2018 IEEE International Parallel and Distributed Processing Symposium

GC-aware Request Steering with Improved Performance and Reliability for SSD-based RAIDs

Suzhen Wu‡, Weidong Zhu‡, Guixin Liu‡, Hong Jiang*, Bo Mao†✉
‡Computer Science Department of Xiamen University, China
*Department of Computer Science & Engineering at University of Texas-Arlington, USA
†Software School of Xiamen University, China
✉Corresponding author: maobo@xmu.edu.cn

***Abstract*—SSD-based RAIDs have been widely deployed in high-end enterprise systems to provide high-performance and highly reliable storage for data-intensive computing. However, SSD-based RAIDs suffer from significant performance degradation whenever user I/O requests conflict with the ongoing Garbage Collection (GC) operations which introduces tail latency. Moreover, the performance characteristics of SSDs make the traditional HDD-based RAID reconstruction algorithms are not compatible with or suitable for SSD-based RAIDs. In this paper, we proposed GC-aware Request Steering (short for *GC-Steering*), a scheme aware of the GC process within an SSD-based RAID, to significantly boost the performance and reliability of SSD-based RAIDs. GC-Steering effectively outsources the popular read requests and all write requests addressed to the SSD currently in the GC state to a staging space such as a dedicated spare SSD or the reserved space of each SSD within the RAID. GC-Steering also accelerates the performance of the failure-recovery process by both request steering and parallel recovery. Our extensive evaluations on a lightweight GC-Steering prototype driven by HPC-like and real-world enterprise workloads show that the GC-Steering scheme significantly reduces the average response time by an average of 63.3% and 65.8%, compared with the state-of-the-art LGC and GGC schemes. Moreover, GC-Steering scheme also significantly reduces the average response times by an average of 55.7% during RAID reconstruction than the normal state.**

***Index Terms*—SSD-based RAIDs, Garbage Collection, Request Steering, Parallel Reconstruction**

I. Introduction

Flash memory technology is disrupting the storage media market, leading to a significant evolutionary investment and innovation in the storage systems market [4]. Flash-based Solid State Disks (SSDs) have emerged as alternatives to Hard Disk Drives (HDDs), increasingly replacing or coexisting with HDDs in desktops, enterprise storage systems and large-scale data centers [13], [24]. In addition to read and write operations that are common in both HDDs and SSDs, flash-based SSDs require additional time-consuming erase and garbage collection (GC) operations. The process time of an erase operation is an order of magnitude more than that of a read or write operation [1], [29]. Moreover, the performance of the incoming user I/O requests during the GC period will be significantly degraded by the GC process [18], [31], due to the sever contention between the external user I/O requests and the internal GC-induced requests. The internal GC-induced performance degradation is the primary cause of slowdowns for SSD-based storage systems which causes significant tail latency [36], [38].

On the other hand, a single SSD cannot satisfy the performance, capacity and reliability requirements of the high-performance computing and enterprise environments. Thus it is necessary to apply the RAID (Redundant Array of Independent Disks) [26] algorithm to SSDs to build large-scale SSD-based storage systems with high performance and high reliability [2], [6]. Similar to HDD-based RAIDs, the performance of SSD-based RAIDs is restricted by the slowest device in the array. Therefore, the intermittent performance degradation of the individual SSDs within an SSD-based RAID caused by the GC process will cause SSD-based RAIDs to exhibit serious performance variability [12], [36]. Kim *et al.* [17] found that the uncoordinated GC processes on individual SSDs significantly degraded the performance of SSD-based RAIDs.

Moreover, recent studies on the data collected from a majority of flash-based SSDs at Facebook data centers over a period of nearly 4 years reveal that SSD failures are relatively common events with 4.2%-34.1% of SSDs reporting uncorrectable errors [23]. Similar studies on flash reliability based on the data collected over a 6-year period on SSDs used in production in Google data centers find that 1%-2% of flash-based SSDs are replaced annually due to the suspected hardware problems over the first 4 years in the production [28]. These studies imply that it is important and urgent to investigate and improve the failure-recovery process for SSD-based RAIDs.

To this end, we propose GC-aware Request Steering (short for *GC-Steering*), a scheme aware of the GC process within an SSD-based RAID, to address both the performance and reliability issues of SSD-based RAIDs alluded to above. The main idea behind GC-Steering is to fully exploit the workload characteristics and utilize the pre-reserved space (*e.g.*, staging space such as a dedicated SSD or the reserved space of each SSD within RAID) in an SSD-based RAID to alleviate the negative impact of the GC process on the system performance and reliability. By proactively migrating "hot" read data to the staging space, the subsequent read requests addressed to an SSD currently in the GC state can be alternatively serviced by the staging space without being interfered by the ongoing GC process. The incoming write requests addressed to an SSD

1530-2075/18/$31.00 ©2018 IEEE
DOI 10.1109/IPDPS.2018.00039

currently in the GC state are also temporally redirected to the staging space. Consequently, the contention between the external user I/O requests and the internal GC-induced requests is significantly alleviated, if not completely eliminated. For the RAID failure-recovery process, by temporarily redirecting all write requests and popular read requests originally targeting at the degraded SSD-based RAID to the staging space, the recovery speed is also accelerated. Moreover, by exploiting the parallel access characteristics of flash-based SSDs, the failure-recovery process can be further improved.

The rest of this paper is organized as follows. Background and motivation are presented in Section II. We describe the design details of the GC-Steering scheme in Section III. The performance evaluation is presented in Section IV. The related work is presented in Section V. We conclude this paper and point out the directions for future research in Section VI.

