

# Harmonia: A Globally Coordinated Garbage Collector for Arrays of Solid-state Drives

Youngjae Kim, Sarp Oral, Galen M. Shipman, Junghee Lee<sup>†</sup>, David A. Dillow, and Feiyi Wang  
National Center for Computational Sciences  
Oak Ridge National Laboratory, Oak Ridge, TN 37831-6016  
{kimy1, oralhs, gshipman, 7o2, dillowa, fwang2}@ornl.gov

**Abstract**—Solid-State Drives (SSDs) offer significant performance improvements over hard disk drives (HDD) on a number of workloads. The frequency of garbage collection (GC) activity is directly correlated with the pattern, frequency, and volume of write requests, and scheduling of GC is controlled by logic internal to the SSD. SSDs can exhibit significant performance degradations when garbage collection (GC) conflicts with an ongoing I/O request stream. When using SSDs in a RAID array, the lack of coordination of the local GC processes amplifies these performance degradations. No RAID controller or SSD available today has the technology to overcome this limitation. This paper presents *Harmonia*, a Global Garbage Collection (GGC) mechanism to improve response times and reduce performance variability for a RAID array of SSDs. Our proposal includes a high-level design of SSD-aware RAID controller and GGC-capable SSD devices, as well as algorithms to coordinate the global GC cycles. Our simulations show that this design improves response time and reduces performance variability for a wide variety of enterprise workloads. For bursty, write dominant workloads response time was improved by 69% while performance variability was reduced by 71%.

## I. INTRODUCTION

Hard disk drives (HDD) are the leading media in storage systems. HDDs are widely deployed from embedded to enterprise-scale systems for the last several decades. HDD manufacturers were successful in providing a continuous improvement in total disk capacity by increasing the storage density while bringing down the price-per-byte using mass production. Perpendicular recording [29] has continued this trend but further advances will require new technologies such as patterned media which present significant manufacturing challenges. On the other hand, HDD I/O performance increased at a slower pace compared to the storage density. Increasing the platter rotational speed (rotations per minute – RPM) was key to this progress. A recent single enterprise-class magnetic disk today can provide up to 204 MB/s at 15,000 RPMs [43]. However, we are now at a point where HDD designers conclude it is extremely hard to increase platter RPM beyond its current state because of power consumption and thermal dissipation issues [12].

Solid state disks (SSD), especially NAND Flash memory-based SSDs, are a leading media in storage systems. Re-

cently several attempts have been made to employ SSDs for enterprise-scale and HPC storage systems [3], [13], [26], [32]. Unlike magnetic rotational disks, NAND Flash memory-based SSDs have no mechanical moving parts, such as spindles and voice-coil motors. Therefore, NAND Flash memory technology offers a number of benefits over conventional hard disk drives (HDDs), such as lower power consumption, lighter weight, higher resilience to external shocks, ability to sustain hotter operating regimes, and lower I/O access times [8]. Additionally, since SSD Flash chips are packaged in HDD form factors and SSDs are compatible with HDD device drivers and I/O buses, one-to-one replacement of HDDs with SSDs is possible. Such that, operating systems see the SSDs as normal block devices just as HDDs. This enables to simply replace HDDs and provide high bandwidth and lower latency, however, SSD's performance is highly limited by I/O access patterns.

Unlike magnetic disk, NAND flash memory requires an erase operation in addition to normal read and write operations as in HDDs [34]. Each read and write operation is performed at the granularity of a page (2-4KB) whereas an erase operation is performed at the granularity of a block (128-256KB). In addition to this mismatch of operational granularities, in flash, no data can be written into a page that is not in the erased state. Thus, SSD allows out-of-place update operations, which eventually requires a cleaning process to collect stale data for providing free space, known as garbage collection (GC) process. While GC process is happening, incoming requests, if their target is the same Flash chip that is busy with GC, are delayed until the completion of the GC [22]. Furthermore, fragmentation, which caused by small random writes/updates, can significantly increase the GC overhead [4], [23], [20], resulting in higher frequency of copy-operations for non-stale data pages and block erase operations [20], [11].

Redundant Arrays of Inexpensive (or Independent) Disks (RAID) [38] were introduced to increase the performance and reliability of HDD-based storage systems. RAID provides parallelism of I/O operations by combining multiple inexpensive disks thereby achieving higher performance and robustness than a single HDD. RAID has since become the de facto standard for building high-performance and robust HDD-based storage systems. Similarly, we analyzed SSD-based RAID sets

<sup>†</sup>He is currently a doctoral graduate student in the school of Electrical and Computer Engineering at the Georgia Institute of Technology.

and we found that a RAID of cheap commercially-off-the-shelf (COTS) SSDs is more cost-effective than a high-performance and expensive PCIe based SSD [21]. However, we also observed that SSD-based RAID configurations can exhibit serious bandwidth variability. due to temporary bandwidth degradations of individual SSDs and as well known, aggregate RAID performance is dictated by the slowest drive of the array. Further investigation provided empirical evidence that uncoordinated garbage collection processes on individual SSDs are the culprit for the overall performance degradations. Thus, in this paper, we propose a RAID-level garbage collector, called *Harmonia* that can globally coordinate local GC operations in the array of SSDs.

This paper makes the following specific contributions:

- While synchronization of distributed processes to reduce performance variability has been previously discussed [35], [39], [7], there has been a lack of research on coordinating GC processes in a RAID array of SSDs. To our knowledge, this is the first work to address the performance variability in a RAID set of SSDs due to uncoordinated GC processes.
- We empirically observed that the performance of an SSD can be highly impeded by GC processes. We term this performance degradation *pathological behavior* of an SSD. Then, we identify that such pathological behavior can be worse in RAID configurations compared to individual SSDs as GCs are scheduled independently by each SSDs in an array.
- We propose a globally coordinated GC (GGC) mechanism to reduce the effect of an aggregate performance degradation. To provide such a GC synchronization mechanism to a RAID of SSDs, we propose designs of SSD-aware RAID controllers and RAID-aware SSDs. Connected to an SSD-aware RAID controller, RAID-aware SSDs will participate in the globally coordinated GC process. Moreover, we propose synchronized GC algorithms in which the RAID-aware SSD controllers can communicate with the SSD-aware RAID controller to coordinate GC tasks.
- For evaluation of our proposed GC algorithm, we enhanced an SSD simulator developed by Microsoft Research [1] with a globally synchronized GC mechanism for RAID of SSDs. Our experiments with realistic workloads showed that a globally synchronized GC mechanism can improve the overall response time by up to 69% and also significantly reduce the variability of performance, compared to a baseline of non-synchronized GC mechanism.

The remainder of this paper is organized as follows. We first present an overview of the material and technology in Section II followed by motivation in Section III. Section IV introduces our proposed improved RAID and SSD controller designs, as well as the globally synchronized garbage collection algorithm. In Section V, we present simulation results of our proposed designs and algorithm. Then, we conclude in

Section VI, followed by future works.

## II. BACKGROUND

### A. Flash Memory Technology

Unlike rotating media and volatile memories, where only read and write operations are needed, Flash memory-based storage devices require an *erase* operation [34]. Erase operations are performed at the granularity of a block which is composed of multiple pages, while reads and writes are performed at page level. Each page on Flash can be in one of three states: (i) *valid*, (ii) *invalid* and (iii) *free/erased*. When no data has been written to a page, it is in the erased state. A write operation can only be performed on an erased page, changing its state to valid. Erase operations are significantly slower than reads/writes. On average an erase operation costs 1.5 ms (on average a read operation costs 0.025 ms and a write operation costs 0.200 ms). Therefore, out-of-place writes (in contrast to in-place writes in HDDs) are performed to existing free pages along with marking the page storing the previous version invalid.

