Coordinating Garbage Collection for Arrays of Solid-state Drives

Youngjae Kim, Junghee Lee, Sarp Oral, David A. Dillow, Feiyi Wang, and Galen M. Shipman

Abstract—Although solid-state drives (SSDs) offer significant performance improvements over hard disk drives (HDDs) for a number of workloads, they can exhibit substantial variance in request latency and throughput as a result of garbage collection (GC). When GC conflicts with an I/O stream, the stream can make no forward progress until the GC cycle completes. GC cycles are scheduled by logic internal to the SSD based on several factors such as the pattern, frequency, and volume of write requests. When SSDs are used in a RAID with currently available technology, the lack of coordination of the SSD-local GC cycles amplifies this performance variance. We propose a global garbage collection (GGC) mechanism to improve response times and reduce performance variability for a RAID of SSDs. We include a high-level design of SSD-aware RAID controller and GGC-capable SSD devices and algorithms to coordinate the GGC cycles. We develop *reactive* and *proactive* GC coordination algorithms and evaluate their I/O performance and block erase counts for various workloads. Our simulations show that GC coordination by a reactive scheme improves average response time and reduces performance variability for a wide variety of enterprise workloads. For bursty, write-dominated workloads, response time was improved by 69% and performance variability was reduced by 71%. We show that a proactive GC coordination algorithm can further improve the I/O response times by up to 9% and the performance variability by up to 15%. We also observe that it could increase the lifetimes of SSDs with some workloads (e.g. Financial) by reducing the number of block erase counts by up to 79% relative to a reactive algorithm for write-dominant enterprise workloads.

Index Terms—Storage Systems, Solid-state Drives, Flash Memory, Garbage Collection, Redundant Array of Inexpensive Disks.

1 Introduction

Widely deployed in systems ranging from enterprise-scale down to embedded deployments, hard disk drives (HDDs) remain the dominant media in online and near-line storage systems. HDD manufacturers have been successful in providing a long series of improvements in storage density, which have increased the total disk capacity while bringing down the price per byte. Perpendicular recording [26] has extended this trend, but further advances will likely require new technologies, such as patterned media [41]. The changes in technology will present significant manufacturing challenges and may disrupt the economies of scale that mass production has brought to the industry.

Although storage density has seen numerous improvements, I/O performance has been increasing at a much slower pace. The improved density helps move more data onto and off the disk in a single revolution, but the largest gains have come from increasing the rotational speed—a single enterprise-class HDD can now provide up to 204 MB/s when operating at 15,000 revolutions per minute [36]. Unfortunately, HDD designers now believe it will be extremely difficult to further increase the rotational speed of the platters because of power consumption and challenging thermal dissipation issues [10], [17].

In contrast, solid state disks (SSDs) are the up-and-comer in the storage industry, with SSDs based on NAND Flash memory leading the charge. SSDs offer a number of benefits over conventional HDDs [6]: improved I/O access times, less power consumption, better resilience to operating in harsh environments with external shocks and hotter temperatures, and lighter-weight devices that help reduce the need for additional floor reinforcement in data centers. These benefits have led to several successful deployments in enterprise and high-performance computing (HPC) storage systems [2], [11], [24], [29], and the pace of adoption is likely to increase. There is also ongoing research into designing hybrid storage systems combining SSDs and HDDs to balance the benefits and costs associated with each technology [29], [16], [8].

Packaging SSDs with form factors and electrical interfaces common to HDDs permits a direct replacement in current systems. Operating systems interact with the SSDs as normal block devices, allowing system designers the ability to plug in an SSD and gain the performance benefits without a full redesign of the system. However, this interoperability comes with costs—not only is current process technology more expensive in terms of price per GB, but also NAND Flash presents different semantics from magnetic media, requiring a software translation layer between the block storage APIs and the Flash itself. This layer of abstraction can restrict the high bandwidth and low latency achievable by SSDs when presented with particular I/O access patterns.

- *Y. Kim, S. Oral, D. Dillow, G. Shipman, and F. Wang are with the National Center for Computational Sciences, Oak Ridge National Laboratory, Oak Ridge, TN 37831 USA. E-mail: ({kimy1, oralhs, gshipman, dillowda, fwang2}@ornl.gov). Y. Kim is a corresponding author.*
- *J. Lee is with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 USA. E-mail: {jlee36}@ece.gatech.edu.*

Digital Object Indentifier 10.1109/TC.2012.256

NAND Flash memory is unable to directly overwrite a storage location in the same manner as magnetic media. Once a location has been written, NAND requires an erase operation before data in that location can be changed [31]. Further complicating matters, read and write operations can be performed at a finer granularity than erase operations. Reads and writes work on pages, typically 2 to 4 KB in size, whereas groups of 64 to 128 pages called blocks must be erased as a single unit. As erase cycles are much more expensive than read/write operations, SSDs incorporate software to allow out-of-place update operations and to map sectors from the storage API to their current location in the Flash. This out-of-place update eventually requires a sweep of the storage area to find stale data and consolidate active pages in order to create blocks that can be erased. This process, known as garbage collection (GC), can block incoming requests that are mapped to Flash chips currently performing erase operations [20]. Note that SSDs are composed of multiple flash chips. The frequency and overhead of GC operations are significantly affected by random writes and updates with small request sizes [3], [22], [15]. These I/O patterns can also increase the movement of active data and incur more block erase operations [15], [9], which further slow down GC and decrease the lifetime of the NAND device.

Redundant arrays of inexpensive (or independent) disks (RAID) [34] were introduced to increase the performance and reliability of disk drive systems. RAID provides parallelism of I/O operations by combining multiple inexpensive disks, thereby achieving higher performance and robustness than a single drive. RAID has become the de facto standard for building high-performance and robust HDD-based storage systems. Hypothesizing that RAID could provide similar benefits for SSDs, we analyzed SSD-based configurations and found that a RAID consisting of commercial-off-the-shelf (COTS) SSDs was more cost-effective than a high-performance peripheral component interconnect express (PCIe) based SSD [18], [19]. However, our comprehensive evaluation also found that SSD-based RAID configurations exhibit serious bandwidth variability due to GCs of individual SSDs [18]. Note that the aggregate performance of a RAID is often dictated by the slowest drive in it. Our investigation provided empirical evidence that uncoordinated GC processes on individual SSDs are a significant contributor to the performance variability of the RAID [19]. We thus proposed *Harmonia*, a garbage collector that operates at the RAID level and globally coordinates local GC cycles for each SSD in the array. In this paper, we make the following specific contributions:

- Although the effects of unsynchronized processes on performance variability in large-scale HPC systems have been discussed [32], [35], [5], research into the effects of uncoordinated GC processes on the performance of a RAID is an open field. To our knowledge, this is the first work addressing this performance variability.
- We empirically observe that the performance of an SSD can be highly degraded by GC processes. We observe that this effect is amplified in RAID configurations without coordinated GC cycles.
- We propose a global GC (GGC) mechanism to coordinate GC cycles across a set of SSDs, thereby reducing overall performance variability. This proposal includes SSD-aware RAID controllers that implement our synchronized GC algorithms, and RAID-aware SSDs that provide information to the RAID controllers and additional functionality to allow participation in a global GC cycle.
- We propose both *reactive* and *proactive* GGC algorithms. We describe two reactive algorithm implementations that differ in the conditions under which each individual SSD in the array participates in the GGC cycle. Our proactive algorithm initiates opportunistic GGC during idle periods of the incoming I/O stream.
- We extend Microsoft Research (MSR)'s SSD simulator to implement our proposed SSD GGC algorithms for SSD RAID storage system. Using industry-standard workloads, our experiments show that GGC can reduce overall latency by up to 15% as well as provide a significant reduction in the performance variability when compared with the standard uncoordinated GC of individual drives in the RAID.

The reminder 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 3. Section 4 introduces our proposed improved RAID and SSD controller designs, as well as the globally synchronized garbage collection algorithms. In Section 5 and 6 present simulation results of our proposed GC coordination algorithms and their comparison on I/O performance and block cleaning efficiency for various workloads. Then, we conclude in Section 7.

2 Background

Flash memory–based storage devices provide quite low access times relative to magnetic media— on average, a read operation requires 0.025 ms and a write requires approximately 0.200 ms. However, these low latencies come at a substantial cost in terms of access semantics. Whereas magnetic media and volatile memories are able to overwrite existing data with a write operation, Flash systems require an *erase* operation before new data may be written to a previously used location [31]. These erase operations are expensive compared with read and write operations—on average, each erase takes 1.5 ms to complete, during which time the affected device is unable to perform other tasks. Further complicating matters, the granularity of the erase operation is a significant multiple of the page size used for reads and writes.

Software called the Flash Translation Layer (FTL) translates the logical address for each request to the

physical device and address where the data actually resides. The FTL emulates an HDD by performing out-of-place updates, allowing the device to avoid incurring the cost of an erase each time a page needs to be overwritten. The mapping from logical to physical addresses is often stored in a small amount of fast SRAM, and the mapping granularity—the size of each contiguous chunk of data—can vary for each device. Many different FTL algorithms have been proposed and studied [23], [9], [21], as well as versions that improve write buffering [15] and buffer management [33]. Recently, demand-based FTL [9] was proposed to overcome the limitations of page-based FTLs by using the temporal locality of workloads to reduce the number of mappings stored in the SRAM.

