LAWC: Optimizing Write Cache Using Layout-Aware I/O Scheduling for All Flash Storage

Kalidas Ganesh, Youngjae Kim, Monobrata Debnath, Sungyong Park, and Junghee Lee

Abstract—Flash memory-based SSD-RAIDs are swiftly replacing conventional hard disk drives by exhibiting improved performance and stability, especially in I/O-intensive environments. However, the variations in latency and throughput occurring due to uncoordinated internal garbage collection cripples further boosting of performance. In addition, the unwanted variations in each SSD can influence the overall performance of the entire flash storage adversely. This performance bottleneck can be essentially reduced by an internal write cache in the RAID controller designed prudently by considering the crucial device characteristics. The state-of-the-art cache write for the RAID controller fails to incorporate device characteristics of flash memory-based SSDs and mitigates the performance gain. In this paper, we propose a novel cache design namely Layout-Aware Write Cache (LAWC) to overcome the performance barrier inculcated by independent garbage collections. LAWC implements (i) improved I/O scheduling for logically partitioned write caches, (ii) a destage write synchronization mechanism to allow individual write caches to flush write blocks into the SSD array in a coordinated manner, and (iii) a two-level hybrid cache algorithm utilizing small front level cache for the improved write cache efficiency. LAWC shows significant reduction in response time by 82.39 percent on RAID-0 and 68.51 percent on RAID-5 types of SSDs when compared with state-of-the-art write cache algorithms.

Index Terms—Flash memory, I/O scheduling, RAID, solid-state drive, storage system, write cache

1 Introduction

With the advent of Big Data, the amount of data produced every year has exponentially increased. For instance, many science facilities produce a vast amount of experimental and simulation data: several U.S. Department of Energy (DOE) leadership-computing facilities, such as the Oak Ridge Leadership Computing Facility (OLCF) [1], the Argonne Leadership Computing Facility (ALCF) [2], and the National Energy Research Scientific Computing (NERSC) [3] generate hundreds of petabytes per year of simulation data and are projected to generate in excess of 1 exabyte per year by 2018 [4]. In fact, according to the Big Data and Scientific Discovery report from the DOE, Office of Science, Office of Advanced Scientific Computing Research (ASCR) [5], some of the scientific data challenges are the worsening input/output (I/O) bottleneck.

To accommodate such big data, organizations will continue to deploy larger storage infrastructures which need to be well provisioned to meet the expected data growth rate. The storage system can employ multiple disk drives to build arrays of disk drives. Redundant array of inexpensive disks (RAID) were introduced to increase the performance and reliability of disk drive systems [6]. For example, a peta-scale storage system for Titan at the Oak Ridge National Laboratory uses 8+2 RAID-6 configuration for each volume group [7], [8]. RAID provides parallelism of I/O operations by combining multiple inexpensive disks, thereby achieving higher performance and robustness. RAID has become the de facto standard for building high performance and robust storage systems.

Traditional storage systems use hard disk drives which offered price efficiency. However, nowadays the solid-state drive (SSD) storage market is continually growing towards high performance storage at low cost per byte of data. Unlike in-place update operations in HDDs, SSDs incorporate software to allow out-of-place update operations and to map sectors from the host into their current locations in the SSDs [9], [10], [11]. This out-of-place update operation eventually requires a sweep of storage area to find stale data and consolidate active data in order to create free space. This process, known as garbage collection (GC), can significantly increase the service time of incoming requests. When SSDs are configured in RAID, the performance variability of individual SSDs becomes a major concern, because the overall performance of the SSD-based RAID could be limited by the slowest SSD [12], [13], [14], [15].

In this paper, we propose a novel Layout-Aware Write Cache (LAWC) to alleviate the unstable performance concern in SSD-based RAID systems. LAWC can mitigate the performance degradation by uncoordinated GC operations of SSDs in the array by performing coordinated destage operations in the write buffer on the RAID controller. LAWC organizes multiple sub-buffers, instead of a single global buffer, as many as the number of SSDs, and the I/O

- *K. Ganesh, M. Debnath, and J. Lee are with the Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX 78249. E-mail: dyk567@my.utsa.edu, {monobrata.debnath, junghee.lee}@utsa.edu.*
- *Y. Kim and S. Park are with the Department of Computer Science and Engineering, Sogang University, Seoul 04107, South Korea. E-mail: {youkim, parksy}@sogang.ac.kr.*

Manuscript received 22 June 2016; revised 20 Mar. 2017; accepted 30 Mar. 2017. Date of publication 22 May 2017; date of current version 16 Oct. 2017. (Corresponding author: Youngjae Kim.)
Recommended for acceptance by R. F DeMara.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2017.2707408

scheduler synchronizes the requests in their corresponding queues to increase the chance that multiple SSDs trigger GC at the same time, resulting in improved I/O performance. LAWC also implements a pre-parity-computation technique for reliable RAID configuration with parity computation, which will help flexible I/O scheduling. Moreover, to alleviate the overhead of the pre-parity-computation, we enhance LAWC by adding a small cache in front of LAWC.

LAWC is developed to offer comparable I/O performance to Harmonia- a Globally Coordinated Garbage Collection(GGC) [12], [13] without incurring an expense of extension of storage protocols such as SATA and SCSI for controlling the additional communication capabilities on both RAID controller and individual SSDs. LAWC offers significant performance improvement compared with the state-of-art write cache algorithm for HDD-based RAID, Wise Ordering of Writes (WOW) [16], by optimizing the write cache architecture of the RAID controller.

The contributions of this paper are summarized as below.

- This paper proposes independent write caches and pre-parity-computation technique that allow flexible I/O scheduling of destage writes, which contributes to mitigating the adverse impact of performance variation in SSDs in their arrays caused by GCs of individual SSDs.
- The destage write synchronization technique is proposed, which delays destage write operations of write caches until all write caches are ready to destage at once. In order to compensate the performance penalty by pre-parity computation, the two-level hybrid caching technique is proposed, which serves as a higher-level cache for independent write caches. It mitigates the overhead incurred by the pre-parity-computation and synchronization techniques for less I/O intensive workloads.
- Our experiments show that LAWC improves the average number of synchronized GCs by up to 53.1 times compared to WOW. Moreover, LAWC achieves huge performance improvement of 82.39 percent in average response time compared to WOW for heavy write dominant workloads. For non-intensive workloads, LAWC offers comparable performance to WOW.

The remainder of this paper is organized as follows. We first present an overview of the material and technology in Section 2 followed by motivation in Section 2.2. Section 3 presents our proposed layout-aware write cache design and implementation with performance optimization techniques such as destage write synchronization and two-level hybrid caching techniques. Section 4 presents evaluation results of our proposed design with realistic and synthetic workloads. After discussing related works in Section 5, we conclude in Section 6.

2 Background

When SSDs are comprised for an array, the write cache on a traditional RAID controller can be optimized considering the hardware characteristics of devices. SSD has unique performance characteristics different from HDD. SSD can exhibit substantial variance in request latency and throughput as a result of garbage collection [9], [11], [17]. The main problem of applying traditional write cache architecture to an array of SSDs is that each SSD in the array can show different performance such as bandwidth and I/O response time due to uncontrolled GC process and the overall read and write bandwidth of the array is limited by the slowest SSD.

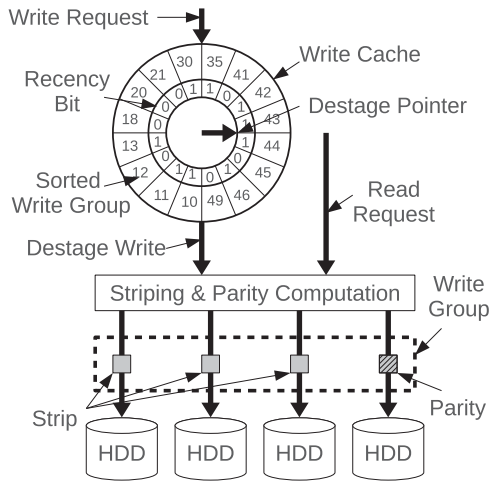


Fig. 1. Depiction of the-state-of-the art write cache algorithm, Wise Ordering of Writes (WOW) [16] on the RAID controller for HDD arrays.

2.1 Traditional Write Cache for HDD Arrays

A traditional RAID controller employs a write cache [16], [18]. It can employ non-volatile memory or battery-back memory in order to prevent data loss in case of power-failure on the system [16], [18]. As long as there is free space in the write cache, an incoming write request is stored in the write cache and it is immediately committed to the requester. Otherwise, the request should be pending in the I/O queue until the write cache becomes available. The data in the write cache will be synchronized with the disks later in background. This process is called *destaging* and write operations to the disk issued for destaging are called *destage writes*. A destage write is split into multiple strips. Here, a *strip* means a discrete chuck of data, which is going to be stored in *one* disk. Depending on the RAID configuration, a parity strip may be added. A *write group* is a set of strips that have to be stored in disks at the same time. In WOW, when a write request comes, it is split to strips and strips are stored in their corresponding write groups. When a destage write is triggered, all strips in a write group are transferred to their corresponding disks.