### B. Flash Translation Layer

The Flash Translation Layer (FTL) is a software layer within the SSD that translates logical addresses to physical addresses on a Flash device. FTL emulates a normal block device by performing out-of-place updates, which in turn helps to hide the erase operations from the host system. The mapping table is often stored in a small, fast SRAM. FTL can be implemented at different granularities in terms of how large an address space a single entry in the mapping table captures. Many FTL schemes [6], [24], [19], [25], [42], [5] and their improved write-buffering [20] versions have been studied. A recent page-based FTL scheme called DFTL [11] utilizes temporal locality in workloads to overcome the shortcomings of the regular page-based scheme by storing only a subset of mappings (those likely to be accessed) on the limited SRAM and storing the remainder on the Flash device itself. Also, there are several works in progress on the optimization of buffer management in NAND Flash based environments [36], [17].

### C. Garbage Collection

While read and write operations to erased pages in SSDs are conceptually similar to their HDD counterparts, overwrite operations, where the destination address is not empty, are much more complicated. For an overwrite operation to succeed, the contents of that particular block first needs to be erased before a write operation can be performed reliably. Therefore, a read-erase-modify-write operation is not efficient due to the high overhead of erase operations. Alternatively, SSDs use a copy-on-write operation in which the contents of the target block are copied into the SSD-on-board memory, modified, then written to a known free block. Then the original target block for the overwrite operation is marked *invalid*. Although this is more efficient compared to the read-erase-modify-write method (since it does not require an explicit erase operation), over time the number of available free blocks

decreases thereby requiring free blocks to be reaped from the invalid block pool. This process is known as Garbage Collection (GC).

Current generation SSDs use a variety of different algorithms and policies for GC that are vendor dependent (specific). It has been empirically observed that GC activity is directly correlated with the frequency of write operations, amount of data written, and/or the free space on the SSD [4]. This GC process can significantly impede both read and write performance, increasing queueing delay [11].

#### D. Mitigating GC overheads

While GC overhead is highly dependent on workload characteristics, random writes often have the most negative impact on overall performance [23], [20]. Thus, it has been attempted to reduce the GC overhead by increasing the sequentiality of requests in workloads. Requests are rearranged and coalesced to increase the sequentiality [20]. Others have attempted to design FTLs that minimize GC overhead [6], [25], [24], [11]. In spite of these attempts, SSDs still suffer from variable performance due to GC. Furthermore, during an ongoing GC process incoming requests targeted for the same Flash device that is busy with the ongoing GC process are stalled and placed in a queue and scheduled for service following the completion of the GC process [22]. This stalling can significantly degrade performance when incoming requests are bursty. Other than the degraded performance, the high variability of in terms of I/O bandwidth and system response times makes the overall RAID configuration less reliable as system response can not be guaranteed. Thus, *performance robustness* of systems becomes as important as providing high performance, such that a robust system can operate with certain variations. In the next section we present our empirical findings on overall RAID-level performance variation for bursty and write dominant workloads.

### III. PERFORMANCE DEGRADATION DUE TO GCs IN SSDs

In this section we perform a series of experiments using various COTS SSDs and RAID controllers in order to empirically observe the effect of GC and to establish a *baseline* performance of an individual SSD.

#### A. Experimental Setup

All experiments were performed on a single server with 24 GB of RAM and an Intel Xeon Quad Core 2.93GHz CPU [15]. The operating system was Linux with Lustre-patched 2.6.18-128 kernel. The *noop* I/O scheduler that implements FIFO queueing was used [41]. The test-bed has seven 8x PCIe slots and two were populated with LSI MegaRAID SAS 9260-8i KIT PCIe RAID Adapters [28], each of which can support up to 8 SATA drives.

We examined two representative SSDs that are detailed in Table I. We selected the Super Talent 128 GB FTM28GX25H SSD [47] as a representative of multi-level cell (MLC) SSDs with SATA interfaces and the Intel 64 GB SSDA2SH064G101 SSD [16] as a representative of single-level

Label	SSD(M)	SSD(S)
Company	Super-Talent	Intel
Type	MLC	SLC
Interface	SATA-II	SATA-II
Capacity (GB)	120	64
Erase (#)	10-100K	100K-1M
Power (W)	1-2	1-2

TABLE I  
SSD CHARACTERISTICS.

RAID 0	1 SSD (No)	4 SSDs (Yes)	6 SSDs (Yes)
Queue depth	64	64	64
Request size	313 KB	1.25 MB	1.87 MB

TABLE II  
BENCHMARK WORKLOADS – QUEUE DEPTHS AND REQUEST SIZES.

RAID Scheme	0
Device (#)	6
Write Cache	Write Through
Read Ahead	No
Direct I/O	Yes
Stripe Size	256KB

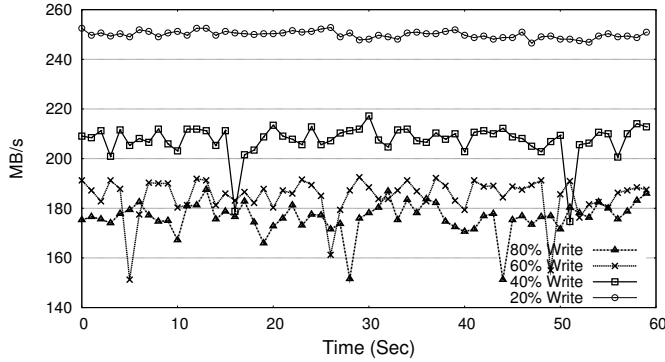
TABLE III  
SETTINGS OF THE LSI MEGARAID CONTROLLER.

cell (SLC) SSDs. We denote the SuperTalent MLC, and Intel SLC devices as SSD(M), and SSD(S) in the remainder of this study, respectively.

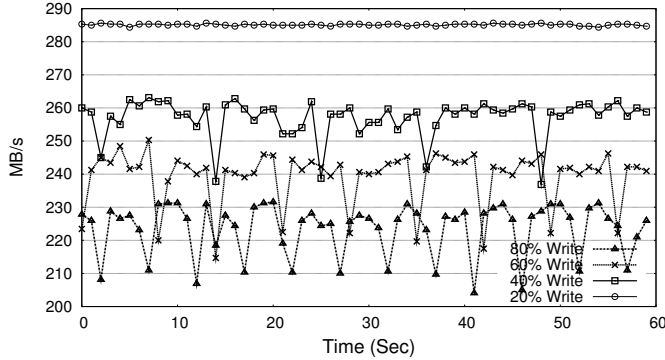
We examined the I/O bandwidth responses of individual COTS SSD for workloads described in Table II. To measure the I/O performance, we used a benchmark tool that uses the *libaio* asynchronous I/O library on Linux [21]. The *libaio* provides an interface that can submit one or more I/O requests in one system call *iosubmit()* without waiting for I/O completion. It also can perform reads and writes on raw block devices. We used the direct I/O interface to bypass the operating system I/O buffer cache by setting the *O-DIRECT* and *O-SYNC* flags in the file *open()* call.

#### B. Benchmark Workloads

In order to conduct a fair comparison for the performance variability, we exercised the identical I/O loads to both SSDs. A high queue depth (number of outstanding requests in the I/O queue) is used to observe the impact of GC in time domain. Also, we varied the percentage of writes in workloads between 20% and 80% in increasing steps of 20%. We measured I/O bandwidth in one second intervals. We describe the details of request size and queue depth settings for individual and RAID SSDs in Table II. In order to use imbalanced I/O loads for the tests of RAID of SSDs, we used 1.25MB of request size for RAID of 4 SSDs while stripe size is 256KB. Note that in this case, one of four SSDs will receive two 256KB striped requests whereas others will have one 256KB striped request. Taking this as a default request size in workloads, we scaled the request size by the number of SSDs in RAID.