Although the out-of-place update provided by the FTL avoids a highly expensive read-erase-modify-write cycle when writing to a block containing existing data, the number of free blocks available for updates decreases over time as they are consumed. To replenish this pool, the SSD will engage a process known as GC to find blocks that contain a mix of valid and invalid, or stale, data. The valid data from these blocks is consolidated into a smaller number of blocks, and the newly available blocks are erased to make them ready for future write or update operations.

Current generations of SSDs use a wide variety of algorithms and policies for GC. Examination and analysis of these algorithms is difficult however as most are vendor-specific and highly proprietary. Feng Chen et al. [3] empirically observed that GC activity is directly correlated with the frequency of write operations, amount of data written, and free space on the SSD. The increased queue delay during GC can significantly impede both read and write performance [9]. Random write workloads often have the most negative impact on overall performance [22], [15], although the overhead of GC activity is highly dependent upon the characteristics of the workload presented. Based on these properties, Ningfang Mi et al [15] worked to rearrange and coalesce requests to present a more sequential workload. Others have attempted to improve the design of FTLs to minimize the overhead incurred by GC [21], [23], [9].

Despite this effort, GC-induced performance variability remains a significant problem for SSDs. This variability can be exacerbated by stalled requests targeting Flash devices with ongoing GC; these requests must be placed into a queue and scheduled for service once the GC has completed. This high variability reduces the *performance robustness* of the system, as the I/O bandwidth and system response time cannot be guaranteed. Providing a robust system that operates within specified boundaries is as important as achieving a high-performance environment. Particularly with a bursty request stream with many write I/Os, stalls due to GC can significantly degrade the performance of an SSD, and this degradation can be amplified when multiple SSDs are used in a single RAID configuration.

In the next section, we show that GC-induced performance variability can occur in individual SSDs and their RAID configurations for various workloads, and per-drive bandwidth can decrease as the number of SSDs in the RAID configuration increases.

3 Performance Degradation in SSDs

In this section, we study the pathological behavior of individual SSDs and SSD arrays and empirically observe that individual SSD's bandwidth can drop sharply due to internal GC, and the effect of GC can be worse for SSD arrays. We perform a series of experiments using various configurations of SSDs and RAID controllers.

3.1 Experimental Setup

All experiments are performed on a single server with 24 GB of RAM and an Intel Xeon Quad Core 2.93GHz CPU [13] running Linux with Lustre-patched 2.6.18-128 kernel. The *noop* I/O scheduler that implements FIFO (first in, first out) queuing is used. The testbed has seven 8x PCIe slots, and two are populated with LSI MegaRAID SAS 9260-8i KIT PCIe RAID adapters [25], each of which can support up to eight SATA drives.

Label	SSD(M)	SSD(S)
Vendor	Super-Talent	Intel
Type	MLC	SLC
Interface	SATA-II	SATA-II
Cap. (GB)	120	64
Erase (#)	10–100K	100K–1M

TABLE 1
Device characteristics.

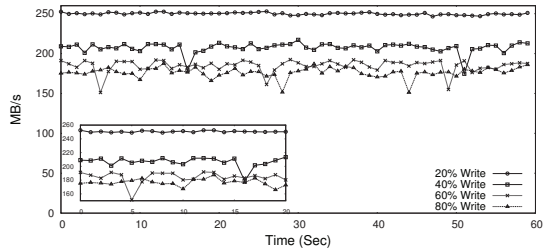
We examine two types of SSDs, Super Talent 128 GB FTM28GX25H SSD [40] and Intel 64 GB SSDSA2SH064G101 SSD [14] SSD [40] as the representative devices for multi-level cell (MLC) and single-level cell (SLC) SSDs respectively. Their specification are detailed in Table 1.

We use an in-house benchmark tool that uses the *libaio* asynchronous I/O library on Linux [18]. The libaio provides an interface that can submit one or more I/O requests using a system call, *iosubmit()*, without waiting for I/O completion. It also can perform reads and writes on raw block devices. We use *O-DIRECT* and *O-SYNC* flags in the file *open()* call to directly measure device performance by bypassing the OS I/O buffer cache.

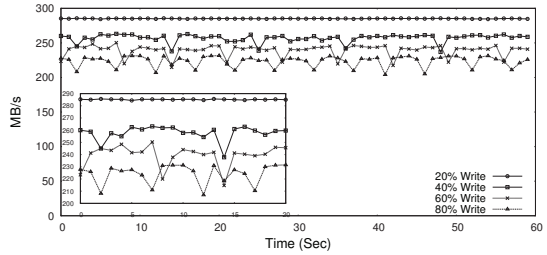
Workload	W1	W2	W3
Request size	313 KB	1.25 MB	1.87 MB
Queue depth	64	64	64
I/O access pattern	Random		
R/W ratio	Varied (20-80%)		

TABLE 2
Workload Descriptions. Note that singe SSD experiments uses *W1* workloads, and multiple SSD experiments in RAID settings use *W2* (for four SSDs) and *W3* (for six SSDs) workloads.

Table 2 presents three different synthetic workloads that we use to compare performance and bandwidth variability. A high queue depth (number of outstanding requests in the I/O queue) is used to observe the impact

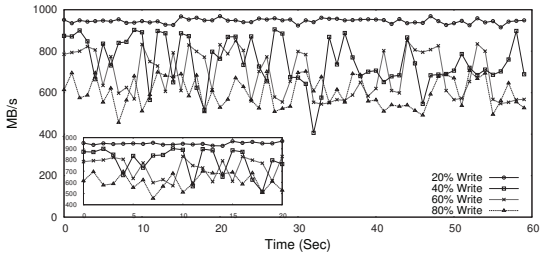


(a) Time-series analysis for SSD(M)

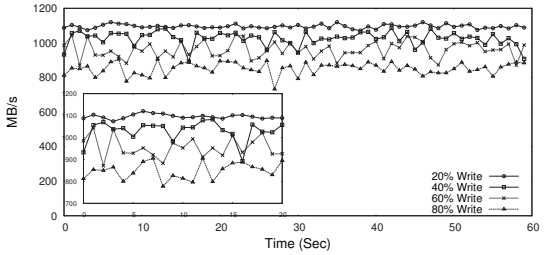


(b) Time-series analysis for SSD(S)

Fig. 1. Pathological behavior of individual SSDs.



(a) Time-series analysis for RAID(M)



(b) Time-series analysis for RAID(S)

Fig. 2. Pathological behavior of RAID of SSDs.

Type	Metric	Write (%) in Workload			
		80	60	40	20
SSD(M)	avg	176.4	184.8	207.4	249.9
	stddev	6.37	7.88	6.73	1.42
	Norm. CV	6.35	7.50	5.71	1
SSD(S)	avg	223.5	239.3	257.1	285.1
	stddev	7.96	8.38	5.86	0.28
	Norm. CV	36.3	35.7	23.2	1

TABLE 3
Original values and variability anlaysis for Figure 1(a)(b). CV values are normalized with respect to the values of the corresponding drives for 20%.

Type	Metric	Write (%) in Workload			
		80	60	40	20
RAID(M)	avg	601.2	689.6	751.2	945.7
	stddev	72.32	110.5	113.94	11.14
	Norm. CV	21.2	28.2	26.7	2.07
RAID(S)	avg	851.5	961.2	1026.1	1095.2
	stddev	34.98	46.37	40.38	11.39
	Norm. CV	41.8	49.1	40.1	10.6

TABLE 4
Original values and variability anlaysis for Figure 2(a)(b). CV values are normalized with respect to the values of the corresponding drives for 20% in Table 3.

of GC in the time domain. We vary the percentage of writes in workloads between 20% and 80% in increasing steps of 20% and their I/O access patterns are random. I/O bandwidth is measured at one second intervals. we use different request sizes for different array configurations so that the I/O demand on every drive should be the same irregardless of storage configuration.

Also we use a bigger request size than the full stripe size (stripe size x Number of SSDs) so that it should make imbalanced I/O loads on the arrays of SSDs. For example, for W2 workload, the request size is 1.25MB whereas, a full stripe request would requires multiple of 1MB (=256KB $\times$ 4) for the array of four SSDs. Similarily we use a 1.87 MB request size for the array of six SSDs. The value of 1.87 MB is obtained by setting a baseline request of 1.25 MB for the four-SSD array and scaling up the request size based on the increased number of SSDs in the array. Similarly, the request size is scaled down for evaluating one SSD. Note that for the array of four SSDs, one of the four SSDs will receive two 256 KB striped requests, whereas others will have one 256 KB striped request.

3.2 Pathological Performance Behavior

Individual SSDs: We first discuss the performance variability of individual SSDs in Figure 1. From the figure, we observe that as we increase the number of writes in the workloads from 20% to 80%, the SSD(M)'s and SSD(S)'s I/O throughput decreases by 41% and 28%, respectively. We also observe that increasing write percentage of workload can increase bandwidth fluctuation. We use coefficient of variation (CV)[1] values to compare the significance of the bandwidth variability for different workloads. Table 3 shows normalized CV values with respect to the CV value of the 20% write workload. For example, CV values for the workloads with more than 40% writes are around 6-7 times higher than the CV value for the workload with 20% writes.