Fig. 1 presents an architectural overview for traditional write caches in HDD arrays and their write back cache algorithms. Wise Ordering of Writes [16] is the-state-of-the art write cache algorithm. In WOW, a write hit can be either a hit on a strip or a hit on a write group. A write group consists of multiple strips. A hit on a strip means there already exists the requested strip in the write cache. Even if the requested strip does not exist, its write group could exist (hit on a write group). Since a parity strip is computed for each write group and parity is computed when a write group is destaged, a hit on a write group can reduce the number of parity computations. Write groups are always ordered by their logical block address (LBA), which leverages spatial locality.

What to destage follows a least recently used (LRU) policy that implements a heuristic approach using recency bits. When destaging is necessary, the destage pointer advances. If the recency bit of the write group pointed by the destage pointer is zero, the write group is destaged. If the recency bit is one, the write group is retained and the recency bit is cleared. Then the destage pointer advances again until it finds a write group whose recency bit is zero. The recency bit is set when the write group is hit. The recency bit gives one more chance for hit write groups to survive for one more cycle, which leverages temporal locality.

In addition to what to destage, the write cache needs to determine when to destage. WOW adopts a linear threshold scheduling to determine when to start destaging and how to adjust the destaging rate [19]. Low and high thresholds are involved. Destaging begins to be triggered when the number of write groups (W) in the write cache reaches a low threshold. The destaging rate linearly increases as W increases. If W reaches a high threshold, it runs at a full rate.

To fully exploit benefits of a write cache in WOW, a cache controller should be designed to leverage temporal and spatial locality as well as to resist bursty writes [16], [18]. High temporal locality can be achieved by keeping as much hot data as possible. On the other hand, to sustain bursty writes, dirty data in the cache should be destaged in advance before the bursty writes come in. Otherwise, it will increase pending requests in the queues, causing delays for writes. In the context of HDDs, leveraging spatial locality means minimizing mechanical movement inside a HDD. Extensive research has been conducted on these cache scheduling algorithms to exploit the spatial locality on a HDD [20], [21]. In order to minimize the delay due to the cache being full while maintaining locality requirements, the cache controller should be carefully designed to make an intelligent decision on when to destage, what to destage, and how much to destage.

There are other basic approaches to decide when to destage. In [22], a rapid destaging is performed on the dirty blocks in idle time by prioritizing the bursty write resistance over temporal locality. In [19], a balance between bursty write resistance and temporal locality is emphasized by scheduling linear threshold levels. Here, destaging rate varies in accordance with the amount of data present in the cache. At low occupancy, slow destaging takes place to ensure locality advantage and at higher occupancy the destaging rate grows proportionally to avoid space tightening.

STOW [18] further improves WOW by identifying random and sequential writes distinctly. All write back cache algorithms including WOW [16] and STOW [18], however well designed, do not consider the characteristics of drives, and will suffer from performance degradation due to uncoordinated garbage collection processes on an array of SSDs. Therefore, *our claim is that write cache algorithms for an array should be redesigned by considering the device characteristics.*

2.2 Problem Definition: Slow Destaging I/Os

The pathological behavior of an array of SSDs has been reported by one of previous studies [12], [23], [24] where they conducted measurements of individual SSDs and the array of SSDs for a variety of I/O workload patterns, and made three important observations: (i) individual SSDs experience sudden bandwidth drops and the frequencies of those bandwidth drops are more likely to happen when the intensity of write requests in the workloads increases, (ii) the performance variability of an array of SSDs increases with the number of SSDs, and (iii) per-drive bandwidth decreases with the number of SSDs. The main cause of this pathological behavior of the SSD array is attributed to uncoordinated GC operations of individual SSDs in their array [12], [23], [24] The write cache on the RAID controller could be utilized to mitigate the problem of the pathological behavior of the SSD array, however, when blocks on the write cache are destaged into the SSD array, uncoordinated destage writes can hurt overall I/O performance of the SSD array.

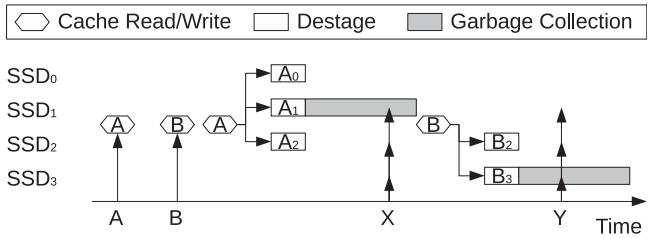


Fig. 2. Timing diagram of destage writes in traditional write caches.

Fig. 2 illustrates how destage writes can be slowed down when they interfere with some GC operations. Suppose that write requests A and B arrive, which are stored in the cache immediately, and the write cache controller will decide destage at some later time. The request A is split into A_0 , A_1 and A_2 which are going to SSD_0, SSD_1 and SSD_2, respectively. If a strip that goes to SSD_1 is delayed by GC, the destage write cannot be committed until SSD_1 finishes GC and processes strip A_1. Only after SSD_1 finishes processing A_1, the cache controller can destage the next write group. If bursty write requests come during this period, the write cache may become full, which incurs long latency to subsequent write requests. Even while SSD_1 is delayed by GC, other SSDs can accept strips. This is because all strips in the same write group have to destage together for parity computation. However, if strips can be destaged independently, it can avoid delay caused by the slow SSD. In HDD arrays, the performance variability is not a big problem because the performance variability is not significant in HDDs. In contrast, in SSDs, the shortest response time and the longest response time could be order-of-magnitude different. Thus, the performance variability should be carefully managed when a write cache is designed.

3 Layout-Aware Write Cache

LAWC is motivated to answer a simple question: *how can we exploit the underlying storage architecture to minimize GC overheads in an array of SSDs?* The ultimate goal of LAWC is to achieve the performance improvement by mitigating performance degradation caused by GCs in SSDs. LAWC has three design components: (i) I/O scheduler with pre-parity computation by logically partitioned write cache design, (ii) destage write synchronization technique, and (iii) two-level hybrid caching technique. LAWC is implemented by modifying the state-of-the-art write cache algorithm–WOW [16]. These three components are explained one by one in following sections.

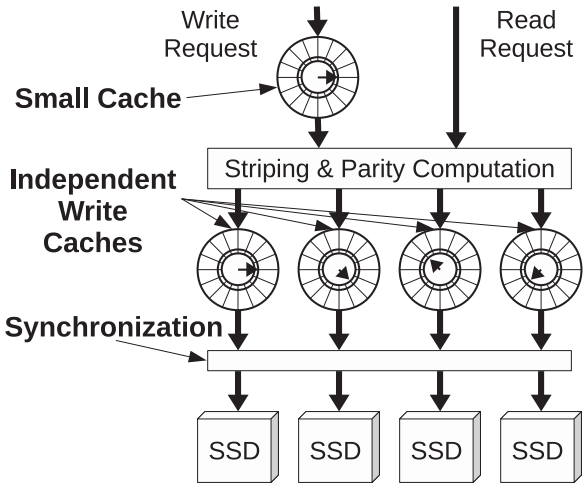


Fig. 3. Architectural overview. The figure shows logically partitioned write caches, instead of a single unified write cache and a small non-volatile front write cache.

3.1 I/O Scheduling with Pre-Parity Computation

Traditional write caches employ a single unified write cache. A unified write cache does not effectively handle the performance variance of SSDs. As shown in Fig. 2, if any SSD is delayed by GC, which takes a significant amount of time, no destage operation can be performed until the GC finishes. To mitigate this problem, we split the write cache so that other write caches can perform destage operations even if one of SSDs is delayed by GC. LAWC employs multiple independent write caches instead of a single unified write cache. Each cache is implemented using separate request queue. In addition, LAWC can use a small non-volatile front cache to further improve the efficiency of the next level write cache. The architectural depiction of LAWC is presented in Fig. 3.

In LAWC, to maximize independent destage operations, parity has to be computed before strips are cached. The pre-parity-computation is not free. It involves parity computation overhead and additional internal I/O overhead cost in the SSD array. Moreover, even if there is a write cache hit, in order to compute parity, it has to additionally read its related data blocks. However, the performance overhead of the pre-parity-computation can be compensated by *destage write synchronization* and *two-level hybrid caching* techniques, described in detail in Sections 3.2 and 3.3. Note that there is no write operation involved in this parity computation process due to write cache hits in the SSD array. Thus, it does not affect the lifetime of SSDs by this operation.

In order to understand the overhead of additional I/Os incurred by the pre-parity-computation and its impact on the locality, let us consider the following example as illustrated in Fig. 4. To simplify the discussion, we consider the RAID-4 with four SSDs (three data SSDs and one parity SSD), which are denoted as SSD_0, SSD_1, SSD_2 (data SSDs) and SSD_p (parity SSD). The scenario is that request A comes, which is split to A_0 and A_1 going to SSD_0 and SSD_1, followed by request A', which is split to A'_1 and A'_2 going to SSD_1, SSD_2. Requests A and A' belong to the same write group.