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



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

Fig. 1. Pathological behavior of individual 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)
SSD(S)	avg	223.5	239.3	257.1	285.1
	(stddev)	(7.96)	(8.38)	(5.86)	(0.28)

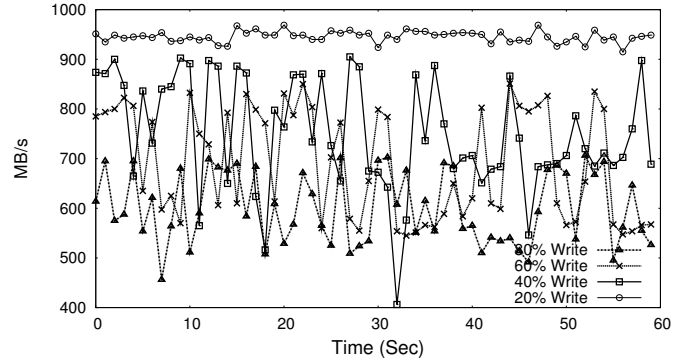
TABLE IV

AVERAGE AND STANDARD DEVIATION FOR FIGURE 1(A)(B). *stddev* DENOTES STANDARD DEVIATION.

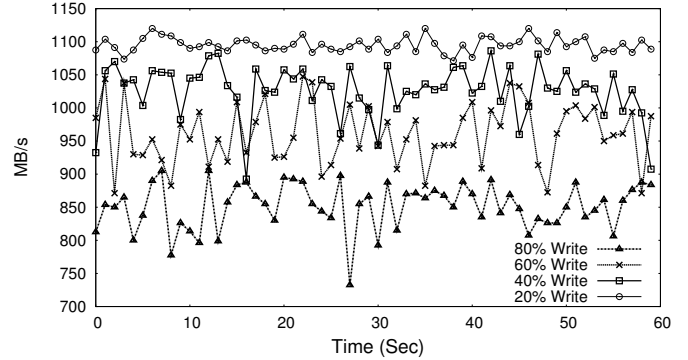
### C. Pathological behavior of individual SSDs

Figure 1 illustrates our results for individual SSDs. We present average values and standard deviations of Figure 1 in Table IV. Figure 1 presents time-series analysis results for workloads that have 20% or more writes. We observe that the bandwidth fluctuates more widely due to GC activity with respect to increasing mix of write requests in workloads.

Figure 1(a) illustrates, for the 80% write dominant I/O workload, the SSD(M) I/O throughput dropping below the peak performance (170 MB/s) at the 10<sup>th</sup> second. I/O throughput drops below 166 MB/s at the 19<sup>th</sup> second and then drops further to 152 MB/s in the next 10 seconds. Overall, SSD(S) shows higher bandwidth than SSD(M) with a similar variance for all workloads we examined, because SSD(S) is an SLC, while SSD(M) is an MLC. For instance, SSD(S)'s I/O throughput reached 210 MB/s at the peak for a workload of 80% writes and dropped to 183 MB/s. As we increased



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



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

Fig. 2. Pathological behavior of RAID.

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)
RAID(S)	avg	851.5	961.2	1026.1	1095.2
	(stddev)	(34.98)	(46.37)	(40.38)	(11.39)

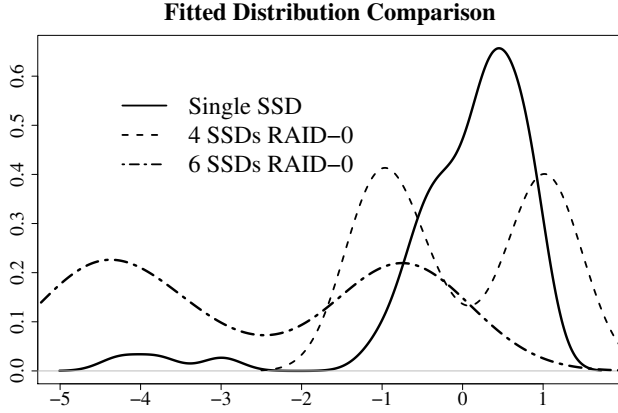
TABLE V

AVERAGE AND STANDARD DEVIATION FOR FIGURE 2(A)(B). *stddev* DENOTES STANDARD DEVIATION.

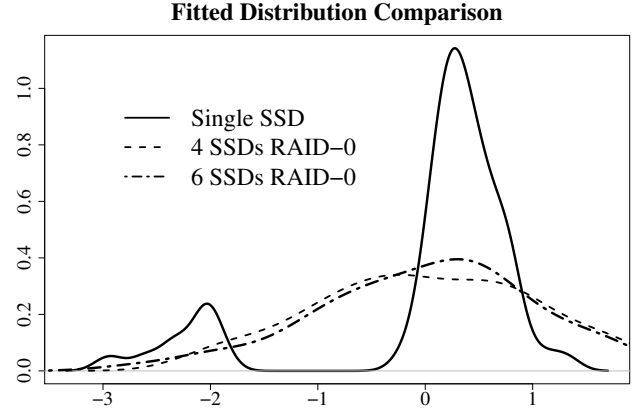
the amount of reads in the workloads from 20% to 80%, we observed that SSD(M)'s and SSD(S)'s I/O throughput increased by 41% and 28%, respectively. Next, we extend our experiments to arrays of COTS SSDs.

### D. Pathological Behavior of Arrays of SSDs

We used two PCIe interfaced hardware RAID controllers for each configuration. We configured RAID of SSDs as given in Table III and experimented with workloads described in Table II. In Figure 2 we present results of our experiments for RAID(M) and RAID(S) arrays. And their average and standard deviations are shown in Table V. RAID(M) and RAID(S) were configured as level 0 arrays for a workload mix of writes and reads by varying write percentage in time-series plot. Similar to performance and variability tests with single SSDs, we observe high performance variability in both RAID(M) and RAID(S) as expected.



(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.

### E. Performance Variability with the Number of SSDs in RAID

We compare the performance variability of the SSD RAID sets for different number of SSDs for a workload of 60% writes. We normalized the I/O bandwidth of each configuration with a Z-transform [18] and then curve-fitted and plotted their density functions. Our tests were performed using workloads described in Table V. As we observed from Table V that the coefficient of variance<sup>1</sup> is the highest when write percentage is 60% in our experiments, we show analysis results for 60% writes of workloads as representative experimental results.

In Figure 3 we see that the lines for RAID of 4 SSDs and 6 SSDs show both wider curves than that for single SSDs. Note that the wider curve is shaped, the higher its performance variability is. Or, in other words tighter the distribution is (e.g. minimal spread at the tails with a single spike at the center) less variability it exhibits in terms of throughput. Thus, we observe the performance variability exhibited by RAID of SSDs far exceeds the projected linear relationship between single SSD and RAID of SSDs. Our conjecture is that uncoordinated GC operations are increasing performance variability. We see that the performance variability can further increase as we increase the number of SSDs in RAID as is clearly seen in Figure 3(a). Furthermore, we also observe that performance variance increases more rapidly for RAID arrays of MLC SSDs compared to their SLC counterparts, with increasing number of SSDs in an array.