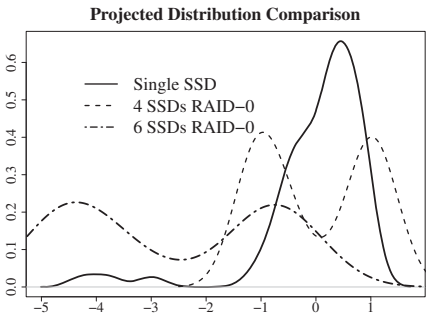
SSD Arrays: We extend our experiments to arrays of SSDs. RAID configuration for the SSD array is shown in Table 5. RAID(M) and RAID(S) designate the RAID configurations of SSD(M) and SSD(S) respectively.

RIAD config.	RAID 0 of 6 SSDs with 256KB stripe size
Cache	Write Through cache, Read ahead diabled

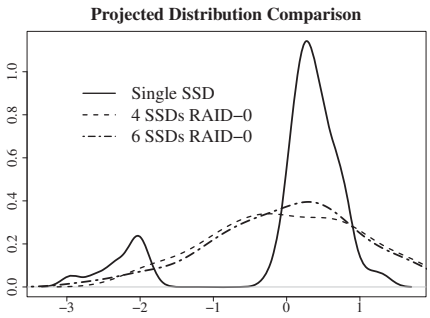
TABLE 5
RAID Settings.

Figure 2 shows the results of our experiments for RAID(M) and RAID(S) sets using the *W3* workloads in Table 2. In Figure 2(a), we see that for the 80% write-dominant I/O workload, the RAID(M) I/O throughput

1. Coefficient of variation (C_v) is a normalized measure of dispersion of a probability distribution, that is, $C_v = \frac{\sigma}{\mu}$.



(a) SSD(M) vs. RAID-0 of 4, 6 SSD(M)s



(b) SSD(S) vs. RAID-0 of 4, 6 SSD(S)s

Fig. 3. Throughput variability comparison for SSD RAIDs with increasing number of drives in the array for a workload of 60% writes. Y-axis represents normalized frequency.

can drop below the peak performance (about 700 MBs/) quite often. The I/O throughput drops below 600 MB/s at the 3rd second and then drops further to below 450 MB/s in the next several seconds. Overall, RAID(S) shows higher bandwidth than RAID(M) (referring to Figure 2(a)(b)), with a similar variance for all workloads we examine, because RAID(S) is composed of SLC SSDs and RAID(M) is composed of MLC SSDs.

We also observe that an increased workload write percentage can result in increased bandwidth fluctuation. Referring to CV values for the workloads with more than 40% writes in Table 4, we have similar observations as we learn from Table 3: CV values are much higher for the workloads with more than 40% writes than for the workload with 20% writes.

In the following sections, we provide more experimental results on performance variability with regard to both the number of SSDs in the array as well as the varying RAID levels.

3.2.1 Performance Variability with the Number of SSDs

In this experiment, we measure I/O bandwidths of SSD arrays by increasing the number of drives in the array for a given workload. Figure 3 presents performance variability for a single SSD and SSD arrays with RAID-0 of four and six SSDs for 60% write workloads as specified in Table 2.

We use normalized Z score to visually compare the distribution trend. For this, we measure I/O bandwidths every second while running the workloads, and collect bandwidth data every second for each run of each configuration. Then, each bandwidth data is normalized with a Z score ($\frac{X-\mu}{\delta}$) and each density function is drawn with curve fitting.

In Figure 3 we can see that the lines for RAIDs of four SSDs and six SSDs both show more spread then the single SSDs. Note that the wider the curve is shaped, the higher its performance variability is. Or, in other words, the tighter the distribution (e.g., minimal spread at the tails with a single spike at the center), the less variability it exhibits in terms of throughput. Also we observe that the average bandwidth of SSD array does not scale linearly as we increase the number of SSDs in the RAID-0 array. For example, for the experiments of Figure 3, a single SSD shows 184.8MB/s whereas four SSDs show 689.7MB/s and six SSDs shows 847.4MB/s. In theory, and if it can be scaled linearly with the number of SSDs, four SSDs and six SSDs are expected to offer about 740MB/s and 1108MB/s. Our conjecture is that imbalanced I/O workloads on the SSD array followed by uncoordinated GC operations on individual SSDs are increasing performance variability and contributing to the large drop in bandwidth throughput.

Moreover, we observe that the performance variability of RAID sets comprising MLC SSDs does not scale as well as that of sets of SLC SSDs. As seen in Figure 3(b), there is not a significant difference between four and six SLC SSDs in the RAID set, unlike the MLC RAID sets shown in Figure 3(a). We believe this variation to be a result of the inherent higher variability in response times of MLC SSDs.

Per-drive bandwidth: We calculate a per-drive bandwidth for a RAID of N SSDs ($N \geq 1$) by dividing the average bandwidth observed by N under the assumption that the I/O loads to storage are balanced across the SSDs in a RAID. We observe that the bandwidth can drop by up to 24% and 20%, respectively, for six RAIDs of SSD(M)s and SSD(S) compared with the bandwidths of their single SSDs.

3.2.2 Performance Variability for various RAID Levels

In this section, we investigate the correlation between various RAID levels and performance variability. In particular, we evaluate RAID-5 and 6 configurations of SSDs against a RAID-0 SSD configuration.

For these experiments, we use 6 SSDs for all RAID levels. We note that the number of data disks changes as dictated by each specific RAID configuration.

Table 6 details our results in terms of average bandwidth (MB/s) and standard deviation of the observed throughput. As can be seen in Table 6, RAID-5 has a lower standard deviation than RAID-0; similarly, RAID-6 has a lower standard deviation than RAID-5. However, standard deviation alone is not a meaningful metric for assessing performance variability. An appropriate metric for comparing bandwidth variability should take into

Type	RAID-0	RAID-5	RAID-6
RAID(M)	847.4 (60.50)	644.1 (56.16)	479.8 (45.53)
RAID(S)	1147.1 (72.02)	793.7 (53.14)	686.7 (49.44)

TABLE 6
Performance variability of various SSD RAID configuration in terms of average bandwidth (MB/s) and standard deviation of observed bandwidth (in parentheses).

account both mean and standard deviation values. We calculate the CV values with the values presented in Table 6 and plot them in Figure 4.

Figure 4 illustrates the normalized coefficient of variation in bandwidth for RAID-0, RAID-5, and RAID-6 configurations. Results in Figure 4 are normalized with respect to the coefficient of variation for RAID-0 results. The terms 'x' and 'y' (x, y) in the legend represent the number of data drives and the number of parity drives, respectively. As illustrated in Figure 4, RAID-5 and RAID-6 configurations demonstrate higher performance variability relative to the RAID-0 configuration. The RAID-6 configuration for both cases presents the highest performance variability in our experiments.

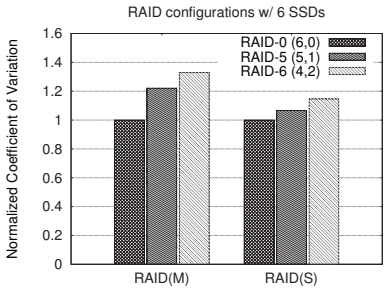


Fig. 4. Performance variability and RAID levels.

RAID-5 configuration has one parity drive and RAID-6 has two parity drives. The extra parity drives in these RAID configurations compared to the RAID-0 setup provide extra reliability with additional associated parity calculation overheads and additional write operations (to the parity drives for the parity update blocks). Our conjecture is that these extra parity calculations and the associated parity update write operations (one extra write operation for RAID-5 and two extra write operations for the RAID-6 configurations, respectively) could increase the observed bandwidth variability.

As we have seen from benchmark results with real SSD and RAID components, there are limitations for current SSD and RAID controller technologies; GC processes per individual SSD are local and they are not coordinated. RAID controllers are not aware of any ongoing GC processes at SSDs; therefore, there is no coordination at the RAID controller level. This lack of coordination causes individual GC processes per SSD to execute independently, resulting in aggregate performance degradation and response time variability at the RAID level. In the next section, we present our design for the SSD-based RAID storage system incorporating the GC coordination mechanism, and our proposed GGC algorithms. We also discuss our implementation for GGC algorithms.

4 GC Coordination for SSD Arrays

In a RAID set of SSDs, the aggregate RAID performance is limited by the slowest component of the array. Our empirical results show that uncoordinated GC can be the major culprit behind these temporary slowdowns on individual SSDs. In this section, we present a solution to this problem to mitigate the performance degradation of SSD RAID sets.

4.1 Coordinated Garbage Collection

Figure 5 depicts conceptual timings of GC processes for a given SSD array, with time on the horizontal dimension. The time line is divided into windows (A through G) as the array transitions from peak to degraded performance as a result of local GC processes. Peak performance at the RAID level is achieved when there is no active GC process on any SSD. Degraded performance occurs when an I/O operation spans even a single device with an active GC process. Assuming full stripe operations, the RAID in Figure 5(a) achieves its peak performance only in time windows B, D, and F. The array is limited to degraded performance in windows A and G because of multiple devices performing GC, and in windows C and E because of a single device with active GC.