Fig. 4a shows the WOW algorithm. In WOW, Request A is stored in the write cache if there is room. When request A' arrives, which belongs to the same write group of request A, but updates different strips. If request A is still stored in the write cache, a hit on a write group occurs. Thus, A_1 and A_2 are updated to A'_1 and A'_2 in the write cache. After a while, when they need to be destaged, A_0, A'_1, and A'_2 are stored to SSD_0, SSD_1, and SSD_2, respectively. The parity is computed when they are destaged and stored in SSD_p. Thus, four write operations are performed in total.

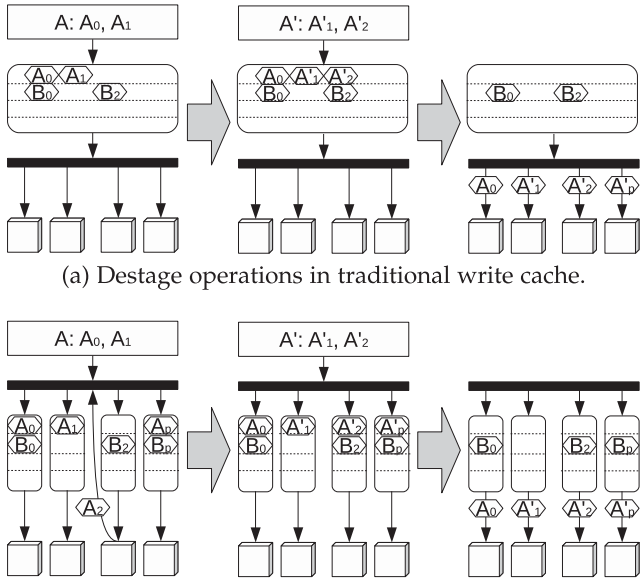
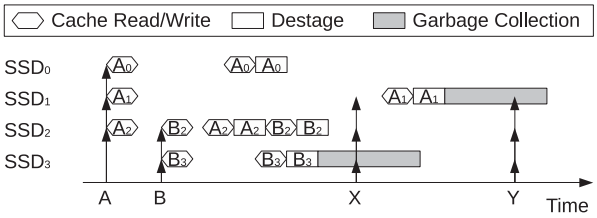


Fig. 4. Illustration of destage writes (WOW versus LAWC).

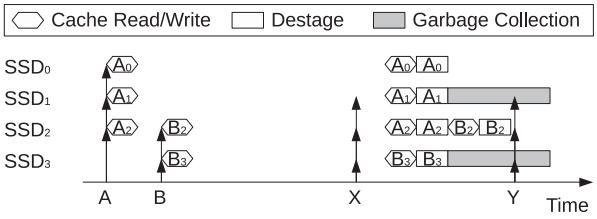
Fig. 4b illustrates our proposed pre-parity-computation and SSD-aware I/O scheduling algorithm in LAWC. In LAWC, the entire physical write cache is logically partitioned into sub write caches as many as the number of SSDs in the array. Suppose such logically partitioned write caches are W_0, W_1, W_2 and W_p for SSD_0, SSD_1, SSD_2 and SSD_p, respectively. When request A arrives, A_0 and A_1 are stored in W_0 and W_1. To compute the parity a read operation is performed to SSD_2, which reads A_2; the parity is computed; and the parity strip, A_p is stored in W_p. When request A' arrives, A'_1 in W_1 is replaced (hit on a strip) to A'_1, A_2 is updated to A'_2 in W_2. To compute parity, the RAID controller needs to read A_0, A'_1, and A'_2, all of which are stored in their corresponding write caches. Thus, no more read operation from disks is required. The parity strip, A_p, is updated to A'_p in W_p (hit on a strip). Therefore, until they are destaged, 4 write and 1 read operations are involved.

LAWC can incur additional read operations to compute the parity, but does not increase write operations as shown in the example. This is because the write caches can still absorb write requests to SSDs as long as write cache space is available. As it will be demonstrated by experiments in Section 4, write caches of LAWC have lower chance of being full. As for additional read operations, they do not have significant impact on the overall performance because read operations are very fast in SSDs. Also, it does not have any adverse impact on the lifetime of SSDs.

The aforementioned example illustrates the situation where both temporal and spatial localities are exploited by LAWC. A_1 is updated by both request A and A'. Since A_1 is stored in W_1, a hit occurs when request A' comes, which is an example of temporal locality. Request A and A' are



(a) Independent write cache.



(b) When synchronization is employed.

Fig. 5. Timing diagram of destage writes in LAWC.

spatially adjacent. Since they are of the same write group, they could be coalesced. In case of WOW, the coalescing is explicit. In case of LAWC, it is not explicit, but it happens. As shown in the example, request A' does not incur any additional I/O to SSDs. By exploiting existing strips of request A, request A' is coalesced with request A.

As for the error recovery, the proposed design has no issues with recovery from a disk failure because it employs non-volatile memory as a cache. If the RAID controller detects failure on an SSD, it destages all the data stored in the cache into the SSDs except for the failed SSD before it starts rebuilding the failed disk. Then it can rebuild the failed SSD based on other SSDs since all the up-to-date data is in the SSDs.

3.2 Destage Write Synchronization

Independent write cache structure allows flexible I/O scheduling for destage writes. However, uncoordinated GCs across SSDs may interfere with incoming requests. Fig. 5a illustrates this situation. The scenario of incoming requests is the same with Fig. 2. When request A comes, it is immediately split into three strips and they are stored in corresponding write caches. One of the three strips may be a parity strip, which means that parity computation is done at this moment (by pre-parity-computation). Similarly, when request B comes, which is independent from request A, it is split into two strips and they are stored. At this moment, SSD_0 has A_0 in its write cache; SSD_1 has A_1; SSD_2 has A_2 and B_2; and SSD_3 has B_3. If all write caches work independently, destage writes can also happen independently. It is hard to predict when any request is affected by GC. In this example, read requests X and Y come after write requests A and B. Request X is split to SSD_1, SSD_2, and SSD_3. When request X arrives, SSD_3 is running GC. Therefore, request X cannot be committed until the strip going to SSD_3 completes. In the same way, request Y is markedly delayed by SSD_1.

To address this issue, we propose a *destage write synchronization* technique. It attempts to destage strips in all write caches together at the same time. If all the write caches have at least one strip to destage, they destage together. Note that those strips destaged together do not have to belong to the same write group. The destage pointer advances independently. It advances while clearing the recency bit, until it finds a strip whose recency bit has been already cleared. The strip with cleared recency bit is ready to destage. When the pointer finds a strip to destage, it has to stop there until all the other pointers find one. However, if the waiting time of those strips to be destaged together exceeds a predefined threshold, it is allowed to destage to prevent the waiting write caches from becoming full. Except for those having no strip, other write caches can destage together.

The destrage write synchronization technique incurs two types of overhead. One is additional complexity to scheduling, and the other is additional delay in destrage writes. The additional complexity is negligible because what is added is only looking up other write caches to check if they have a strip to destage. The additional delay occurs because the write cache should wait for other caches to be ready. This may cause the waiting cache to become full, when it might be available if the synchronization were not employed. However, this additional delay can be compensated by mitigated GC delay effect as demonstrated by experiments in Section 4.

The pre-parity-computation followed by the parities to be cached will help increase the chances that a whole strip group are destaged together. Moreover, with this pre-parity-computation, multiple write caches are independent in that strips belonging to one write group do not have to be destaged together because a parity strip is already computed. Even if any SSD is delayed by GC, independent scheduling of write caches allow other SSDs to accept a strip from their corresponding write caches.

GC synchronization of individual SSDs in the array can improve the overall performance of an SSD array [12], [13]. Kim et. al. [12], [13] proposed a coordinated GC mechanism to forcefully trigger GCs of individual SSDs at the same time. However this technique requires bus interface (e.g., SATA) to be modified for synchronization of GCs and all SSDs in the array to support it. In LAWC, instead of GCs of individual SSDs, destage writes are synchronized because destage writes cause to trigger GCs. Synchronization of destage writes increases the possibility of GC synchronization.

Fig. 5b illustrates the impact of synchronization. As shown in this figure, destages writes, A_0, A_1, A_2, and B_3, are triggered at the same time. Note that B_3 does not belong to the same write group of other strips. The synchronization mechanism contributes to the performance improvement by increasing the chance of GCs running simultaneously. Since GC is triggered by a destage write, synchronized destage writes increase the probability of GC synchronization between SSDs. In this example, we can see that request X is not affected by any GC whereas request Y is now affected by two GCs. However, even if request Y is delayed by two GCs, its response time does not become doubled because the two strips are serviced in parallel.