Moreover, we also observe that the performance variability of RAID sets comprised of MLC SSDs does not scale as well as that of SLC SSDs. As seen in Figure 3(b), there is not a significance difference between 4 and 6 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. In

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

order to generalize this statement, we will further investigate the uncoordinated GC synchronization problems for various COTS SSDs by increasing the number of devices in the RAID array.

We also found that a per-drive bandwidth drops as we increase the number of SSDs in RAID. Table VI presents a per-drive bandwidth for single SSD and RAID of four and six SSDs. We calculate a per-drive bandwidth for 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 SSDs in RAID. In Table VI, it can be seen that there are bandwidth drops by up to 43 and 48 MB/s respectively for 6 RAID of SSD(M)s and SSD(S), compared to the bandwidths of their single SSDs.

Type	1 SSD	4 SSDs	6 SSDs
SSD(M)	184.8	172.4	141.2
SSD(S)	239.3	240.3	191.1

TABLE VI  
PER-DRIVE BANDWIDTH (MB/S).

### F. Performance Variability and RAID Levels

We investigated the correlation between various RAID levels and performance variability. For these experiments the total number of disks for all RAID levels were same. However, the number of data disks changed as dictated by each specific RAID configuration. Table VII details our results in terms of average bandwidth (MB/s) and standard deviation of the observed throughput. As can be seen in Table VII RAID-5 has a lower standard deviation than RAID-0 and similarly RAID-6 has a lower standard deviation than RAID-5. However, standard deviation is not a meaningful metric by itself for assessing the performance variability. A right metric for comparing bandwidth variability should take into account both mean and standard deviation values. Therefore, we calculated the

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 VII  
PERFORMANCE VARIABILITY IN TERMS OF AVERAGE BANDWIDTH (MB/S)  
AND STANDARD DEVIATION OF OBSERVED BANDWIDTH (IN PARENTHESIS)  
FOR VARIOUS RAID CONFIGURATIONS.

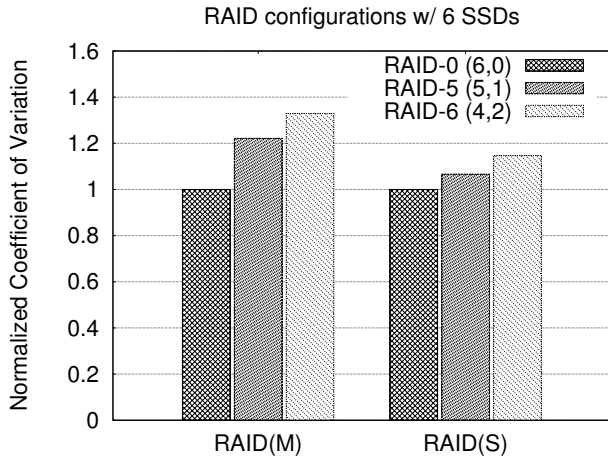


Fig. 4. Performance variability with respect to various RAID levels.

coefficient of variables of the values presented in Table VII and plotted them in Figure 4. Figure 4 illustrates the normalized coefficient of variation in bandwidth for RAID-0, RAID-5, and RAID-6 configurations. The coefficient of variation is useful in comparing the performance variability as it provides a method of understanding the standard deviation relative to the mean. Results in Figure 4 are normalized with respect to the coefficient of variation of RAID-0 results. ‘x’ and ‘y’ in (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. And RAID-6 configuration for both cases presented the highest performance variability in our experiments.

As we have seen from benchmark results with real COTS SSD and RAID components there were technology limitations for current SSD and RAID controller technologies; GC processes per individual SSDs are local and they are never coordinated. COTS 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 will describe components and algorithms for our proposed high-performance and robust SSD-based RAID storage system.

#### IV. HARMONIA: AN EFFICIENT SSD RAID SYSTEM

In a RAID set of SSDs, the aggregate RAID performance is limited by the slowest component of the array. Our empirical results showed that uncoordinated garbage collection is the major culprit behind these temporary slow-downs on individual SSDs. In this section, we present a solution to this problem as to mitigate the performance degradations of SSD RAID sets.

##### A. Coordinated Garbage Collections in SSD RAID sets

Figure 5 depicts conceptual timings of GC processes for a given RAID 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 due to 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 array in Figure 5(a) only achieves its peak performance in time windows B and D. The array is limited to degraded performance is windows A and G due to multiple devices performing GC, as well as in windows C and E due to 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. In this mechanism, GC processes are shifted in time to allow longer windows of peak performance from the RAID array. By advancing the GC process in Figure 5(a) window C to occur simultaneously with the other processes in window A, we 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.

##### B. Design

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 “Global GC optimized SSD” (O-SSD).
- 3) A set of algorithms to perform a globally coordinated GC process on a given SSD-based RAID set comprised of an O-RAID and multiple O-SSD devices.
- 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

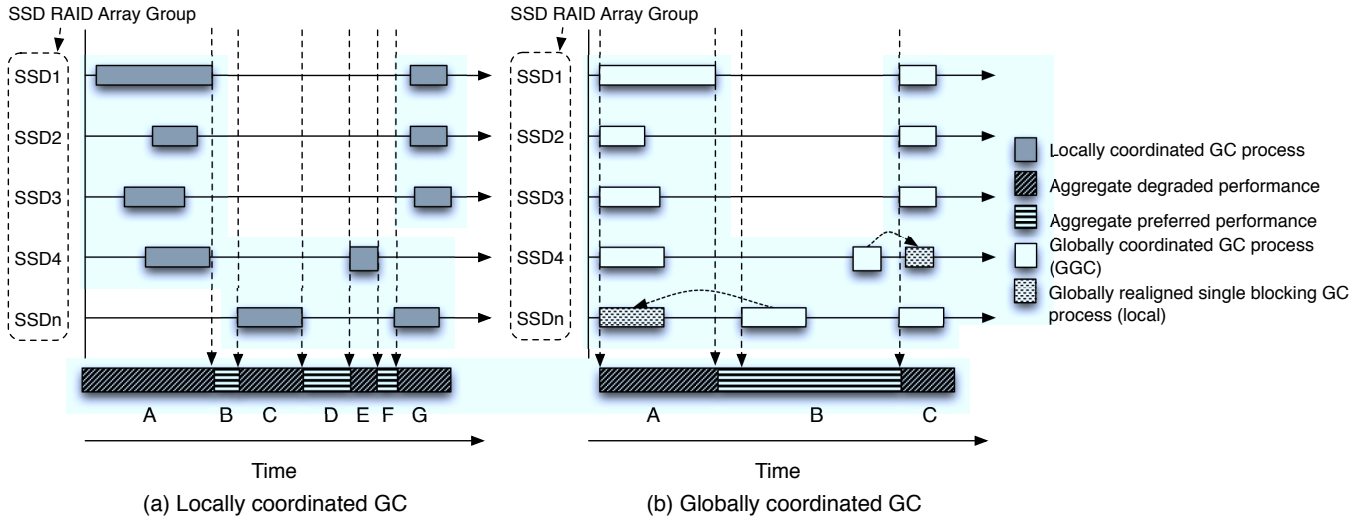


Fig. 5. Effect of globally coordinated GC (GGC). (a) presents locally coordinated GC processes for an un-optimized RAID array, whereas (b) presents globally coordinated GC processes for a GGC optimized RAID array.

global level with global knowledge obtained from all O-SSDs, while an O-SSD only has 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 global garbage collection can only provide maximum effectiveness 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.