Figure 5(b) shows the desired benefits of our proposed mechanism to coordinate and synchronize the local GC processes of each SSD. We call this proposed mechanism Global Garbage Collection (GGC). In this mechanism, GC processes are shifted in time to allow longer windows of peak performance from the RAID. By advancing the GC process in Figure 5(a) window C to occur simultaneously with the other processes in window A, we can eliminate one source of degraded performance. Similarly, delaying the GC in window E to window G allows more opportunity for the RAID controller to issue operations that do not span devices with active GC processes.

4.2 Architectural Design

We argue that the aggregate performance degradation induced by uncoordinated GC processes can be resolved by providing the following:

1) A RAID controller designed to enable global coordination of garbage collection when used with SSDs supporting that capability. This optimized RAID controller will be referred to as an *SSD optimized RAID controller* (O-RAID).
2) An SSD designed for participating in a globally coordinated garbage collection process in a O-RAID. This new SSD will be referred to as *GGC optimized SSD* (O-SSD).
3) A set of algorithms to perform a globally coordinated GC process on a given SSD array comprising an O-RAID and multiple O-SSD devices.

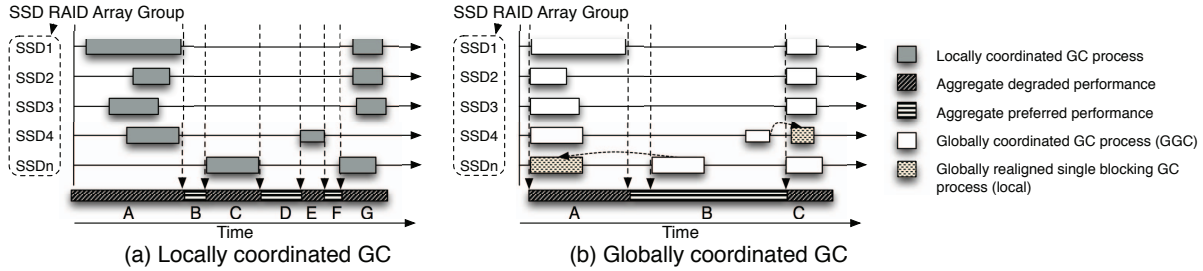


Fig. 5. Effect of GGC: (a) presents locally coordinated GC processes for an unoptimized RAID array; (b) presents globally coordinated GC processes for a GGC optimized RAID array.

4) Extension of storage protocols such as SATA and SCSI for controlling the additional capabilities of O-SSD devices.

Furthermore, we observe the following conventions and constraints in this architecture: the O-RAID operates at the global level with global knowledge obtained from all O-SSDs, whereas an O-SSD has only local knowledge in the form of internal fragmentation level, number of available free blocks, and other similar information used to determine when to start its GC process.

While the GGC process can provide maximum effectiveness only when all SSD devices in a given RAID set support the GGC capability, reduced benefits may still be obtained if only a subset of devices offers support.

4.3 Coordinated Garbage Collection Algorithms

Coordination is essential to achieving the performance benefits of GGC. Among multiple ways of GGC implementation, we discuss two possible GGC algorithms and our implementations of each algorithm in this section.

4.3.1 Reactive GGC Algorithms

In reactive GGC a specialized O-RAID reacts to notification by the O-SSD that garbage collection is imminent, attempting to coordinate other O-SSD's garbage collection processes. Reactive GGC is fairly simple to implement, requiring minimal information from O-SSDs and simple decision-making mechanisms based on thresholds. In this GGC algorithm, an O-SSD notifies the O-RAID when it has reached an internal threshold indicating that it will soon need to initiate a GC process. This communication may also provide additional information to the O-RAID, such as an estimate of how much data can be written before a hard threshold is reached and an uncoordinated GC process must be initiated.

Once the O-RAID has been notified, it will direct each O-SSD to initiate a GC process. The O-RAID can optionally delay this initiation in order to allow more O-SSDs to register their need of GC, or to potentially find a more optimal point in the request stream for the GC cycle to begin. If the O-RAID chooses to delay the GC cycle, it can use the additional information from the notification to avoid triggering uncoordinated GC. With this scheme, the O-SSD will delay its GC cycle until it reaches a hard threshold at which it must begin a GC cycle. The O-SSD's communication to the O-RAID of the need for GC is advisory in nature, and a lack of response from the O-RAID will not prevent the O-SSD from locally performing needed GC. The reactive soft-limit algorithm can be implemented in different ways. In this paper, we present and evaluate two possible implementations, which are $Reactive_{inclusive}$ and $Reactive_{selective}$.

$Reactive_{inclusive}$: In reactive GGC algorithms, an SSD in the array that reaches an internal GC threshold issues a GGC request message to the O-RAID. Upon receipt of this message, the O-RAID schedules a GGC event by iterating over all connected devices and issue a *FORCE GC* event to each. Upon receipt of the *FORCE GC* event, a local GC process is triggered to clean the stale/invalid elements until the number of free blocks exceeds an internal threshold. In our $Reactive_{inclusive}$ implementation, all SSDs in the array have to participate in GC coordination irrespective of their internal status (eg., the number of free blocks). As every SSD is agnostic of other SSDs' internal status this implementation could aggressively trigger GGCs on SSDS and could shorten the lifespan of SSDs.

$Reactive_{selective}$: In order to improve the lifetime concern of $Reactive_{inclusive}$, we developed $Reactive_{selective}$. This implementation reduces the number of GC operations on individual SSDs. Unlike $Reactive_{inclusive}$, in $Reactive_{selective}$, not all of the SSDs in the array have to participate in GGC events. Thus, this algorithm improves SSD lifespans compared to $Reactive_{inclusive}$. However, $Reactive_{selective}$ requires more complicated protocol design and implementation than $Reactive_{inclusive}$.

The implementation of the $Reactive_{selective}$ is composed of two phases: (1) a phase for GC registration, in which O-SSDs are registered to participate in the GC coordination process, and (2) the second phase for GC coordination by participating O-SSDs. For implementing the $Reactive_{selective}$, soft (T_{soft}) and hard (T_{hard}) thresholds are defined. T_{hard} is the minimum number of free blocks required to sustain the system operation. T_{soft} (where $T_{soft} > T_{hard}$) is the number of free blocks that indicates GC is needed shortly but not urgently. A new event of *PARTICIPATE GGC* needs to be defined in addition to the *FORCE GC* event. *PARTICIPATE GGC* is an event by which O-SSDs request their enrollment for GC coordination to O-RAID. If any O-SSD reaches T_{soft}, it is registered in an O-RAID maintained list by issuing an event of *PARTICIPATE GGC* to the O-RAID. The first

O-SSD that reaches T_{hard} becomes a GC coordinator and notifies O-RAID for GC coordination. Then O-RAID issues *FORCE GC* events to all registered O-SSDs.

4.3.2 Proactive GGC Algorithms

In proactive GGC, O-RAID actively monitors O-SSDs and attempts to optimize scheduling of GC processing to minimize the impact on I/O workloads. In particular, O-SSD's internal information and idle times in workloads can be used to develop proactive GGC. Below, we discuss two possible ways of realizing a proactive GGC mechanism, which we call *proactive idle* and *proactive soft-limit*.

In the *proactive idle* scheme, the O-RAID identifies points in the I/O stream that are expected to have extended idle periods and initiates a GGC cycle during those lulls in activity. Idle time is defined as the status in which there is no SSD in the array that is servicing requests or has pending requests. The challenge of this approach is to identify idle times in workloads. Much research has been done to develop techniques to detect the idle times. In the following section we describe the use of one of the popular idle time detection heuristics in our idle time based algorithms.

Another implementable scheme is *proactive soft-limit*. In this scheme, the O-RAID periodically collects GC state information from each O-SSD. This information collection can be initiated by the O-RAID via a pull mechanism, or each O-SSD can periodically push the information to the O-RAID. A combination of both reactive and proactive GGC may also be used. The O-RAID uses the collected information to determine when each O-SSD has reached a state in which a GGC cycle would be beneficial, and attempts to find an optimal point in the I/O stream to initiate it. State information useful for determining the need for a GC cycle includes, but is not limited to (i) Internal fragmentation level (ratio of free to used erase blocks), (ii) Number of free erase blocks available, (iii) ECC correctable error rate on reads, etc.

In this paper, we have only implemented and evaluated the idle time based proactive scheme, which we call $Proactive_{idle}$. However, it is important to note that *proactive soft-limit* and *proactive idle* are not mutually exclusive; both may be used concurrently.

$Proactive_{idle}$: I/O workloads are known to exhibit periods of idleness between bursts of requests [27], providing opportunities for GC coordination. Much research has gone into developing techniques to identify and utilize these idle periods [27], [28], [4]. Specifically, Mi et al. categorized workloads based on idle periods into tail-based, body-based, and body-tail-based [27]. We use idle times to trigger global coordination of GC events.

We extend the dual threshold scheme, $Reactive_{selective}$ to trigger GC coordinations during idle times. When idle time is detected, a *FORCE GC* event is sent to all the registered O-SSDs in the array for GC coordination. Receiving the FORCE GC event, an O-SSD starts generating free blocks until the number of free blocks reaches the total reserved free blocks (i.e. maximum number of free blocks) or until there is an incoming I/O request in the queue. To implement the idle time detection algorithm, we use a heuristic on-line algorithm presented in [4]. In this method the O-RAID checks the status periodically. If an idle time is detected longer than a pre-defined threshold, then it starts a coordinated GC event by issuing *FORCE GC* commands to O-SSDs.