3.3 Two-Level Hybrid Caching

One drawback of LAWC is additional computation overhead by the pre-parity-computation. Even though the write cache hits, the parity computation delay is always added to the response time. The overhead is compensated by coordinating GC processes when the workload is write-intensive, which means the inter-arrival time of subsequent write requests is short and their request size is large. However,

for non-intensive workloads, this overhead cannot be compensated. Since the GC processes do not incur much overhead for the non-intensive workloads, the pre-parity-computation overhead is hardly compensated by coordinating them. To overcome this problem, a small write cache is employed to mitigate the pre-parity-computation overhead for non-intensive workloads.

As illustrated in Fig. 3, an additional small write cache can be employed in front of independent write caches. The small write cache works in the same way as WOW does except for when it is full. In WOW, when it is full, subsequent write requests are pending until it becomes available. In LAWC, when the small cache is full, subsequent write requests bypass the cache unless they have dependency on those in the small cache. Here, dependency means requests on the same address. For non-intensive workloads, the small cache can effectively serve requests. Since it does not require pre-parity-computation, its performance is close to WOW. For intensive workloads, small cache can more frequently become full. Most of requests are likely to bypass the small cache. When they bypass the small cache, pre-parity-computation overhead incurs, but this overhead is compensated by the mechanism of independent write caches and destage write synchronization technique. For intensive workloads, LAWC can improve performance by coordinating GC processes. In summary, for non-intensive workloads, the upper-level small cache can serve most of requests and offer similar performance with WOW, while for intensive workloads, the small cache is bypassed and the independent write caches contribute to the performance improvement.

3.4 Algorithm: Putting All Together

Algorithm 1 shows the pseudo code of LAWC that integrates all three techniques aforementioned. When a write request arrives, a procedure named `RequestArrive` is called. It first checks if a front small cache is full. If it is full, it forwards the request down to independent write caches unless it has dependency. If the incoming request has dependency on requests stored in the small cache, the incoming request is pending until the small cache becomes available. Otherwise, it stores the request in the small cache. The small cache will make free space after destage procedure is performed.

In `IndependentWriteCache`, the incoming request is split into strips and the parity is computed. All strips including the parity are stored in their corresponding write caches. If any write cache is full, the request is pending until it becomes available by destage writes. Destage writes are triggered independently from handling requests. The small cache destages strips in the same way with WOW.

`DestageSmallCache` shows the destage write algorithm. The destage pointer indicates a candidate write group. If the recency bit of a candidate is true, the bit is cleared and the write group is retained. The destage pointer advances until a write group whose recency bit is false is found. The write group with false recency bit is removed from the small cache and forwarded down to independent write caches.

Destaging of each level write cache (front write cache and next level independent write caches) is triggered independently. A candidate strip for destaging is selected in the same way with WOW, as shown in `Destage`. Note that a unit of destaging is a strip in independent write caches, whereas it is a write group in the small cache. Even though a candidate is found, it will not be immediately destaged. Instead, it is delayed until other write caches find candidates or the timer expires. The timer prevents a write cache from being pended for a long time.

TABLE 1
SSD Model Parameters

SSD configuration	
Total capacity	8 GB
Reserved free blocks	15 %
Minimum free blocks	5 %
Cleaning policy	Greedy
Flash chip packages	4
Planes per package	4
Blocks per plane	512
Pages per block	64
Page size	4 KB
Latency	
Page read	0.025 ms
Page write	0.200 ms
Block erase	1.5 ms
Copy back	disabled
RAID	
No. of SSDs	8, 9
Redundancy	RAID-0/5
Write cache	32 KB per SSD

4 Evaluation

4.1 Experimental Setup

In order to study the performance implications of LAWC, we enhanced the DiskSim and SSD simulator developed by Microsoft Research [17]. Our tests were performed on RAID-0 and RAID-5. In RAID-0, eight SSDs are employed, and in RAID-5, nine SSDs are employed, where eight SSDs are for data strips and one is for a parity strip. The strip unit size for RAID is 128 KB, so the 1 MB request constitutes a write group (128 KB $\times$ 8 = 1 MB) for RAID-0 configuration. In simulating an SSD, we used 15 percent reserved free blocks with 5 percent minimum free blocks and a greedy GC policy. Each SSD uses four flash chip packages, where each package consists of four planes, and each plane uses 512 blocks. Each block consists of 64 4 KB pages. Thus, the size of each SSD is 2 GB. Page read and write times of 0.025 and 0.2 ms are used and block erase time of 1.5 ms is used. The simulation parameters are summarized in Table 1.

We assume a byte-addressable non-volatile memory is employed for the small cache and independent write caches. Any type of byte addressable non-volatile memory (NVM) is applicable. The read and write latency varies a lot depending on the type of the memory. For evaluation, we used phase-change random-access memory (PRAM) as the front write cache, where read and write latencies are 0.125 and 1 μs respectively [25]. The write cache configuration parameters are shown in Table 2.

Implementation. DiskSim implements a single global queue at the I/O driver to handle incoming I/O requests. For implementing WOW write cache, we implemented another write cache layer on the RAID controller. When I/O requests are dequeued from the I/O driver queue, they are inserted to

TABLE 2
Write Cache Parameters

Configuraton	RAID-0	RAID-5
Size of small cache (S_s)	64 KB	72 KB
Size of one write cache (S_c)	24 KB	24 KB
No. of SSDs (N)	8	9
Total Size ($S_s + N \times S_c$)	256 KB	288 KB
Read Latency	0.125 μs	
Write Latency	1 μs	

the WOW write cache queue on the RAID controller. Unlike WOW write cache queue, LAWC implements partitioned subqueues on the RAID controller as many as the number of SSDs in the array. To handle the events of cache hit and miss, we defined hit and miss events for WOW and LAWC.

Workloads. We used a mixture of real enterprise-scale I/O workloads and synthetic HPC workloads to study the efficacy of LAWC on a wide spectrum of I/O workloads. We use Microsoft (MSR) Exchange Server [26], Cello [27], Financial [28], TPC-H [29], FIU-VM [30], and FIU-MAIL [30] traces as real-world enterprise workloads. In addition, a skewed trace is used to validate the effectiveness of LAWC for a heavily skewed workload. To further explore the performance of the proposed cache design in multi-application storage environment, we used the multistream trace which is a mixture of different workloads. For an intensive multistream workload, we mixed Exchange and Cello applications, which is denoted by MS-EC. And for a non-intensive multistream workload (MS-FT), we used a mixture of Financial and TPC-H. Finally, we mixed these four traces into one multistream workload, which is denoted by MS-ECFT. Characteristics of these workloads are summarized in Table 3.

In addition, for the synthetic HPC workload, we used a synthesized HPC trace that was generated based on the characterization study of real HPC I/O workloads of the Spider file system [7]. The HPC trace analyzed shows about 60 percent writes and 40 percent reads, a bi-normal distribution for request size, and a Poisson distribution for inter-arrival times with an average of 20 ms. For the request size distribution, there are about 50 percent 4K small requests and 50 percent large requests (17 percent of 512 KB and 32 percent of 1 MB requests). Based on this study, we performed experiments varying I/O request size, arrival rate of the requests, and the percentage of write requests. Table 4 summarizes the values of these parameters used in the experiments.

4.2 Performance Analysis of LAWC

To evaluate the performance of LAWC, we compare the following algorithms:

TABLE 4
Synthetic Workload Characteristics

Parameter	Values
Request size (R)	4 KB, 512 KB, 1024 KB, 4096 KB
Inter-arrival time (I)	10 ms, 20 ms, 40 ms
Write percentage (W)	20 %, 60 %, 80 %

W(R, I, W) denotes request size R (KB), inter-arrival time I (ms), and write percentage W (%).

- WOW [16]: A state-of-the-art write cache algorithm for RAID of HDDs.
- GGC [12]: A technique that improves the performance of RAID by coordinating GCs of SSDs.
- IWC: An independent write caches (IWC) without synchronization nor the small cache.
- IWC+S: an enhanced IWC with destage write synchronization technique but not the small cache.
- LAWC: a combination of IWC, synchronization and the small cache.

To make fair comparisons, the total amount of cache size (including the small write cache) is the same across different architectures.

WOW employs a linear threshold scheme to determine when to trigger destage writes, as mentioned in Section 2.1. The same scheme could be applied to the small cache and independent write caches in LAWC. However, in our implementation, both high and low threshold values are zero, which means destaging is triggered immediately at a full rate when any request is stored in write caches of both LAWC and WOW. Due to fundamental difference in architecture, different threshold values have different impact on performance of LAWC and WOW. For example, LAWC offers the best performance with a certain threshold value, which does not work for WOW. Since the threshold value is orthogonal to LAWC design, we exclude the impact of threshold for fair comparison by setting them zero.

