feb. 2008.

[20] Y. Kim, R. Gunasekaran, G. M. Shipman, D. A. Dillow, Z. Zhang, and B. W. Settlemyer. Workload Characterization of a Leadership Class Storage. In *Proceedings of the 5th Petascale Data Storage Workshop Supercomputing (PDSW'10)*, New Orleans, LA, Nov. 2010.

[21] Y. Kim, S. Oral, G. M. Shipman, J. Lee, D. A. Dillow, and F. Wang. Harmonia: A Globally Coordinated Garbage Collector for Arrays of Solid-state Drives. In *Proceedings of the 27th IEEE Symposium on Mass Storage Systems and Technologies (MSST'11)*, Denver, CO, May 2011.

[22] J. Lee, Y. Kim, G. Shipman, S. Oral, F. Wang, and J. Kim. A Semi-Preemptive Garbage Collector for Solid State Drives. In *Proceedings of the 2011 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'11)*, Austin, TX, Apr. 2011.

[23] B. Mao, H. Jiang, S. Wu, Y. Fu, and L. Tian. Sar: Ssd assisted restore optimization for deduplication-based storage systems in the cloud. In *Proceedings of 7th IEEE International Conference on Networking, Architecture, and Storage (NAS'12)*, Xiamen, China, Jun. 2012.

[24] B. Mao, H. Jiang, S. Wu, L. Tian, D. Feng, J. Chen, and L. Zeng. HPDA: A Hybrid Parity-based Disk Array for Enhanced Performance and Reliability. *ACM Transactions on Storage*, 8(1):1–20, 2012.

[25] B. Mao and S. Wu. Exploiting Request Characteristics and Internal Parallelism to Improve SSD Performance. In *Proceedings of the 33rd IEEE International Conference on Computer Design (ICCD'15)*, Oct. 2015.

[26] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating Server Storage to SSDs: Analysis of Tradeoffs. In *Proceedings of the 4th European Conference on Computer Systems*

(EuroSys'09), Nuremberg, Germany, Mar. 2009.

[27] S. Oral, F. Wang, D. A. Dillow, G. M. Shipman, and R.Miller. Efficient Object Storage Journaling in a Distributed Parallel Filesystem. In *Proceedings of the 8th USENIX Conference on File and Storage Technology (FAST'10)*, San Jose, CA, Feb. 2010.

[28] S. Park, D. Jung, J. Kang, J. 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 (CASES'06)*, Oct. 2006.

[29] H. Shim, B. Seo, J. Kim, and S. Maeng. An Adaptive Partitioning Scheme for DRAM-based Cache in Solid State Drives. In *Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST'10)*, May 2010.

[30] A. Tavakkol, M. Arjomand, and H. Sarbazi-Azad. Unleashing the potentials of dynamism for page allocation strategies in ssds. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'14)*, Austin, TX, USA, Jun. 2014.

[31] Understanding SSD Performance Project of SNIA. http://www.snia.org/forums/sssi/pts.

[32] S. Wan, Q. Cao, J. Huang, S. Li, X. Li, S. Zhan, L. Yu, C. Xie, and X. He. Victim Disk First: An Asymmetric Cache to Boost the Performance of Disk Arrays under Faulty Conditions. In *Proceedings of the 2011 USENIX Annual Technical Conference (USENIX'11)*, Portland, OR, Jun. 2011.

[33] W. Wang, T. Xie, and D. Zhou. Understanding the impact of threshold voltage on mlc flash memory performance and reliability. In *Proceedings of the 2014 International Conference on Supercomputing (ICS'14)*, Muenchen, Germany, Jun. 2014.

[34] G. Wu and B. He. Reducing SSD Read Latency via NAND Flash Program and Erase Suspension. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*, San Jose, CA, Feb. 2012.

[35] S. Wu, B. Mao, D. Feng, and J. Chen. Availability-Aware Cache Management with Improved RAID Reconstruction Performance. In *Proceedings of 13th IEEE International Conference on Computational Science and Engineering (CSE'10)*, Hong Kong, Dec. 2010.

[36] X. Zhang, J. Li, H. Wang, K. Zhao, and T. Zhang. Reducing Solid-State Storage Device Write Stress through Opportunistic In-place Delta Compression. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16)*, Feb. 2016.

[37] Q. Zhu, A. Shankar, and Y. Zhou. PB-LRU: A Self-tuning Power Aware Storage Cache Replacement Algorithm for Conserving Disk Energy. In *Proceedings of the 18th Annual International Conference on Supercomputing (ICS'04)*, Malo, France, Jun. 2004.