5 Performance Study of SSD RAID

5.1 GGC Simulator Design

To explore our proposed GGC design and GGC algorithms, we extend the SSD simulator developed by Microsoft Research [1] and evaluate our proposed GGC-optimized RAID. In GGC algorithms, the initiator SSD in the array is set as the GC global coordinator. Unless individual SSDs receive an event of *FORCE GC*, they operate as normal (without GGC coordination). Otherwise they are forced to start the GGC process. Note that in our reactive algorithm, the first SSD that reaches the hard threshold is set as a GGC coordinator and a *FORCE GC* event can be issued from it. The O-RAID receives a GGC request message from the initiator SSD. Upon receiving this message, O-RAID prepares to schedule a GGC event. It iterates over all connected devices and for each device issues a *FORCED GC* event. Upon receipt of the *FORCED GC* event a local GC process will be triggered to clean the stale/invalid elements until the number of free blocks exceeds an internal threshold. We implement both reactive and proactive GGC algorithms (their implementations are described in Section 4). In this section, we present the results for $Reactive_{inclusive}$ and in the following Section 6, we present the results of comparing reactive and proactive GGC schemes.

5.2 Experimental Setup and Workloads

For the baseline RAID environment, we configure an SSD RAID simulator to analyze a RAID-0 array. The SSD simulator is configured to simulate eight SSDs in RAID-0 using 4 KB stripe. In the baseline configuration, there is no GC coordination among SSDs in the array. Each SSD in the simulator is configured as the specifications shown in Table 7.

SSD configuration	
Total capacity	32 GB
Reserved free blocks	15 %
Minimum free blocks	5 %
Cleaning policy	Greedy
Flash chip elements	64
Planes per package	4
Blocks per plane	512
Pages per block	64
Page size	4 KB

Flash Operational Latency	
Page read	0.025 ms
Page write	0.200 ms
Block erase	1.5 ms

TABLE 7
SSD model parameters.

Simulation "Warm-up": Prior to collecting performance data from the simulator, we fill the entire space on each SSD in the simulator with valid data by marking the flag

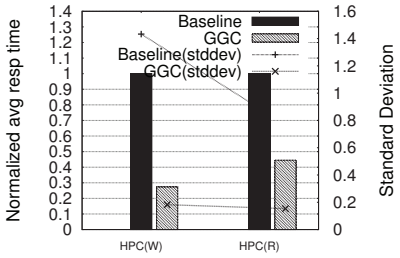
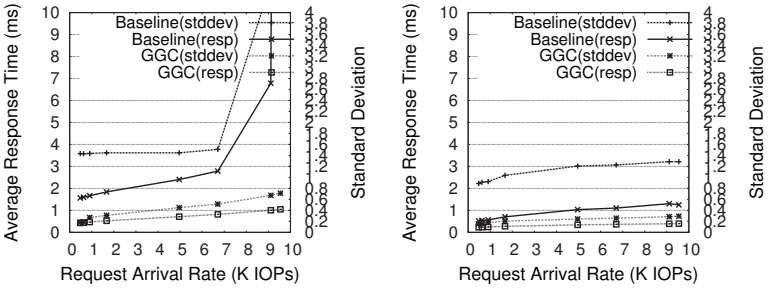


Fig. 6. Results with HPC workloads. Avg. Resp. Time (ms) for baseline={1.57,0.29}



(a) Write-dominant (80% write) (b) Read-dominant (20% write)

Fig. 7. Results with changing write percentage of workloads.

that presents the status of the page on every OOB (out-of-band) in the simulator as 1s. This ensures that GC is required on the SSD devices as new write requests arrive during the experimental run.

We use a wide spectrum of workloads from industry and research sources to evaluate the performance of our GGC schemes. We use a mixture of HPC-like workloads and realistic enterprise-scale workloads. This broad spectrum is chosen to obtain a more realistic view of the benefits of coordinated garbage collection. As described in Table 8, these workloads include both read and write-dominated traces.

For HPC-like workloads, we choose read and write and bursty workloads whose characteristics are described in Table 8. HPC(W) is a write-dominated (80%) workload that represents I/O patterns in HPC systems as they periodically write checkpoint states and large result files during their calculations [37], [7], [30]. HPC(R) is a read-dominated (80%) workload that represents heavy read patterns of HPC environments [43]. For enterprise-scale realistic workloads, five commercial I/O traces are used. We use write-dominant I/O traces from an online transaction processing application known as Financial trace [39] and TPC-C [42] made available by the Storage Performance Council (running at a financial institution) and from Cello99 [38], a disk access trace collected from a time-sharing server exhibiting significant writes (running the HP-UX operating system at Hewlett-Packard Laboratories). We also examine two read-dominant workloads. TPC-H [44] is a disk I/O trace collected from an online analytical processing application examining large volumes of data to execute complex database queries. Also, we consider e-mail server workloads referred to as Openmail [12]. TPC-C [42] is also used.

Workloads	Req. Size (KB)	Read (%)	Arrival (IOP/s)
HPC(W)	510.53	20.12	476.50
HPC(R)	510.53	80.08	476.50
Financial	7.09	18.92	47.19
TPC-C	7.06	20.50	388.32
Cello	7.06	19.63	74.24
TPC-H	31.62	91.80	172.73
Openmail	9.49	63.30	846.62

TABLE 8
Descriptions of HPC-like and Enterprise Workloads.

Although the device service time captures the overhead of GC and the device's internal bus contention, it does not include queuing delays for requests pending in the I/O driver queues. Additionally, using an average service time loses information about the variance of the individual response times. In this study, we use (1) the response time measured at the block device queue (I/O service time) and (2) the variance of the measured response times. The I/O service time captures the sum of the device service time and the additional time spent waiting for the device to begin servicing the request.

5.3 Results

Figure 6 shows the average response time of the GGC-enhanced RAID compared with the baseline SSD RAID without GC coordination for HPC-like workloads. We note a 60% reduction in response time for the *HPC(R)* read-dominated load and a 71% reduction for the *HPC(W)* write-dominated load. Also, we observe that GGC improves the variance of response times of the storage system for both HPC(W) and HPC(R) workloads.

In order to exploit a wide range of workloads, we vary the request arrival rates of the HPC workloads. Figure 7(a) shows that the baseline configuration has high response times when the workload is write-intensive (80% writes). In addition, there is a very large gradient in the response time and variability as the arrive rate increases. This behavior does not provide a robust system response. In contrast, our GGC scheme exhibits lower average response times than the baseline and a more gradual increase in variability. This confirms that GGC can help deliver robust and stable system performance. For read-dominated workloads such as those in Figure 7(b), GGC continues to deliver improved performance and system robustness.

While experiments presented in previous paragraphs are performed with eight SSDs in the RAID set, we also investigate how the number of devices in the array affects the performance.

Figure 8 compares the average response time under the *HPC(W)* workload as the number of SSDs configured in the RAID set is varied. As expected, both configurations improve their performance as the number of SSDs increases. However, GGC maintains a performance edge over the baseline throughout the experiment. At two SSDs, the baseline response time is 2.7 times longer than GGC, and the margin grows to 3.2 times as we expand

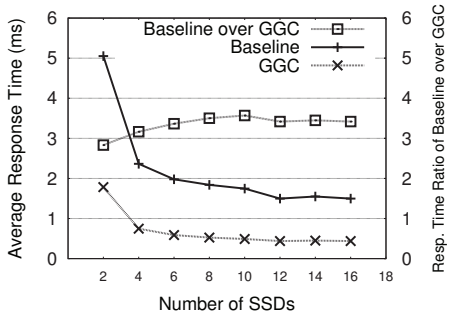


Fig. 8. Results with varing the number of drives in the array.

the RAID set to 18 SSDs. Interestingly the baseline requires eight SSDs to provide a response time equivalent to that delivered by two devices using GGC. Even with 18 devices, the baseline performs 184% worse than GGC using only 4 devices.

We further analyze GGC for enterprise-scale workloads. Figure 9 presents the results for enterprise-scale workloads. We can observe GGC not only improves average response times by 10% but also enhances the robustness and predictability of the RAID set of SSDs. However the improvement by GC coordination for the enterprise-scale workloads is smaller compared with HPC-like workloads. It is mainly because (i) HPC workloads have much higher I/O demands than Enterprise workloads (refer to Table 8), and (ii) large requests in the HPC workloads are more frequently conflict with GC invocation of drives, increasing the I/O response times.

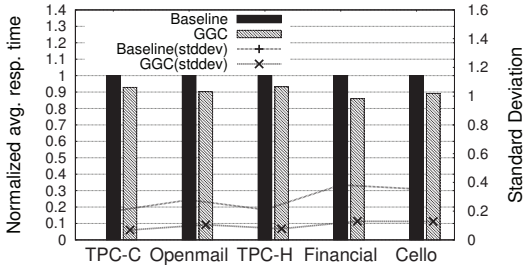


Fig. 9. Results with various enterprise-scale workloads. Note that response times and standard deviations are normalized with respect to a baseline. Avg. resp. times (ms) for baseline={0.16, 0.20, 0.17, 0.27, 0.30}

It could be thought that large requests are more evenly striped over the drives than small requests and every drive could trigger GC at the same time, however, this is only true for an ideal situation where all drives start empty and do writes with full stripes on entire drives in the array over and over. However, in real production systems, the ideal situation is not likely to occur because of meta data operations that are mostly small writes, and I/O operations not being fully striped across the drives in the RAID set. Our experiments are configured to closely mimic this behavior. Before running an experiment, SSDs are filled with random small block size data. This way, every drive has a different internal status before our benchmark runs. This ensures that we have a closer representation of a real life scenario and every flash drive in the RAID set could trigger GC at different times, even if we perfectly stripe the requests over the drives in the array.