4.2.1 Impact of Front Cache Size

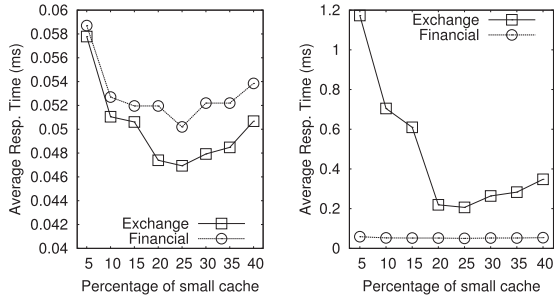
Recall that LAWC consists of a small cache and an array of independent write caches. Different size distributed between the small cache and the independent write caches was experimented to get an idea of the size distribution which offers optimal performance. This experimental analysis is important to decide the size ratio of the small buffer to the main cache by learning how size has an effect on overall performance. Exchange and Financial workloads are used as one representative intensive workload and one non-intensive workload for these sensitivity analysis respectively.

TABLE 3
Characteristics of Realistic Workloads

Workload	Write IOP/s	Write Size (KB)	Read IOP/s	Read Size (KB)	Total IOP/s	Total Size (KB)
Exchange	533.57	7.77	159.40	8.01	692.87	7.82
Cello	55.36	6.05	18.91	7.96	73.53	6.54
Financial	4.08	8.44	1.81	4.05	5.89	7.04
TPC-H	4.57	10.19	17.03	37.35	21.59	31.62
FIU-VM	15.80	4.00	3.80	4.00	19.16	4.00
FIU-MAIL	83.62	4.00	62.02	4.00	142.19	4.00
Skewed	199.76	4.00	50.33	4.00	250.08	4.00

Algorithm 1. Pseudo-Code of LAWC

```
1  PROCEDURE: ReqestArrive
2  Input: R /* A write request
3  if the small cache is full then
4    if R has dependency on requests in the small cache then
5      wait until the small cache is available;
6    else
7      call IndependentWriteCache(R);
8    end
9  else
10   store R to the small cache;
11 end
12
13 PROCEDURE: IndependentWriteCache
14 Input: R /* A write request
15 split R to strips, R_0, R_1, ... R_k;
16 compute parity strip R_p;
17 for all strips do
18   if the corresponding write cache is full then
19     wait until the cache is available;
20   end
21   store the strip to the corresponding write cache;
22   set the recency bit of the stript as true;
23 end
24
25 PROCEDURE: DestageSmallCache
26 if the write cache is not empty then
27   repeat
28     advance the destage pointer;
29     if the recency bit is true then
30       clear the recency bit;
31     end
32   until a write group is found whose recency bit is false;
33   call IndepedentWriteCache(the found write group);
34 end
35
36 PROCEDURE: Destage
37 Input: i /* The index of the write cache
38 if the write cache is not empty then
39   repeat
40     advance the destage pointer;
41     if the recency bit is true then
42       clear the recency bit;
43     end
44   until a strip is found whose recency bit is false;
45   mark i − th write cache is ready to destage;
46   repeat
47     wait;
48   until Synchronization();
49   destage the found strip to the corresponding SSD;
50 end
51
52 PROCEDURE: Synchronization
53 Output: true or false
54 if all write caches are ready to destage then
55   clear marks;
56   reset timer;
57   return true;
58 end
59 if timer expires then
60   clear marks;
61   reset timer;
62   return true;
63 end
```



(a) RAID-0 (b) RAID-5

Fig. 6. Performance of LAWC compared to cache size.

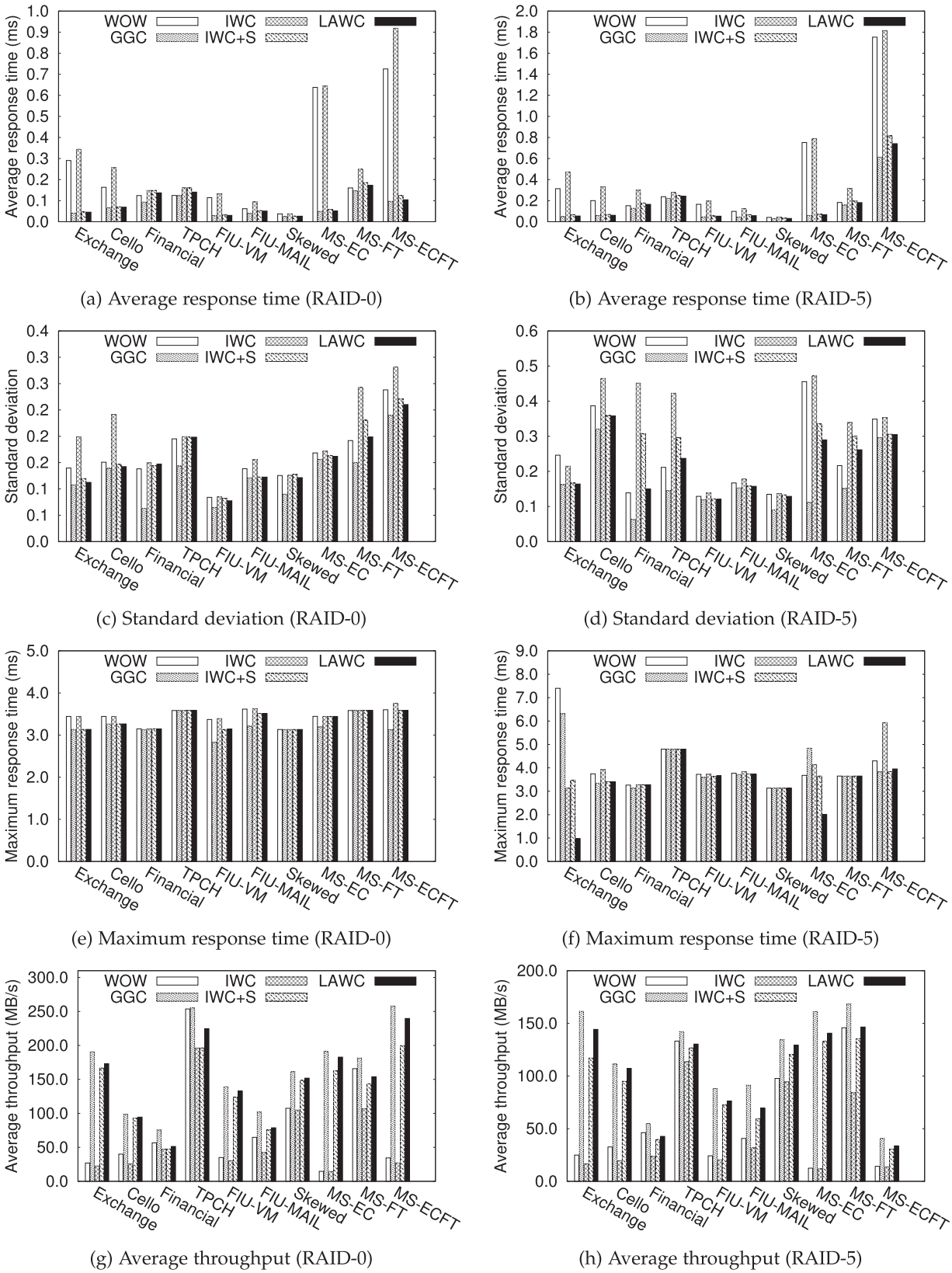
In our experiment, the small cache ratio is calculated as the following; Denote the size of the small cache as S_s, the number of SSDs as N, and the size of one write cache S_c. Then the ratio is calculated as $\frac{S_s}{S_s+N\times S_c.}$ We explored the ratio by changing S_s and S_c while keeping the total size $(S_s + N \times S_c)$ unchanged. For example, we used 64 KB as S_s and 24 KB as S_c, which give us 256 KB as the total size and 25 percent as the ratio for RAID-0 (refer to Table 2). If we use 24 KB as S_s and 29 KB as S_c, the total size is the same (256 KB), but the ratio is 9.375 percent. Fig. 6 shows the performance of LAWC with respect to the varied ratio of small cache to the main cache. Precisely, the size of the small cache and the size of independent write caches are varied to explore different ratios. In both Figs. 6a and 6b, we observe that the peak performance is achieved when the small cache size is 25 percent. For a fair comparison, we use 25 percent as the small cache size ratio throughout the rest of simulations.

4.2.2 Evaluating LAWC for Enterprise Workloads

Fig. 7 shows the results with real enterprise workloads. We observe that WOW outperforms IWC because IWC has separate caches, but WOW has a unified cache, which results in higher utilization of the cache space. In IWC, one of write caches may become full while there is room in other write caches because there is no interaction between write caches. In contrast, WOW can mitigate this problem because all SSDs share the unified write cache.

We see that IWC does not help to improve performance. However, when the destage write synchronization technique is employed, the performance of IWC+S is drastically improved when compared to IWC. In particular, for write-intensive workloads (Exchange, Cell, FIU-VM, FIU-MAIL, MS-EC, and MS-ECFT), IWC+S significantly outperforms WOW. IWC+S outperforms WOW by 82.39 and 68.51 percent for RAID-0 and RAID-5, respectively, as shown in Figs. 7a and 7b. This is because IWC+S increases the probability of GC synchronization. GGC enforces GC coordination with a help of modified block interface on both RAID and individual controllers, which results in the best performance in Figs. 7a and 7b.