### C. GC Coordination Algorithms

1) *Reactive Method*: Coordination is essential to achieving the performance benefits of GGC. There are multiple ways to coordinate a global garbage collection process. However, given the page limitations of this paper we present the reactive soft-limit scheme as a case study. In the *reactive soft-limit* method, an O-SSD communicates to the O-RAID that it has reached an internal threshold and 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 ask each GGC-capable device in the RAID set to initiate a GC cycle. 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. In this method, the O-SSD will delay its GC cycle until it reaches a hard threshold where 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 the lack of a response from the O-RAID will not prevent the O-SSD from performing needed GC.

2) *Proactive Method*: However, one can implement *proactive soft-limit*, and *proactive idle*. All of reactive and proactive schemes are non-exhaustive; multiple methods may be used concurrently. In the *proactive soft-limit* method, 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 methods 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 will attempt 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:

- Internal fragmentation level (ratio of free to used erase blocks)
- Number of free erase blocks available
- ECC correctable error rate on reads

In the *proactive idle* method, the O-RAID identifies points in the IO stream that are expected to have extended idle periods, and initiates a GGC cycle during those lulls in activity.

For the remainder of this paper we will focus on the reactive scheme to explain our proposed design. In our reactive algorithm, the initiator SSD in the array is set as the GGC global coordinator. Unless individual SSDs receive an event of *FORCE GC*, they operate as normal (without a global GC coordination). Otherwise they are forced to start GGC process.

## V. EXPERIMENTAL RESULTS

We extended the SSD simulator developed by Microsoft Research (MSR) [1] to evaluate our proposed GGC-optimized RAID array. The MSR SSD simulator is an enhancement of DiskSim [2] from the CMU Parallel Data Lab. The MSR SSD simulator has been recently released to the public and widely used in several studies to evaluate SSD based storage

systems [37], [27], [40]. The DiskSim simulator has been also widely used for research and development and has been extensively validated [9], [2].

### A. GGC Simulator Design

To explore our proposed design, we extended the baseline DiskSim and MSR SSD environment to implement the *re-active soft-limit* coordination method. In this algorithm, the initiator SSD in the array is set as the GGC global coordinator. Unless individual SSDs receive an event of *FORCE GC*, they operate as normal (without a global GC coordination). Otherwise they are forced to start GGC process. The O-RAID receives a GGC request message from the initiator SSD ( $S_{coordinator}$ ). Upon receiving this message, O-RAID prepares to schedule a GGC event. It iterates over all connected devices and for each device and every stale/invalid element issues a *Forced GC* event. Regardless of the source of the GC event, a local garbage collection process will be triggered to clean the stale/invalid elements until the number of free blocks exceeds an internal threshold.

### B. Baseline Simulation Environment

For the baseline COTS RAID environment, we configured SSD RAID simulator to analyze a RAID-0 array described in Table VIII. The SSD simulator was configured such that each SSD presented the specifications shown in Table IX. In the baseline configuration, there is no coordination of the internal garbage collection for each SSD.

RAID model	
# of SSDs	8
Stripe unit	4KB

TABLE VIII  
RAID-0 CONFIGURATION.

*Simulation “Warm-up”*: Prior to collecting performance data from the simulator, we fill the entire space on each SSD with valid data. This ensures that GC is required on the SSD devices as new write requests arrive during the experimental run.

SSD model	
Total capacity	32GB
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
Page read latency	0.025 ms
Page write latency	0.200 ms
Block erase latency	1.5 ms

TABLE IX  
SSD MODEL PARAMETERS.

Workloads	Avg. Req. Size (KB)	Read (%)	Arrival Rate (IOP/s)
HPC(W)	510.53	20.12	476.50
HPC(R)	510.53	80.08	476.50

TABLE X  
HPC-LIKE SYNTHETIC WORKLOADS.

Workloads	Avg. Req. Size (KB)	Read (%)	Arrival Rate (IOP/s)
TPC-C [48]	7.06	20.50	388.32
Openmail [14]	9.49	63.30	846.62
TPC-H [50]	31.62	91.80	172.73
Financial [46]	7.09	18.92	47.19
Cello [45]	7.06	19.63	74.24

TABLE XI  
ENTERPRISE-SCALE REALISTIC WORKLOADS.

### C. Workload Descriptions and Metrics

We use a wide spectrum of workloads from industry and research sources to evaluate the performance of our GGC method. We use a mixture of HPC-like workloads and realistic enterprise-scale workloads to study the impact of our proposed *Harmonia*, globally coordinated garbage collection in RAID of SSDs. This broad spectrum was chosen to obtain a more realistic view of the benefits of coordinated garbage collection. As described in Table X and XI, these workloads include both read and write dominated traces.

For HPC-like workloads, we chose read and write and bursty workloads whose characteristics are described in Table X. 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 [44], [10], [33]. HPC(R) is a read-dominated (80%) workload that represents heavy read patterns of HPC environments [49].

For enterprise-scale realistic workloads, five commercial I/O traces are used, details of which are shown in Table XI. We used write dominant I/O traces from an OLTP application running at a financial institution made available by the Storage Performance Council (SPC), referred to as the Financial trace, and from Cello99, which is a disk access trace collected from a time-sharing server exhibiting significant writes which was running the HP-UX operating system at Hewlett-Packard Laboratories. Also, we examined two read-dominant workloads. TPC-H is a disk I/O trace collected from an OLAP application examining large volumes of data to execute complex database queries. Also, we consider e-mail server workloads referred as Openmail.

While the device service time captures the overhead of garbage collection 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 utilize (i) the response time measured at the block device queue and (ii) the variance



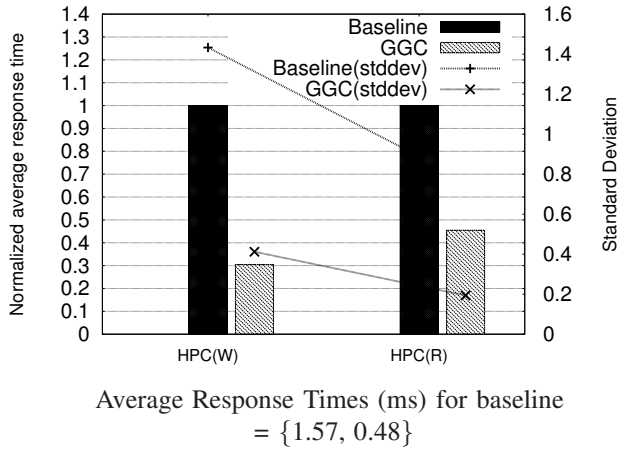
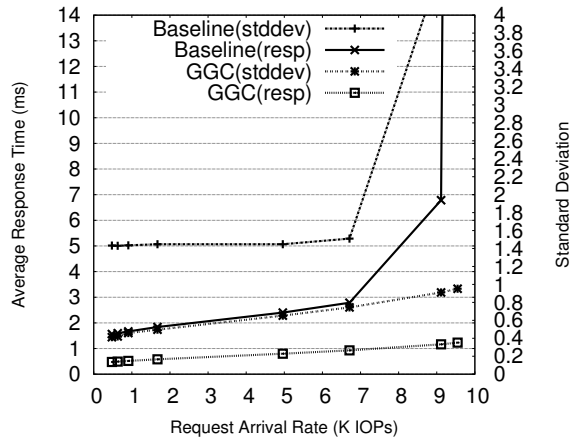
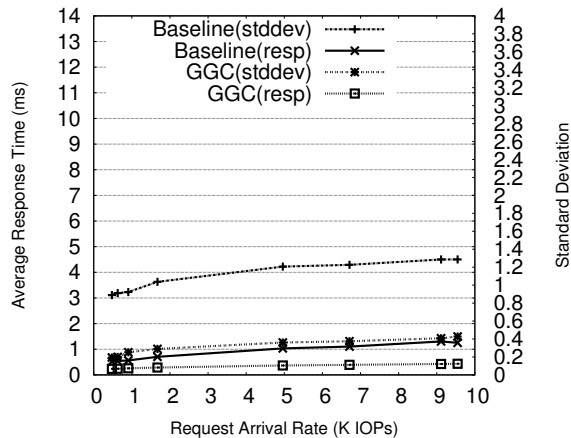


Fig. 6. Baseline and GGC normalized response time statistics for write and read dominant workloads. Average response time is plotted against the left y-axis and normalized standard deviation is plotted against the right y-axis.