We conduct a detailed analysis of the impact of GGC on device response times and GC invocations of individual SSDs in the RAID set. Figure 10 illustrates a set of consecutive requests serviced by two of the eight SSD devices in our simulated RAID.

The response time for each request is captured during a 300 ms interval in the *HPC(W)* workload by both the baseline and our GGC scheme. As clearly indicated by Figure 10, the baseline incurs more frequent overhead from GC, which results in larger latencies than GGC. The overall RAID response latency is a function of the convolution of the response time of each SSD in the array and is determined by the slowest device. In Figure 10(b), we clearly see fewer spikes than in the baseline without GGC. The total number of GC processes invoked is the same between the two approaches, however, many GC operations are synchronized in GGC compared with the baseline where GC operations are not synchronized. Also of note is that each SSD is composed of multiple packages. When GC is not coordinated inside SSDs, each package can trigger GC independently. By further forcing GC coordination across the packages, we could achieve significantly less aggregate GC overhead in GGC-enabled SSD RAID sets.

6 Performance Study of Various GGC Algorithms at SSD RAID

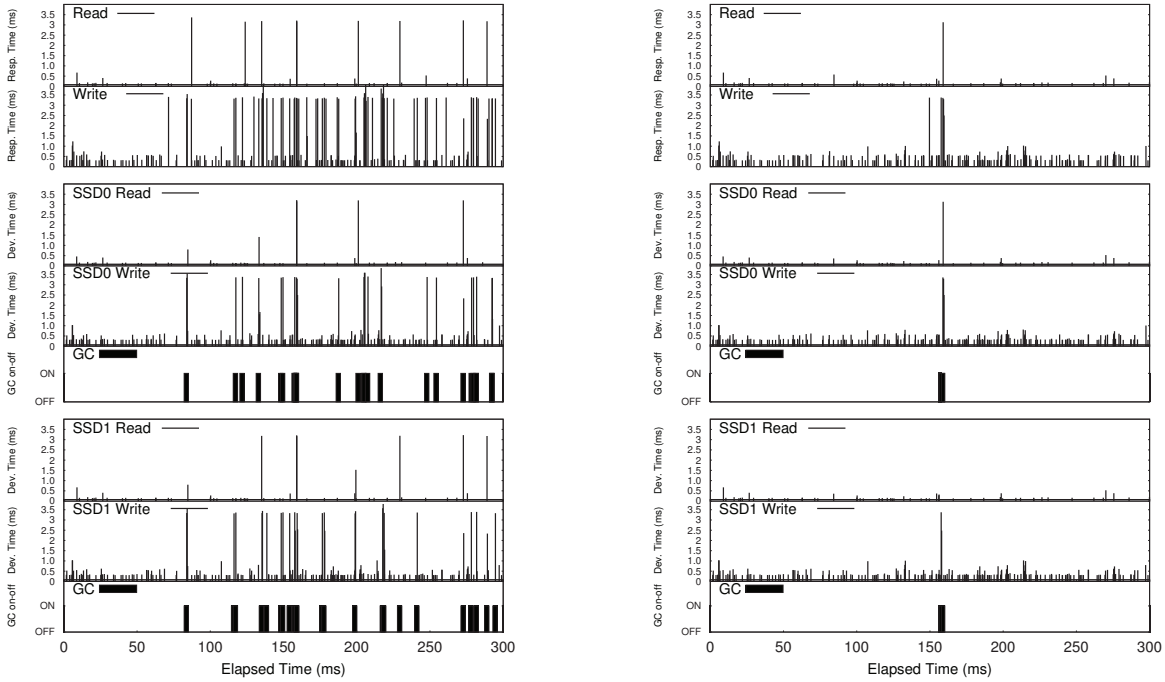
In this section, we compare various GGC algorithms ($Reactive_{selective}$, $Proactive_{idle}$, and $Reactive_{inclusive}$) for their performance and block erase efficiency against the baseline with no GC coordination.

Besides the workloads in Tables 8, three more synthetic workloads are used to cover wider range of workload characteristics. The details are described in Table 9. HPC(W,Skew) is a workload in which 50% of I/O requests go to a particular SSD and others are evenly requested over the SSDs in the array. HPC(W, Burst, M) and HPC(W, Burst, H) are bursty workloads with I/O inter-arrival rates higher than those of HPC(W) by 10 and 100 times, respectively.

Workloads	Request (KB)	Read (%)	Arrival (IOP/s)	Access Pattern
HPC(W)	510.53	20.12	476.50	Evenly
HPC(W,Skew)	510.53	20.12	476.50	Skewed
HPC(W,Burst,M)	510.53	20.12	4,765.00	Evenly
HPC(W,Burst,H)	510.53	20.12	47,650.00	Evenly

TABLE 9
Exploring a wider range of workload characteristics.

Figure 11(a) shows the normalized response time for various GGC schemes with respect to the baseline. The baseline is a RAID of SSDs without GC coordination. To find the lower bound of the response time, we measure the response time of an ideal configuration in which the overhead of GC is eliminated (denoted as $Ideal$ in the following plots).



(a) Baseline (without GGC) (b) GGC

Fig. 10. Microscopic analysis of non-GGC vs. GGC. The first two rows show system response times of overall RAID for read and write requests. Rows 3–5 show device service times for read and write and GC duration for SSD-0. Rows 6–8, similar to rows 3–5, show device service times and GC duration for SSD-1. We present just the time series analysis plots of two SSDs out of eight SSDs used for RAID-0 in our evaluation.

For the real workloads, we see the improvement with the three methods is very similar except for the financial workload; in the financial workload, it can be observed that the response time is reduced by $Reactive_{selective}$ compared with $Reactive_{inclusive}$ and reduced even more by $Proactive_{idle}$, which implies that the financial workload is both skewed and bursty. Also we can observe $Proactive_{idle}$ improves the lifetime compared with $Reactive_{inclusive}$ by reducing the number of erase operations. As a result, $Proactive_{idle}$ improves the performance by 9% and reduces the number of erase by 79%, respectively for the financial workload. Overall, $Proactive_{idle}$ improves the response time by 5.74% to 14.93% (9.59% on average) compared with the baseline. Compared with the ideal, the overhead is as small as 0.0006% to 0.1566% (0.0597% on average). This means that the proposed coordinated GC eliminates most of the GC impact on the response times.

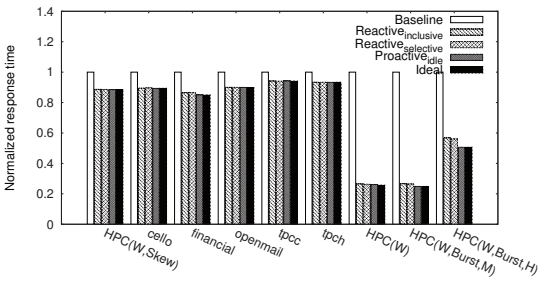
For the workload HPC(W,Skew), we observe that the number of erase operations of $Reactive_{inclusive}$ is about three to four times that of $Reactive_{selective}$. That is because in $Reactive_{inclusive}$, whenever the coordinator needs GC, all SSDs must run their GCs at the same time regardless of their status. $Reactive_{inclusive}$ may incur unnecessary GC for other SSDs. In contrast, in $Reactive_{selective}$, only participants that have met an internal threshold are forced to run GC simultaneously. Thus $Reactive_{selective}$ can reduce the number of erase operations compared with $Reactive_{inclusive}$. However, again note that their response times are not found to be significantly different in Figure 11(a).

In the bursty workloads, we can see an improvement in response time by exploiting idle times for GGC with $Proactive_{idle}$. The more bursty the workload is, the greater the improvement. $Proactive_{idle}$ improves the response time compared with $Reactive_{selective}$ by 0.77%, 5.83%, and 9.47% for HPC(W), HPC(W,Bursty,M), and HPC(W,Bursty,H), respectively. However, the proactive scheme could incur a few additional erase operations. When the workload is bursty, there can be more chances to prepare free blocks in advance during the idle time. Therefore, the number of erase operations can be increased compared with the others.

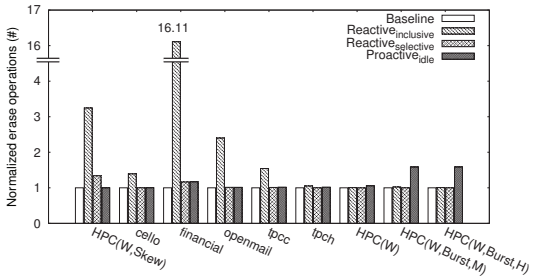
Figure 12 shows the cumulative distribution of response times for the financial and bursty workloads (HPC(W,Burst,H)). We see that $Proactive_{idle}$ is able to reach almost the upper limit of performance improvement that can be achieved. We can see that the lines for $Proactive_{idle}$ and $Ideal$ are almost overlapped.