The drawback of IWC+S could be the pre-parity-computation. The overhead of pre-parity-computation is compensated by GC synchronization for write-intensive workloads, but for non-intensive workloads (Financial and TPC-H), it is not. By employing the small cache, LAWCcan mask the performance degradation due to this overhead from the response time. Thus, LAWCoffers similar performance with WOW for non-intensive workloads.



(a) Average response time (RAID-0)

(b) Average response time (RAID-5)

(c) Standard deviation (RAID-0)

(d) Standard deviation (RAID-5)

(e) Maximum response time (RAID-0)

(f) Maximum response time (RAID-5)

(g) Average throughput (RAID-0)

(h) Average throughput (RAID-5)

Fig. 7. Performance comparison of different implementations. Each bar is shown as the following order: WOW, GGC, IWC, IWC+S, and LAWC.

The difference in the average response time between LAWC and WOW is 8.19 and 2.19 percent for Financial and TPC-H, respectively. Therefore, LAWC offers similar performance with WOW for non-intensive workloads, while outperforming WOW for intensive workloads.

We performed experiments with highly skewed trace. In the skewed trace, 50 percent of requests are on one SSD, and 50 percent of requests are randomly distributed. Though the performance improvement of LAWC for the skewed trace is not as significant as for other write-intensive

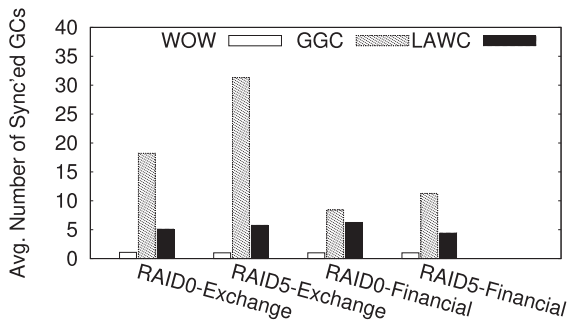
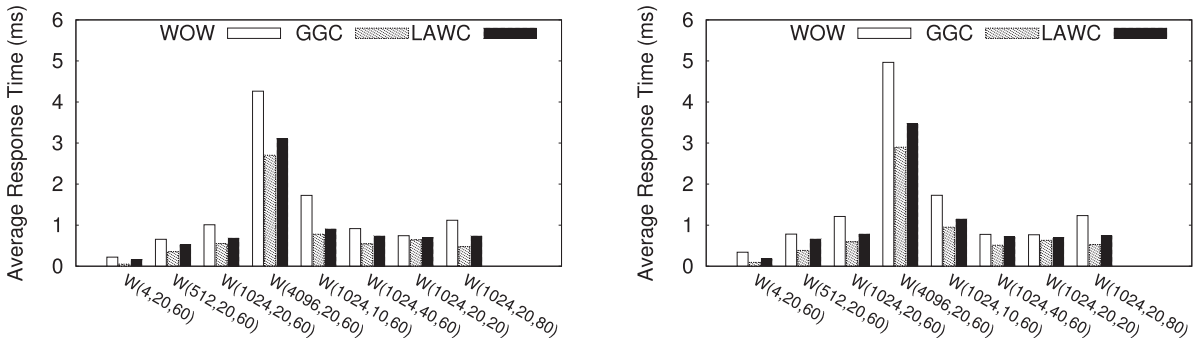


Fig. 8. Average number of synchronized GC processes. Each bar is shown as the following order: WOW, GGC, and LAWC.

workloads, LAWC still offers drastic improvement in average response time. Specifically, LAWC improves the average response time by 29.22 and 24.47 percent compared to WOW for RAID-0 and RAID-5, respectively.

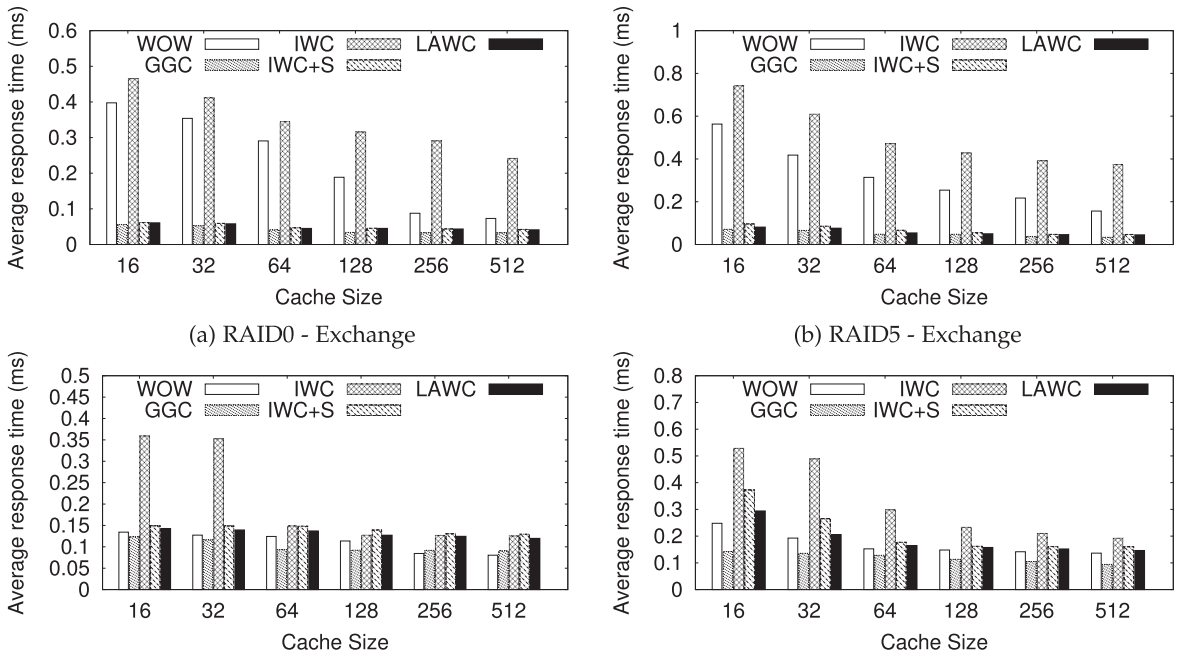
In both RAID-0 and RAID-5, the trend of the standard deviation, shown in Figs. 7c and 7d is similar with that of the average response time. GGC exhibits the smallest standard deviation, and LAWC offers smaller standard deviation compared to WOW. Specifically, LAWC offers 4.77 and 10.79 percent smaller average standard deviation compared to WOW, for RAID-0 and RAID-5, respectively. In case of maximum response time shown in Figs. 7e and 7f, it is similar across all implementations. The throughput is reversely related with the response time. Figs. 7g and 7h show that the average throughput of LAWC is improved by 85.53 and 78.35 percent compared to WOW for RAID-0 and RAID-5, respectively.

Effect of Synchronized Destage Writes. The synchronization mechanism improves the performance by increasing the probability of GCs operating simultaneously. Fig. 8 compares the average number of synchronized GC processes. We count the number of SSDs that run GCs during a fixed time slice, for which we use 0.1 ms in our measurement. This result clearly shows that the probability of GC synchronization markedly increases with LAWC compared to WOW. The average number of synchronized GC operations is increased by up to 6.24 times.



(a) RAID-0 (b) RAID-5

Fig. 9. Comparing WOW and LAWC for a wider range of workload characteristics. Each bar is shown as the following order: WOW, GGC, and LAWC.



(a) RAID0 - Exchange (b) RAID5 - Exchange

(c) RAID0 - Financial (d) RAID5 - Financial

Fig. 10. Sensitivity analysis for different cache sizes. Each bar is shown as the following order: WOW, GGC, IWC, IWC+S, and LAWC.

TABLE 5
Write Cache Access Latency Parameters for the Sensitivity Analysis

Configuration	Read latency (μs)	Write latency (μs)
SS (Very Slow)	0.5	4
S (Slow)	0.25	2
B (Baseline)	0.125	1
F (Fast)	0.0625	0.5
FF (Very Fast)	0.03125	0.25

4.2.3 Evaluating LAWC for HPC Workloads

In order to cover a wide-range of workload characteristics, we evaluated LAWC with synthetic HPC workloads. Fig. 9 shows the results comparing LAWC and WOW. The main benefit of employing LAWC comes from large requests. In Fig. 9a, compared to WOW, the response time of W(1024, 20, 60), W(4096, 20, 60), and W(1024, 10, 60) is reduced by 32.33, 27.13, and 47.24 percent, respectively. Since the response time of large requests is much longer than that of small requests, the response time reduction for large requests substantially improves the overall performance. Varying inter-arrival time (e.g., W(1024, 10, 60) and W(1024, 40, 60)) and percentage of write requests (e.g., W(1024, 20, 20) and W(1024, 20, 80)) does not affect the overall trend.