(a) Write-dominant (80% Write)



(b) Read-dominant (20% Write)

Fig. 7. Average response times (left y-axis) and standard deviations (right y-axis) with respect to request arrival rates for baseline and GGC for write-dominant (a) and read-dominant (b) workloads.

in these measurements. This captures the sum of the device service time and the additional time spent waiting for the

device to begin to service the request.

#### D. Performance Analysis for HPC-like Workloads

We analyzed the response times of the GGC-enhanced RAID compared and the baseline schemes. The average response time for GGC was normalized with the respect to the baseline configuration in Figure 6. We note a 55% improvement for the *HPC(R)* read-dominated load and a 70% improvement for the *HPC(W)* write-dominated load. A system can be said to be robust if the response time can be predictable and it is capable of working with minimal variances. We observed the variance of response times for each workload in our experiments. Figure 10 presents standard deviations for each workload. GGC improves the response time by 73.8% on average. Also we observe GGC improves the robustness and predictability of the storage system.

#### E. Exploiting a Wide Range of Workload Characteristics

We have seen the improvement in response time and its variance for different realistic workloads with the GGC. In this experiment, we exploit a wide range of workloads in particular by varying the requests arrival rates. 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 quickens. This behavior does not provide a robust system response. In contrast, our GGC method exhibits lower average response times than the baseline, and a more gradual increase in variability. This confirms that GGC can help deliver a robust and stable system. For read-dominated workloads such as Figure 7(b), GGC continues to deliver improved performance and system robustness.

#### F. Scalability

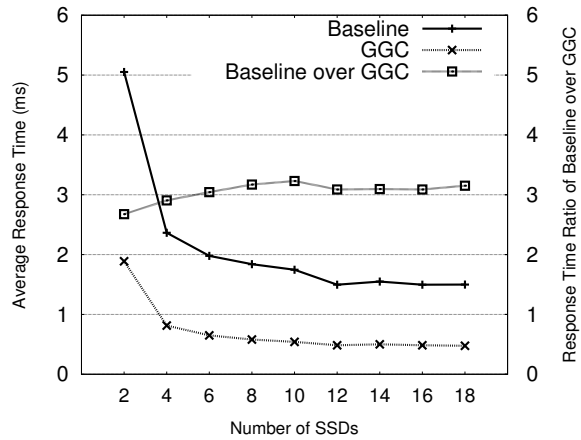


Fig. 8. Average response times (left y-axis) and standard deviation (right y-axis) of baseline and GGC schemes with respect to varying number of SSDs in a RAID array.

While experiments presented in previous sections were performed with eight SSDs in the RAID set, we also investigated how the number of devices in the array affected

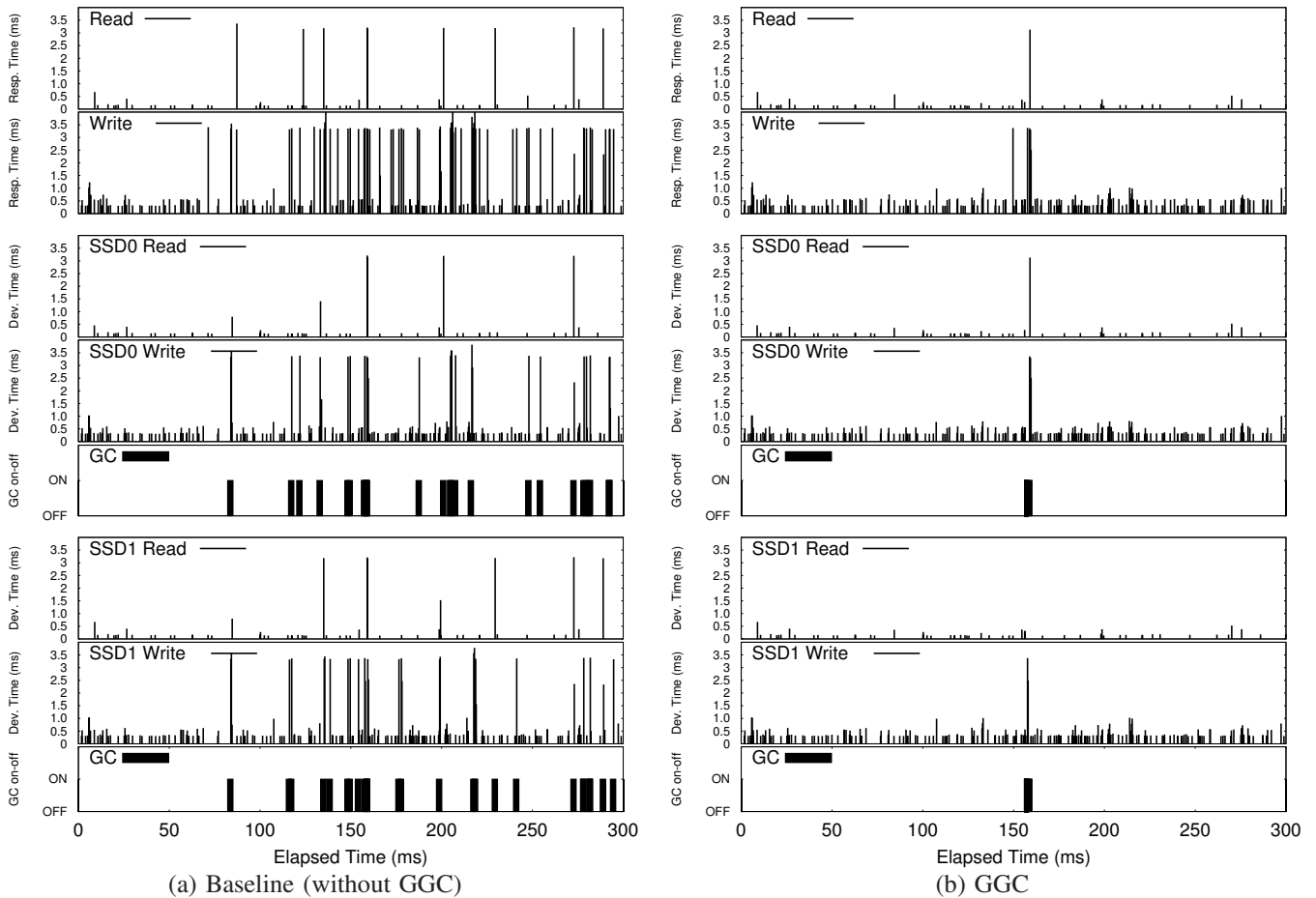


Fig. 9. Microscopic analysis for non-GGC vs. GGC. The first two rows show system response times of overall RAID for read and write requests. The 3rd to 5th rows show device service times for read and write and garbage collection duration for SSD-0. The 6th to 8th rows, similar to 3rd to 5th rows, show device service times and GC duration for SSD-1. We just present the timeseries analysis plots of 2 SSDs out of 8 SSDs used for RAID-0 in our evaluation.