Figure 13 shows the detailed analysis for the HPC(W,Burst,H) workload. It compares $Reactive_{selective}$ and $Proactive_{idle}$. The first row shows the system response time of the overall RAID. The second row indicates whether GC is running. In $Reactive_{selective}$, GC can be triggered only by incoming requests (never triggered in the idle times of workloads). GC thus affects the foreground operations. The high peak synchronized with GC reflects this situation. In contrast, GC is executed during idle time in $Proactive_{idle}$. As shown in Figures 11(a,b), GC running during idle time does not affect the foreground operations.

Table 10 summarizes our observations from the evalu-

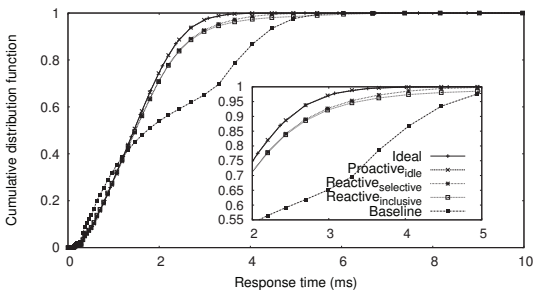


(a) Response times

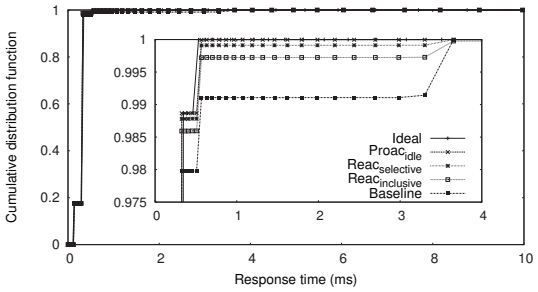


(b) Number of erase operations

Fig. 11. Comparing various GGC algorithms with synthetic and real workloads. All values are normalized with respect to a baseline (no GC coordination).



(a) HPC(W,Bursty,H)



(b) Financial

Fig. 12. Comparing the response times of various GGC algorithms using cumulative distribution function for bursty synthetic workload and financial workload.

ation of various GGC algorithms. For evenly distributed workloads, $Reactive_{inclusive}$ scheme exhibits comparable performance and lifetime to others. Since it is easy to implement, which means shorter development cycles and costs, $Reactive_{inclusive}$ might be the best option for evenly distributed workloads. However, if the workloads are not likely to be evenly distributed, then $Reactive_{selective}$ might be a better choice since it improves the SSD lifetime. If the system requires high performance and the lifetime is not of the utmost importance, $Proactive_{idle}$ could be selected.

7 Conclusions

In evaluating the existing NAND Flash memory-based SSD technology to employ SSDs for our large-scale HPC storage systems in RAID configurations, we empirically observed significant performance degradations and variations in terms of aggregate I/O throughput and I/O response times. Based on our findings, we believe that when current NAND Flash memory-based SSDs and RAID controllers are used in RAID-set configurations, lack of coordination of the local GC processes amplifies these performance degradations and variations. Furthermore, our work reveals that these performance degradations and variations are more pronounced and directly correlated with the number of SSDs configured in a RAID. We observed that these performance degradations and variations can be worse in RAID configurations than in individual SSDs, as GCs are scheduled independently by each SSD in an array. From our point of view, although employing SSDs in large-scale HPC storage systems has potential benefits, performance degradations and variation as a result of GC negates some of these benefits.

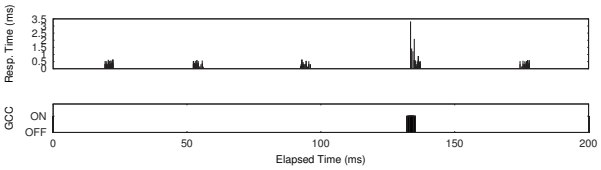
Our paper presents a solution to the performance degradation and variability problem due to GC in NAND Flash memory-based SSDs configured in a RAID. Our proposed architecture, called *Harmonia*, aims to orchestrate a globally coordinated GC process among SSDs in a RAID for improved performance characteristics in terms of aggregate I/O throughput and I/O response times. Our solution includes designs of SSD-aware RAID controllers and RAID-aware SSDs. Connected to an SSD-aware RAID controller, RAID-aware SSDs can participate in the GGC process. We also propose synchronized GGC algorithms in which the RAID-aware SSD controllers can communicate with the SSD-aware RAID controller to coordinate GC tasks. To implement the GGC mechanism, we designed and implemented *reactive* and *proactive* GGC coordination algorithms. For reactive GGC schemes, we propose $Reactive_{inclusive}$ and $Reactive_{selective}$ mechanisms. $Reactive_{inclusive}$ forces all SSDs in the RAID array to participate in the GGC task regardless of the internal status of individual SSDs. $Reactive_{selective}$ provides a refinement to this approach allowing SSDs in the array to participate in GC coordination based individual need. A proactive scheme ($Proactive_{idle}$) invokes GC coordination during idle times, exploiting idle times in common in many I/O workloads.

Our experiments with realistic workloads reveal that the reactive GGC algorithm ($Reactive_{inclusive}$) can improve overall response time by up to 15% (for financial workload) and significantly reduce the variability of performance, compared with a non-synchronized GC mechanism. $Reactive_{selective}$ could reduce the number of block erases compared with the $Reactive_{inclusive}$. Re-

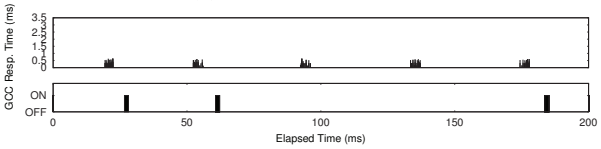
Workload Characteristics	Response time			Number of erase operations		
	$Reactive_{incl.}$	$Reactive_{sel.}$	$Proactive_{idle}$	$Reactive_{incl.}$	$Reactive_{sel.}$	$Proactive_{idle}$
Evenly distributed	Fair	Fair	Fair	Fair	Fair	Fair
Skewed	Fair	Fair	Fair	Poor	Good	Good
Bursty	Poor	Poor	Good	Good	Good	Poor

TABLE 10

Summary of comparing GGC coordination methods. "Poor" means longer response time or more erase operations. $Reactive_{incl.}$ and $Reactive_{sel.}$ denote $Reactive_{inclusive}$ and $Reactive_{selective}$ respectively.



(a) $Reactive_{selective}$



(b) $Proactive_{idle}$

Fig. 13. Microscopic analysis of Reactive selective and Proactive idle GGC algorithms.

sponse time and performance variability were improved for all workloads in our study. In particular, for bursty workloads dominated by large writes (HPC(W) workload), we observed a 69% improvement in response time and a 71% reduction in performance variability compared with uncoordinated GC. We also showed that our proactive GGC algorithm ($Proactive_{idle}$) can further improve the I/O performance by up to 9% while increasing the lifetimes of SSDs by reducing the number of block erase counts by up to 79% compared with a reactive algorithm (in particular for the Financial workload).

ACKNOWLEDGMENTS

We would like to specially thank Doug Reitz for his detailed comments and proof-reading which helped us improve the quality of this paper. This research used resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.

REFERENCES

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for ssd performance. In *USENIX Annual Technical Conference on Annual Technical Conference*, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.

[2] S. S. Center. Supercomputer uses flash to solve data-intensive problems 10 times faster, 2010. http://www.sdsc.edu/News%20Items/PR110409_gordon.html.

[3] F. Chen, D. A. Koufaty, and X. Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, SIGMETRICS'09, pages 181–192, 2009.

[4] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the power-hungry disk. In *In Proceedings of the 1994 Winter USENIX Conference*, pages 293–306, 1994.

[5] K. B. F. Ferreira, P. Bridges, and R. Brightwel. Characterizing application sensitivity to OS interference using kernel-level noise injection. In *Proceedings of Supercomputing*, SC'08, pages 1–12, 2008.

[6] E. Gal and S. Toledo. Algorithms and data structures for flash memories. *ACM Computing Survey*, 37(2):138–163, 2005.

[7] G. Greider. HPC I/O and file systems issues and perspectives, 2006. http://www.dtc.umn.edu/disc/isw/presentations/isw46.pdf.

[8] J. Guerra, H. Pucha, J. Glider, W. Belluomini, and R. Rangaswami. Cost effective storage using extent based dynamic tiering. In *Proceedings of the 9th USENIX conference on File and stroage technologies*, FAST'11, pages 20–20, Feburary 2011.

[9] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, ASPLOS'09, pages 229–240, 2009.

[10] S. Gurumurthi, A. Sivasubramaniam, and V. K. Natarajan. Disk drive roadmap from the thermal perspective: A case for dynamic thermal management. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA '05, pages 38–49, 2005.

[11] J. H. He, A. Jagatheesan, S. Gupta, J. Bennett, and A. Snavely. DASH: a recipe for a flash-based data intensive supercomputer. In *Proceedings of Supercomputing)*, SC'10, November 2010.

[12] HP-Labs. The Openmail Trace. http://tesla.hpl.hp.com/opensource/openmail/.