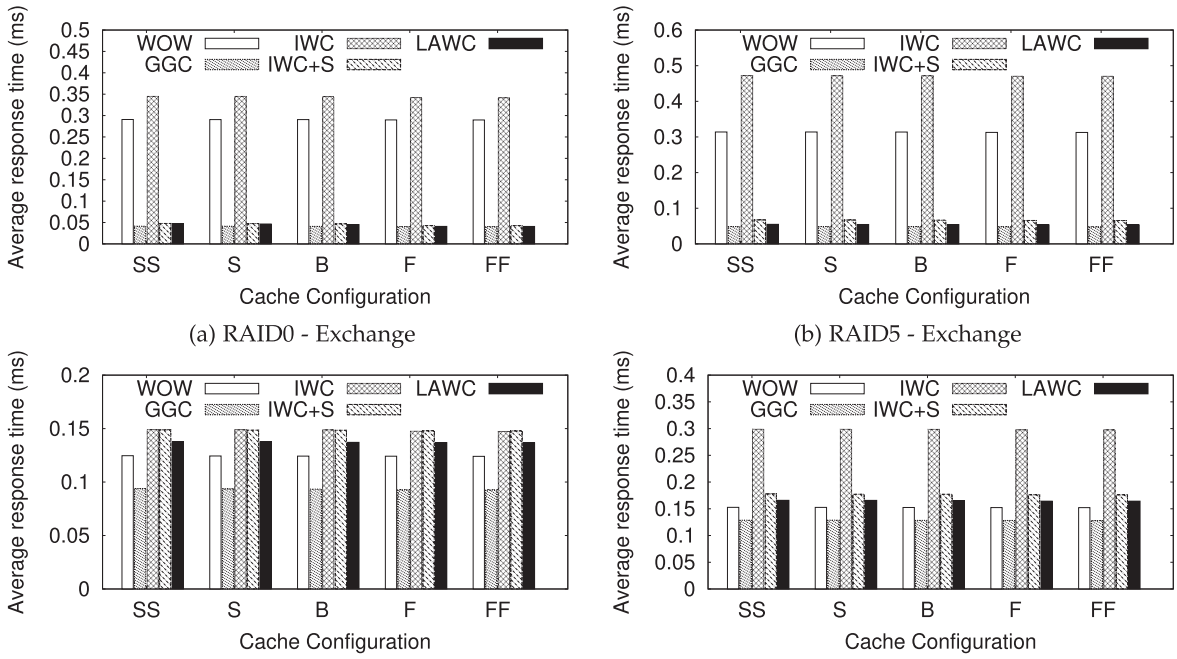
If the RAID configuration requires parity computation (RAID-5), the overhead of the pre-parity-computation increases, but the front small cache effectively hides it. Fig. 9b shows the results of RAID-5. For W(1024, 20, 60), W (4096, 20, 60) and W(1024, 10, 60), the performance improvement is 35.22, 30.14, and 33.36 percent, respectively.

4.2.4 Sensitivity Analysis with Different Cache Parameters

To further examine the effective performance of the proposed techniques, different cache sizes were used to study the performance of RAID SSD with intensive and non-intensive workloads. Exchange [26] and Financial [28] workloads were chosen for evaluation.

Fig. 10 show the results of RAID-0 and RAID-5 configurations for Exchange and Financial workloads. The results for intensive workload show that, for all cache sizes, from small to large, LAWC still outperforms WOW in terms of average response time. We can clearly see WOW performs worst in all different cache sizes, whereas LAWC offers comparable performance to GGC. LAWC offers 84.67 percent decrease in response time when compared with WOW and performs 8.47 percent worse than GGC. Therefore, LAWC offers drastic performance improvement by exploiting internal cache design. The results of non-intensive workload, shown in Figs. 10c and 10d, show that for small cache size, performance of WOW is 9.57 and 15.76 percent better than LAWC. The performance of LAWC is comparable even for non-intensive workloads when the cache size is small. For other sizes, the performance is in accordance with other results; LAWC offers more optimal performance than IWC+S for non-intensive workloads, by mitigating the pre-parity-computation overhead.

To quantify the impact of the write cache access time, we performed a sensitivity analysis with different access latencies. The cache latency parameters used for this analysis are summarized in Table 5. As shown in Fig. 11, the cache configuration (access latency) has negligible impact on the results. This is because the portion of the access time of the cache over the average overall access time is not significant. For example, in the case of LAWC, RAID0, and Exchange, the average access time is 0.045 ms while the read and write latency of the baseline cache configuration is only 0.000125 and 0.001 ms, respectively. Furthermore, we assumed the same cache configuration for all other techniques. The performance difference depicted in Fig. 11 is mainly attributed to the way how GC of SSDs affect the



(a) RAID0 - Exchange
(b) RAID5 - Exchange
(c) RAID0 - Financial
(d) RAID5 - Financial

Fig. 11. Sensitivity analysis for different cache access latencies. Each bar is shown as the following order: WOW, GGC, IWC, IWC+S, and LAWC.

response time. LAWC and GGC minimize the GC impact by synchronizing GC processes in the array of SSDs.

5 Related Work

RAID combines multiple disks into a logical unit to provide fault tolerance and improve overall performance in a system. RAID uses "striping" technique. It breaks data into segments and sends each segment to independent disks, offering read/write to more than one disk at the same time to improve performance. In addition, parity in RAID provides data redundancy to improving reliability. Performance of RAID in the context of hard disk drives has been extensively studied in [31], [32], [33], [34], [35]. On the other hand, SSDs offer significance performance improvement than HDDs, however, they suffer performance variability due to internal characteristics [12], [13], [14], [15]. When SSDs are configured in RAID, the performance variability of individual SSDs becomes a major concern, because the overall performance of the SSD-based RAID could be limited by the slowest SSD. The main reason behind this performance degradation is due to uncoordinated garbage collection. Kim et al. [12], [13] proposed a coordinated GG algorithm, called GGC [12] to mitigate the performance degradation problem in SSD arrays.

WOW [16] and STOW [18] are popular write cache algorithms for the arrays of HDDs, Even if WOW [16], and STOW [18] significantly improve write performance, their work is limited only to an array of hard drives, On the contrary, our work focuses on the design of write cache for an array of SSDs for all-flash storage. LAWC considers internal SSD characteristics in the array such as multiple chips, channels, multiple cores, etc. To the best of our knowledge, no prior work has been performed on the write cache design for All Flash storage. GC needs to be carefully considered in cache management on the RAID controller for SSDs. The proposed write cache improves performance by increasing the probability of GC synchronization. When GC processes are invoked at the same time, the performance variation across SSDs is reduced, which contributes to reducing the response time. While the proposed write cache increases the probability of GC synchronization, GGC [12] guarantees it. However, GGC requires a modification to the interface (e.g., SATA) to enable the GC synchronization. Since the interface modification is not widely accepted yet, our proposed write cache is a practical solution for an array of existing commodity SSDs.

6 Conclusion

All flash array storage system becomes more and more popular with its high I/O performance and high reliability. Although they offer better performance than HDD based array, frequency of garbage collection makes SSD RAID incur higher overheads. From our study, we observed that independent garbage collection process on different SSDs contributes to the overheads in serving the incoming requests, thus resulting in a overall performance variability of the all-flash array storage.

To minimize the effect of not optimized RAID destage writes, we proposed a Layout-aware Write Cache design, which implements an asynchronous write cache algorithm with a pre-parity-computation technique. Specifically LAWC implements I/O scheduling with pre-parity computation, synchronized destage write, and two-level hybrid caching techniques. Unlike a traditional write cache, LAWC employs as many separate write queues as the number of SSDs in the array. By employing multiple write queues, an impact of GC on any write queue can be isolated from other queues. The results revealed that the synchronization of destage writes across SSDs can increase the probability of overlapped GC operations across SSDs in the array, improving the overall performance of SSD arrays, without requiring interface changes to the architecture as in GGC.

We designed and evaluated LAWC by enhancing the well regarded MSR's SSD simulator. With extensive evaluations, our conclusion is that LAWC increases the average number of synchronized GCs, when compared to a state-of-the-art RAID write cache, WOW [16]. As a result, it offers performance improvement by 89.37 and 89.94 percent for RAID-0 and RAID-5, respectively, on multi-stream storage environment, when compared to WOW. Moreover, the proposed technique is more practical than an existing GC coordination technique such as GGC because it does not require modification to the block interface.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MISP) (No. 2015R1C1A1A0152105). This research used resources of The University of Texas at San Antonio, San Antonio, TX.