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 improved their performance as number of SSDs increased. However, GGC maintains a performance edge over the baseline throughout the experiment. At two SSDs, the baseline response time was 2.7 times longer than GGC, and the margin grew to 3.2 times as we expanded the RAID set to 18 SSDs. It is interesting that the baseline requires eighth 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.

While we believe the results presented in Section V present a strong case for coordination of garbage collection in a RAID set, we note some constraints on this effort. Microsoft Research’s SSD simulator has been used in several studies [27], [37], but has not yet been validated against a hardware RAID set of SSDs. This effort is on-going, and has already demonstrated the performance degradation from uncoordinated GC events with actual hardware, indicating that this problem is not hypothetical.

### G. Microscopic Analysis

We perform a microscopic analysis of the impact of GGC on device response times and garbage collection invocations of individual SSDs in the RAID set. Figure 9 describes a set of consecutive requests serviced by two of the eight SSD devices in our simulated RAID.

The response time for each request was captured during a 300ms interval in the  $HPC(W)$  workload by both the baseline and our GGC method. As clearly indicated by Figures 9, baseline incurs more larger and more frequent overhead from GC collection 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 9(b), we clearly see the less number of spikes than the baseline without GGC. The total number of GC invoked from GCs are unchanged, however, many GC operations are called all at once, so those spikes are visible one. And we found that each SSD is composed of multiple packages. When GC is not coordinated, inside of SSD, each package can call the GC independently. By further forcing GC coordination across the packages, we could achieve significantly less number of GC

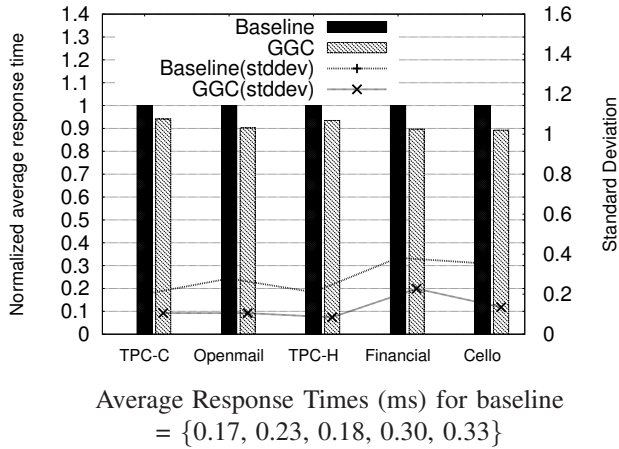


Fig. 10. Normalized average response times (left y-axis) and normalized standard deviation (right y-axis) for baseline and GGC configured RAID arrays under various enterprise-scale workloads.

spikes in GGC-enabled SSD RAID set.

#### H. Results for Enterprise-scale Realistic Workloads

So far in this paper we presented the impact of GGC for HPC-like workloads. We further analyzed GGC for enterprise-scale workloads. In the enterprise-scale workloads, different from the HPC-like workloads, the requests are smaller and more random (Refer to Table X and XI). Although less compared to HPC-like workloads, we observe GGC can not only improve average response times by 10% but also enhances the robustness and predictability of the RAID set of SSDs in Figure 10 for enterprise-scale workloads. The improvement is smaller compared to HPC-like workloads, as in the HPC domain the workloads are much more bursty and heavy in terms of request arrival's intensity.

## VI. CONCLUDING REMARKS

RAIDs of SSDs exhibit high performance variability due to uncoordinated garbage collection (GC) processes performed by individual SSDs within the RAID set. We propose a novel idea to significantly reduce this overhead, *Harmonia*, a global garbage collector, that coordinates the local GC process of the individual SSDs. We enhanced DiskSim and the Microsoft Research SSD simulator to implement one of GC coordination algorithms that we proposed. We evaluated the impact of GGC using this simulation environment against realistic workloads and observed the system response times and performance variability. Response time and performance variability were improved for all workloads in our study. In particular, for bursty workloads dominated by large writes, we observed a 69% improvement in response time and a 71% reduction in performance variability when compared to uncoordinated garbage collection.

We have identified several avenues for future study:

- In this paper, we have evaluated the reactive-method for *Harmonia* algorithm. However, we expect that the *Harmonia* will better perform by exploiting idle time

between I/O requests. Several on-line algorithms for detecting idle time have been suggested [31], [30]. We believe our proposed proactive methods (described in Section IV – *proactive soft-limit* and *proactive idle*) will be able to take advantage of such algorithms to further improve the efficiency of *Harmonia*.

- Our experiments with *Harmonia* are limited to synthetic HPC and realistic enterprise-scale workloads. Based on these experiments we conclude that *Harmonia* will perform better for workloads with large and bursty I/O requests. As a future work, we plan to exercise *Harmonia* with block-level traces gathered from a large-scale production HPC file system and analyze its performance.
- We empirically showed that uncoordinated local GC processes in RAID-5 or 6 configurations can hinder the overall performance and increase the performance variability. We plan to implement *Harmonia* for RAID-5 and 6 configurations in our simulation environment for further evaluation.

## ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their detailed comments which helped us improve the quality of this paper. Also we would like to thank Jason J. Hill for his technical support on the testbed setup. 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] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.
- [2] Jiri Bucy, John S. and Schindler, Steven W. Schlosser, and Gregory R Ganger. *The DiskSim Simulation Environment Version 4.0 Reference Manual*, May 2008.
- [3] Sandiego Supercomputer Center. Supercomputer uses flash to solve data-intensive problems 10 times faster, 2010. [http://www.sdsc.edu/News%20Items/PR110409\\_gordon.html](http://www.sdsc.edu/News%20Items/PR110409_gordon.html).
- [4] Feng Chen, David A. Koufaty, and Xiaodong Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, pages 181–192, New York, NY, USA, 2009. ACM.
- [5] Siddharth Choudhuri and Tony Givargis. Performance improvement of block based NAND flash translation layer. In *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '07, pages 257–262, New York, NY, USA, 2007. ACM.
- [6] Dong-Joo Chung, Tae-Sun Park, Dong-Ho Park, Sangwon Park and Lee, Sang-Won Lee, and Ha-Joo Song. System software for flash memory: A survey. In *Proceedings of the International Conference on Embedded and Ubiquitous Computing*, pages 394–404, August 2006.
- [7] Kurt B. Ferreira Ferreira, Patrick Bridges, and Ron Brightwel. Characterizing application sensitivity to OS interference using kernel-level noise injection. In *SC '08: Proceedings of Supercomputing*, pages 1–12, 2008.
- [8] Eran Gal and Sivan Toledo. Algorithms and data structures for flash memories. *ACM Computing Survey*, 37(2):138–163, 2005.