[13] Intel. IntelXeon Processor X5570 8M Cache, 2.93 GHz, 6.40 GT/s Intel QPI. http://ark.intel.com/Product.aspx?id=37111.

[14] Intel. Intel X25-E Extreme 64GB SATA Solid-State Drive SLC. http://www.intel.com/design/flash/nand/extreme/index.htm.

[15] H. Kim and S. Ahn. BPLRU: A buffer management scheme for improving random writes in flash storage. In *Proceedings of the USENIX Conference on File and Storage Technologies*, FAST'08, pages 1–14, Feburary 2008.

[16] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam. Hybridstore: A cost-efficient, high-performance storage system combining SSDs and HDDs. In *Proceedings of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS'11, July 2011.

[17] Y. Kim, S. Gurumurthi, and A. Sivasubramaniam. Understanding the performance-temperature interactions in disk i/o of server workloads. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, HPCA'06, pages 179–189, Febuary 2006.

[18] Y. Kim, S. Oral, D. A. Dillow, F. Wang, D. Fuller, S. Poole, and G. M. Shipman. An empirical study of redundant array of independent solid-state drives (RAIS). Technical Report ORNL/TM-2010/61, Oak Ridge National Laboratory, March 2010.

[19] Y. Kim, S. Oral, G. M. Shipman, J. Lee, D. Dillow, and F. Wang. Harmonia: A globally coordinated garbage collector for arrays of solid-state drives. In *Proceedings of the IEEE Symposium on Massive Storage Systems and Technologies*, MSST'11, May 2011.

[20] J. Lee, Y. Kim, G. M. Shipman, S. Oral, J. Kim, and F. Wang. A semi-preemptive garbage collector for solid state drives. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, ISPASS'11, April 2011.

[21] S. Lee, D. Shin, Y.-J. Kim, and J. Kim. LAST: Locality-aware sector translation for NAND flash memory-based storage ssystems. *SIGOPS Oper. Syst. Rev.*, 42(6):36–42, 2008.

[22] S.-W. Lee and B. Moon. Design of flash-based DBMS: an in-page logging approach. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD'07, pages 55–66, 2007.

[23] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song. A log buffer-based flash translation layer using fully-

associative sector translation. *ACM Trans. Embed. Comput. Syst.*, 6(3):18, 2007.

[24] M. Li, S. S. Vazhkudai, A. R. Butt, F. Meng, X. Ma, Y. Kim, C. Engelmann, and G. M. Shipman. Functional partitioning to optimize end-to-end performance on many-core architectures. In *Proceedings of Supercomputing*, SC'10, November 2010.

[25] LSI. MegaRAID SAS 9260-8i RAID Card. http://www.lsi.com/channel/products/megaraid/sassata/9260-8i/index.html.

[26] M. Mallary, A. Torabi, and M. Benakli. One terabit per square inch perpendicular recording conceptual design. *IEEE Transactions on Magnetics*, 38(4):1719–1724, July 2002.

[27] N. Mi, A. Riska, E. Smirni, and E. Riedel. Enhancing data availability in disk drives through background activities. In *Proceedings of the Symposium on Dependable Systems and Networks*, DSN'08, pages 492–501, June 2008.

[28] N. Mi, A. Riska, Q. Zhang, E. Smirni, and E. Riedel. Efficient management of idleness in storage systems. *Trans. Storage*, 5:4:1–4:25, June 2009.

[29] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating server storage to SSDs: analysis of tradeoffs. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys'09, pages 145–158, 2009.

[30] H. Newman. What is HPCS and how does it impact I/O, 2009. http://wiki.lustre.org/images/5/5a/NewmanMayLustreWorkshop.pdf.

[31] H. Niijima. Design of a solid-state file using flash eeprom. *IBM Journal of Research and Developement*, 39(5):531–545, 1995.

[32] S. Oral, F. Wang, D. A. Dillow, R. Miller, G. M. Shipman, and D. Maxwell. Reducing application runtime variability on Jaguar XT5. In *Proceedings of Cray User's Group Meeting*, CUG'10, May 2010.

[33] S.-y. Park, D. Jung, J.-u. Kang, J.-s. Kim, and J. Lee. CFLRU: a replacement algorithm for flash memory. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, CASES'06, pages 234–241, 2006.

[34] D. Patterson, G. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of ACM SIGMOD Conference on the Management of Data*, pages 109–116, June 1988.

[35] F. Petrini, D. J. Kerbyson, and S. Pakin. The case of the missing supercomputer performance: achieving optimal performance on the 8,192 processors of ASCI Q . In *Proceedings of Supercomputing*, SC'03, pages 1–12, 2003.

[36] Seagate. Seagate Cheetah 15K.7 Disc Drive. http://www.seagate.com/docs/pdf/datasheet/disc/ds_cheetah_15k_7.pdf.

[37] E. Seppanen, M. T. O'Keefe, and D. J. Lilja. High performance solid state storage under Linux. In *Proceedings of IEEE 26th Symposium on Mass Storage Systems and Technologies*, MSST'10, pages 1–12, 2010.

[38] P. Shenoy and H. M. Vin. Cello: A Disk Scheduling Framework for Next Generation Operating Systems. *Real-Time Syst.*, 22(1/2):9–48, 2002.

[39] Storage-Performance-Council. OLTP Trace from UMass Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage.

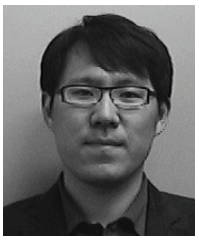
[40] Super Talent. Super Talent 128GB UltraDrive ME SATA-II 25 MLC. http://www.supertalent.com/products/ssd_detail.php?type=UltraDrive%20ME.

[41] D. A. Thompson and J. S. Best. The future of magnetic data storage technology. *IBM J. Res. Dev.*, 44:311–322, May 2000.

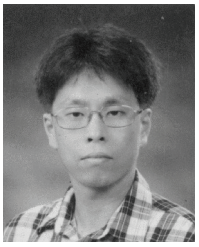
[42] Transaction-Procesing-Perforamnce-Council. TPC-C, an OLTP Benchmark. http://www.tpc.org/tpcc/.

[43] A. Uselton. Deploying server-side file system monitoring at NERSC. In *CUG '09: Proceedings of Cray User's Group (CUG) Meeting*, CUG'09, 2009.

[44] J. Zhang, A. Sivasubramaniam, H. Franke, N. Gautam, Y. Zhang, and S. Nagar. Synthesizing representative I/O workloads for TPC-H. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, HPCA'04, page 142, 2004.



Youngjae Kim received the B.S. degree in computer science from Sogang University, Korea in 2001, the M.S. degree from KAIST in 2003 and the Ph.D. degree in computer science and engineering from Pennsylvania State University in 2009. He joined Oak Ridge National Laboratory as an I/O Systems Computational Scientist in 2009. His research interests include operating systems, parallel I/O and file systems, storage systems, emerging storage technologies, and performance evaluation for high-performance computer systems. Also, he is currently an adjunct professor in the school of electrical and computer engineering at Georgia Tech.



Junghee Lee received the B.S. and M.S. degrees in computer engineering from Seoul National University in 2000 and 2003, respectively. From 2003 to 2008, he was with Samsung Electronics, where he worked on electronic system level design of mobile system-on-chip. He is currently a Ph.D. student of Georgia Institute of Technology. His research interests include architecture design of microprocessors, memory hierarchy, and storage systems for high performance computing and embedded systems.

Sarp Oral is a Research Scientist at the National Center for Computational Sciences of Oak Ridge National Laboratory where he is a staff member of the Technology Integration Group. Dr. Oral holds a Ph.D. in computer engineering from University of Florida in 2003 and an M.Sc. in biomedical engineering from Cukurova University, Turkey in 1996. His research interests are performance evaluation, modeling, and benchmarking, parallel I/O and file systems, high-performance computing and networking, computer architecture, fault-tolerance, and storage technologies.

David A. Dillow is an Advanced Systems Architect and Developer at the National Center for Computational Sciences located at Oak Ridge National Laboratory. He joined NCCS in 2007 to deploy the first large-scale, routed Lustre file system, and he currently co-chairs Open Scalable File Systems Technology Working Group. His research combines his interest in operating systems, parallel computing, and high-performance networks and storage to design systems for future supercomputers.

Feiyi Wang received his Ph.D. in computer engineering from North Carolina State University (NCSU). Before he joined Oak Ridge National Laboratory as a Research Scientist, he worked at Cisco Systems and Microelectronic Center of North Carolina (MCNC) as a lead developer and principal investigator for several DARPA-funded projects. His current research interests include high performance storage system, parallel I/O and file systems, fault tolerance and system simulation, and scientific data management and integration. He is currently a senior member of IEEE.

Galen M. Shipman is the Data Systems Architect for the Computing and Computational Sciences Directorate at Oak Ridge National Laboratory. He is responsible for defining and maintaining an overarching strategy for data storage, data management, and data analysis spanning from research and development to integration, deployment and operations for high-performance and data-intensive computing initiatives at ORNL. Prior to joining ORNL, he was a technical staff member in the Advanced Computing Laboratory at Los Alamos National Laboratory. Mr. Shipman received his B.B.A. in finance in 1998 and a M.S. degree in computer science in 2005 from the University of New Mexico. His research interests include High Performance and Data Intensive Computing.