References

[1] OLCF, "Oak ridge leadership computing facility," [Online]. Available: https://www.olcf.ornl.gov/
[2] ALCF, "Argonne leadership computing facility," [Online]. Available: https://www.alcf.anl.gov
[3] NERSC, "National energy research scientific computing center," [Online]. Available: https://www.nersc.gov/
[4] J. Torrellas, "Architectures for extreme-scale computing," *IEEE Comput.*, vol. 42, no. 11, pp. 28–35, Nov. 2009.
[5] Bill Harrod, "US department of energy big data and scientific discovery," (2014). [Online]. Available: http://www.exascale.org/bdec/sites/www.exascale.org.bdec/files/talk4-Harrod.pdf
[6] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1988, pp. 109–116.
[7] Y. Kim, R. Gunasekaran, G. M. Shipman, D. A. Dillow, Z. Zhang, and B. W. Settlemyer, "Workload characterization of a leadership class storage," in *Proc. Parallel Data Storage Workshop*, 2010, pp. 1–5.
[8] S. Oral, et al., "Best practices and lessons learned from deploying and operating large-scale data-centric parallel file systems," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2014, pp. 217–228.
[9] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," in *Proc. 14th Int. Conf. Architectural Support Program. Languages Operating Syst.*, 2009, pp. 229–240.
[10] J. Lee, Y. Kim, G. M. Shipman, S. Oral, F. Wang, and J. Kim, "A semi-preemptive garbage collector for solid state drives," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Apr. 2011, pp. 12–21.
[11] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding intrinsic characteristics and system implications of flash memory based solid state drives," in *Proc. 11th Int. Joint Conf. Meas. Model. Comput. Syst.*, 2009, pp. 181–192.
[12] Y. Kim, S. Oral, G. Shipman, J. Lee, D. Dillow, and F. Wang, "Harmonia: A globally coordinated garbage collector for arrays of solid-state drives," in *Proc. IEEE 27th Symp. Mass Storage Syst. Technol.*, May 2011, pp. 1–12.

[13] Y. Kim, J. Lee, S. Oral, D. Dillow, F. Wang, and G. M. Shipman, "Coordinating garbage collection for arrays of solid-state drives," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 888–901, Apr. 2014.
[14] S. Moon and A. L. N. Reddy, "Does RAID improve lifetime of SSD arrays?" *ACM Trans. Storage*, vol. 12, no. 3, pp. 11:1–11:29, Apr. 2016.
[15] T.-C. Chiueh, W. Tsao, H.-C. Sun, T.-F. Chien, A.-N. Chang, and C.-D. Chen, "Software orchestrated flash array," in *Proc. Int. Conf. Syst. Storage*, 2014, pp. 14:1–14:11.
[16] B. S. Gill and D. S. Modha, "WOW: Wise ordering for writes-combining spatial and temporal locality in non-volatile caches," in *Proc. 4th Conf. USENIX Conf. File Storage Technol.*, 2005, pp. 10–10.
[17] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proc. USENIX Annu. Tech. Conf.*, 2008, pp. 57–70.
[18] B. S. Gill, M. Ko, B. Debnath, and W. Belluomini, "STOW: A spatially and temporally optimized write caching algorithm," in *Proc. Conf. USENIX Annu. Tech. Conf.*, 2009, pp. 26–26.
[19] A. Varma and Q. Jacobson, "Destage algorithms for disk arrays with nonvolatile caches," *IEEE Trans. Comput.*, vol. 47, no. 2, pp. 228–235, Feb. 1998.
[20] T. R. Haining, "Non-volatile cache management for improving write response time with rotating magnetic media," PhD dissertation, Dept. Comnput. Sci., Univ. California, Santa Cruz, Santa Cruz, CA, USA, 2000.
[21] X. Ding, S. Jiang, and F. Chen, "A buffer cache management scheme exploiting both temporal and spatial localities," *ACM Trans. Storage*, vol. 3, no. 2, Jun. 2007, Art. no. 5.
[22] J. Menon and J. Cortney, "The architecture of a fault-tolerant cached RAID controller," in *Proc. 20th Annu. Int. Symp. Comput. Archit.*, May 1993, pp. 76–86.
[23] Y. Kim, J. Lee, S. Oral, D. Dillow, F. Wang, and G. M. Shipman, "Coordinating garbage collection for arrays of solid-state drives," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 888–901, Apr. 2014. [Online]. Available: http://dx.doi.org/10.1109/TC.2012.256
[24] S. Wu, Y. Lin, B. Mao, and H. Jiang, "GCaR: Garbage collection aware cache management with improved performance for flash-based SSDs," in *Proc. Int. Conf. Supercomputing*, 2016, pp. 28:1–28:12. [Online]. Available: http://doi.acm.org/10.1145/2925426.2926263
[25] M. K. Qureshi, M. M. Franceschini, A. Jagmohan, and L. A. Lastras, "PreSET: Improving performance of phase change memories by exploiting asymmetry in write times," in *Proc. 39th Annu. Int. Symp. Comput. Archit.*, 2012. pp. 380–391.
[26] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating server storage to SSDs: Analysis of tradeoffs," in *Proc. 4th ACM Eur. Conf. Comput. Syst.*, 2009, pp. 145–158.
[27] P. J. Shenoy and H. M. Vin, "Cello: A disk scheduling framework for next generation operating systems," in *Proc. ACM SIGMETRICS Joint Int. Conf. Meas. Model. Comput. Syst.*, 1998, pp. 44–55.
[28] Storage-Performance-Council, "OLTP trace from UMass trace repository," [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage
[29] J. Zhang, A. Sivasubramaniam, H. Franke, N. Gautan, Y. Zhang, and S. Nagar, "Synthesizing representative I/O workloads for TPC-H," in *Proc. 10th Int. Symp. High Perform. Comput. Archit.*, 2004, pp. 142–142.
[30] R. Koller and R. Rangaswami, "I/O deduplication: Utilizing content similarity to improve I/O performance," *ACM Trans. Storage*, vol. 6, no. 3, pp. 13:1–13:26, Sep. 2010. [Online]. Available: http://doi.acm.org/10.1145/1837915.1837921
[31] W. A. Burkhard and J. Menon, "Disk array storage system reliability," in *Proc. IEEE 23rd Int. Symp. Fault-Tolerance Comput.*, 1993, pp. 432–441.
[32] S. Chen and D. Towsley, "A performance evaluation of RAID architectures," *IEEE Trans. Comput.*, vol. 45, no. 10, pp. 1116–1130, Oct. 1996.
[33] M. Malhotra and K. S. Trivedi, "Reliability analysis of redundant array of inexpensive disks," *J. Parallel Distrib. Comput.*, vol. 17, pp. 146–151, 1993.
[34] R. R. Muntz and J. C. Lui, "Performance analysis of disk arrays under failure," in *Proc. 16th Very Large Data Bases Conf.*, 1990, pp. 162–173.
[35] X. Wu, J. Li, and H. Kameda, "Reliability analysis of disk array organizations by considering uncorrectable bit errors," in *Proc. 16th Symp. Reliable Distrib. Syst.*, 1997, pp. 2–9.

Kalidas Ganesh received the BE degree in electronics & communications engineering from Anna University, Chennai, in 2014 and the MS degree in electrical engineering from the University of Texas at San Antonio, in 2016. From 2015-2016, he worked at Digital System Design Lab, University of Texas at San Antonio as a research assistant on implementing performance measurement simulation environment for storage systems, before joining with BDIPlus Inc., where he is currently a Jr. R&D Product Developer. His research interests include the area of computer architecture, with a specific interest in performance analysis, evaluation, and modeling of high performance storage systems.

Youngjae Kim received the BS degree in computer science from Sogang University, Korea, in 2001, the MS degree from KAIST, in 2003, and the PhD degree in computer science and engineering from Pennsylvania State University, Pennsylvania, in 2009. He worked as a research staff member in 2009-2015 at the Oak Ridge National Laboratory, Tennessee, and as an assistant professor in the Department of Software and Computer Engineering, Ajou University, Suwon Republic of Korea in 2015-2016. Since 2016, he is in the Department of Computer Science and Engineering, Sogang University, Seoul, Republic of Korea as an assistant professor. His research interests include distributed file and storage, parallel I/O, operating systems, emerging storage technologies, and performance evaluation.

Monobrata Debnath received the BEng degree in electronics and communication from Visvesvaraya Technological University, India, in 2007 and the MS degree majoring computer engineering from the University of Texas at San Antonio, where he is currently working toward the PhD degree in electrical engineering with the concentration in computer architecture. His research interests include high performance computer architecture, network on chip, simultaneous multi-threading, and cpu performance modelling and validation.

Sungyong Park received the BS degree in computer science from Sogang University, and both the MS and PhD degrees in computer science from Syracuse University. He is a professor in the Department of Computer Science and Engineering, Sogang University, Seoul, Korea. From 1987 to 1992, he worked for LG Electronics, Korea, as a research engineer. From 1998 to 1999, he was a research scientist at Telcordia Technologies (formerly Bellcore), where he developed network management software for optical switches. His research interests include cloud computing and systems, virtualization technologies, operating systems, and embedded system software.

Junghee Lee received the BS and MS degrees in computer engineering from Seoul National University, in 2000 and 2003, respectively, and the PhD degree in electrical and computer engineering from the Georgia Institute of Technology, in 2013. From 2003-2008, he worked at Samsung Electronics on electronic system level design of mobile system-on-chip. Since 2014, he is in the Department of Electrical and Computer Engineering, University of Texas at San Antonio as an assistant professor. His research interests include architecture design of microprocessors, memory hierarchy, and storage systems for high performance computing and embedded systems.