- [9] Gregory R. Ganger. Generating representative synthetic workloads: An unsolved problem. In *International Conference on Management and Performance Evaluation of Computer Systems*, page 12631269, 1995.
- [10] Gary Greider. HPC I/O and file systems issues and perspectives, 2006. <http://www.dtc.umn.edu/disc/isw/presentations/isw46.pdf>.
- [11] Aayush Gupta, Youngjae Kim, and Bhuvan Urganonkar. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, pages 229–240, New York, NY, USA, 2009. ACM.
- [12] Sudhanva Gurumurthi, Anand Sivasubramaniam, and Vivek 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, Washington, DC, USA, 2005. IEEE Computer Society.
- [13] Jiahua He He, Arun Jagatheesan, Sandeep Gupta, Jeffrey Bennett, and Allan Snaveley. DASH: a recipe for a flash-based data intensive supercomputer. In *SC '10: Proceedings of Supercomputing*, November 2010.
- [14] HP-Labs. The Openmail Trace. <http://tesla.hpl.hp.com/opensource/openmail/>.
- [15] Intel. IntelXeon Processor X5570 8M Cache, 2.93 GHz, 6.40 GT/s Intel QPI. <http://ark.intel.com/Product.aspx?id=37111>.
- [16] Intel. Intel X25-E Extreme 64GB SATA Solid-State Drive SLC. <http://www.intel.com/design/flash/nand/extreme/index.htm>.
- [17] Heeseung Jo, Jeong-Uk Kang, Seon-Yeong Park, Jin-Soo Kim, and Joonwon Lee. FAB: Flash-aware buffer management policy for portable media players. *IEEE Transactions on Consumer Electronics*, 52(2):485–493, 2006.
- [18] Eliahu Ibrahim Jury. *Theory and Application of the Z-Transform Method*. Wiley-Interscience, 1964.
- [19] Jeong-Uk Kang, Heeseung Jo, Jin-Soo Kim, and Joonwon Lee. A superblock-based flash translation layer for NAND flash memory. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software, EMSOFT '06*, pages 161–170, New York, NY, USA, 2006. ACM.
- [20] Hyojun Kim and Seongjun 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)*, pages 1–14, February 2008.
- [21] Youngjae Kim, Sarp Oral, Dave A Dillow, Feiyi Wang, Douglas Fuller, Stephen Poole, and Galen M. Shipman. An empirical study of redundant array of independent solid-state drives (RAIS). In *Technical Report, ORNL/TM-2010/61, Oak Ridge National Laboratory, National Center for Computational Sciences*, March 2010.
- [22] Junghee Lee, Youngjae Kim, Galen M Shipman, Sarp Oral, Jongman Kim, and Feiyi Wang. A semi-preemptive garbage collector for solid state drives. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2011.
- [23] Sang-Won Lee and Bongki 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, New York, NY, USA, 2007. ACM.
- [24] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sang-won Park, and Ha-Joo Song. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Trans. Embed. Comput. Syst.*, 6(3):18, 2007.
- [25] Sungjin Lee, Dongkun Shin, Young-Jin Kim, and Jihong Kim. LAST: Locality-aware sector translation for NAND flash memory-based storage systems. *SIGOPS Oper. Syst. Rev.*, 42(6):36–42, 2008.
- [26] Min Li, Sudharshan S. Vazhkudai, Ali R. Butt, Fei Meng, Xiaosong Ma, Youngjae Kim, Christian Engelmann, and Galen M. Shipman. Functional partitioning to optimize end-to-end performance on many-core architectures. In *SC '10: Proceedings of Supercomputing*, November 2010.
- [27] Teng Li, Tarek El-Ghazawi, and H. Howie Huang. Reconfigurable active drive: an FPGA accelerated storage architecture for data-intensive applications. In *2010 Symposium on Application Accelerators in High Performance Computing (SAAHPC'10)*, 2010.
- [28] LSI. MegaRAID SAS 9260-8i RAID Card. <http://www.lsi.com/channel/products/megaraid/sasata/9260-8i/index.html>.
- [29] 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.
- [30] Ningfang Mi, Alma Riska, Evgenia Smirni, and Erik Riedel. Enhancing data availability in disk drives through background activities. In *DSN '08: Proceedings of the Symposium on Dependable Systems and Networks*, pages 492–501, June 2008.
- [31] Ningfang Mi, Alma Riska, Qi Zhang, Evgenia Smirni, and Erik Riedel. Efficient management of idleness in storage systems. *Trans. Storage*, 5:4:1–4:25, June 2009.
- [32] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. Migrating server storage to SSDs: analysis of tradeoffs. In *EuroSys '09: Proceedings of the 4th ACM European conference on Computer systems*, pages 145–158, New York, NY, USA, 2009. ACM.
- [33] Henry Newman. What is HPCS and how does it impact I/O, 2009. <http://wiki.lustre.org/images/5/5a/NewmanMayLustreWorkshop.pdf>.
- [34] H. Nijjima. Design of a solid-state file using flash eeprom. *IBM Journal of Research and Development*, 39(5):531–545, 1995.
- [35] Sarp Oral, Feiyi Wang, David A. Dillow, Ross Miller, Galen M. Shipman, and Don Maxwell. Reducing application runtime variability on Jaguar XT5. In *CUG '10: Proceedings of Cray User's Group (CUG) Meeting*, May 2010.
- [36] Seon-yeong Park, Dawoon Jung, Jeong-uk Kang, Jin-soo Kim, and Joonwon 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, New York, NY, USA, 2006. ACM.
- [37] Seon-yeong Park, Euiseong Seo, Ji-Yong Shin, Seungryoul Maeng, and Joonwon Lee. Exploiting internal parallelism of flash-based SSDs. *IEEE Comput. Archit. Lett.*, 9(1):9–12, 2010.
- [38] David Patterson, Garth Gibson, and Randy 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.
- [39] Fabrizio Petrini, Darren J. Kerbyson, and Scott Pakin. The case of the missing supercomputer performance: achieving optimal performance on the 8,192 processors of ASCII Q. In *SC '03: Proceedings of Supercomputing*, pages 1–12, 2003.
- [40] Vijayan Prabhakaran, Thomas L. Rodeheffer, and Lidong Zhou. Transactional flash. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation, OSDI'08*, pages 147–160, Berkeley, CA, USA, 2008. USENIX Association.
- [41] Steven L. Pratt and Dominique A. Heger. Workload dependent performance evaluation of the Linux 2.6 I/O schedulers. In *Linux Symposium*, July 2004.
- [42] Abhishek Rajimwale, Vijayan Prabhakaran, and John D. Davis. Block management in solid-state devices. In *Proceedings of the USENIX Annual Technical Conference*, June 2009.
- [43] Seagate. Seagate Cheetah 15K.7 Disc Drive. [http://www.seagate.com/docs/pdf/datasheet/disc/ds\\_cheetah\\_15k\\_7.pdf](http://www.seagate.com/docs/pdf/datasheet/disc/ds_cheetah_15k_7.pdf).
- [44] Eric Seppanen, Matthew T. O'Keefe, and David J. Lilja. High performance solid state storage under Linux. In *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) 2010*, pages 1–12, 2010.
- [45] Prashant Shenoy and Harrick M. Vin. Cello: A Disk Scheduling Framework for Next Generation Operating Systems. *Real-Time Syst.*, 22(1/2):9–48, 2002.
- [46] Storage-Performance-Council. OLTP Trace from UMass Trace Repository. <http://traces.cs.umass.edu/index.php/Storage/Storage>.
- [47] Super Talent. Super Talent 128GB UltraDrive ME SATA-II 25 MLC. [http://www.supertalent.com/products/ssd\\_detail.php?type=UltraDrive%20ME](http://www.supertalent.com/products/ssd_detail.php?type=UltraDrive%20ME).
- [48] Transaction-Processing-Perforamnce-Council. TPC-C, an OLTP Benchmark. <http://www.tpc.org/tpcc/>.
- [49] Andrew Uselton. Deploying server-side file system monitoring at NERSC. In *CUG '09: Proceedings of Cray User's Group (CUG) Meeting*, 2009.
- [50] Jianyong Zhang, Anand Sivasubramaniam, Hubertus Franke, Natarajan Gautam, Yanyong Zhang, and Shailabh Nagar. Synthesizing representative I/O workloads for TPC-H. In *HPCA '04: Proceedings of the 10th International Symposium on High Performance Computer Architecture*, page 142, Washington, DC, USA, 2004. IEEE Computer Society.