II. Background and Motivation

In this section, we first describe how GC operations degrade the performance of SSD-based RAIDs. Then we elaborate on why the existing RAID failure-recovery algorithms are not suitable for SSD-based RAIDs. We conclude the section by presenting and analyzing the workload characteristics to motivate our new GC-aware performance optimization and failure-recovery scheme for SSD-based RAIDs.

A. The adverse impact of GC on SSD-based RAIDs

Due to the unique physical features of NAND flash, write requests are serviced out-of-place rather than in-place. In flash memory, data can only be written to erased pages (*a.k.a.*, free pages), where the in-place (before-write) pages become invalid (stale) after out-of-place write operations. At some point, invalid pages in a block, called a victim block, must be freed by copying (read followed by write) the data in the valid pages in that block into a free block, before the victim block is erased and made available for subsequent write data. This process is known as the Garbage Collection (GC) process that significantly affects the user I/O performance of SSD-based storage systems [14], [18], [38]. Equally important, since each memory cell in a block has a limited number of erase cycles, GC also significantly affects the reliability (endurance) of SSD-based storage systems. Therefore, how to address the performance and reliability issues caused by GC of SSDs has become a critically important challenge when deploying flash-based SSDs into HPC and enterprise storage systems.

This negative GC impact on an SSD-based RAID is arguably much more significant than on an individual SSD [8]. For example, the frequency of GC operations in the former consisting of multiple SSDs, say N, is at least N times higher than that in the latter. Consequently, user I/O requests are that much more likely to be blocked because one of the SSDs within RAID is currently in the GC state, adversely affecting the response performance accordingly. Kim *et al.* [17] conducted detailed experiments and their evaluation results reveal that the uncoordinated local GC (short for LGC) process on individual SSDs in an SSD-based RAID degrades the performance of SSD-based RAIDs and causes a serious performance variability. Figure 1 shows an example that depicts how the default LGC scheduling scheme affects the performance of SSD-based RAIDs. In an extreme case, such as one illustrated in the figure, the uncoordinated but interleaved occurrences of the GC processes of the individual SSDs in an SSD-based RAID can render the SSD-based RAID in the degraded performance state almost all the time. The performance variability will no doubt lead to Service Level Agreement (SLA) and Service Level Objective (SLO) violations, thus affecting the system availability [9]. Moreover, the GC-induced performance degradation causes significant tail latency problem [7], [38]

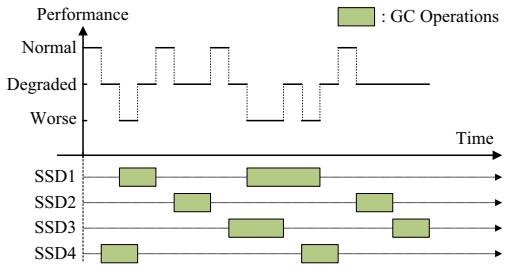


Fig. 1. An example of the impact of GC operations on the performance of an SSD-based RAID with 4 SSDs.

Based on this observation, Kim *et al.* [17] proposed a Globally coordinated GC (short for GGC) strategy to alleviate the performance variability of an SSD-based RAID. GGC forces all the SSDs in an SSD-based RAID to start the GC operations at the same time. However, during the GC period, the SSD-based RAID is unavailable for the user applications from the point of view of Quality of Service (QoS) [9]. The reason is that all the SSDs in an SSD-based RAID are busy with dealing with the GC operations and have no free resources to service the external user I/O requests. Although GGC can guarantee a much longer high-performance period, it introduces many unavailable periods for the end users, which can lead to SLA and SLO violations for many applications, particularly those required to be available 24/7. To guarantee the performance requirement of the end users, it is desirable for SSD-based RAIDs to be available to service the user I/O requests all the time. The key approach to reducing the performance degradation and alleviating the performance variability is to reduce the interference between the external user I/O requests and the internal GC-induced requests.

B. Failure-recovery for SSD-based RAIDs

For HDDs, sequential read/write operations are much faster than their random counterparts. During the RAID failure-recovery (*a.k.a.*, reconstruction) period, the mechanically operated disk head moves between the reconstruction area and the user I/O accessed area. Thus, existing failure-recovery algorithms for HDD-based RAIDs try to make the reconstruction I/O requests sequential on HDDs. However, unlike

HDDs, flash-based SSDs are made of semiconductor chips and have no moving parts (*i.e.*, mechanical positioning parts). By exploiting the internal parallelism of flash-based SSDs, random accesses can be performed much more efficiently on SSDs than on HDDs [5], [15]. However, the read-write performance asymmetry of flash-based SSDs, in which the read performance is much higher (about 10X) than the write performance, means that the hot-spare SSD for replacement during RAID reconstruction will likely become a performance bottleneck when the lost data is regenerated and written to it from the relevant data read (in parallel) from all the surviving SSDs.

On the other hand, a recent Samsung report reveals that failures of flash-based SSDs typically occur in the SSD controller rather than in the individual silicon chips on an SSD [27], which renders the whole SSD unusable and triggers the RAID reconstruction process immediately. Recent studies on the data collected from a majority of flash-based SSDs installed in both Facebook data centers and Google data centers reveal that SSD failures are common events with 4.2%-34.1% of SSDs reporting uncorrectable errors [23] and 1%-2% of flash-based SSDs being replaced annually due to suspected hardware problems [28]. These trends and findings make it abundantly clear why it is important to speed up the RAID failure-recovery process in SSD-based RAIDs to achieve high performance and high reliability.

C. Trace characteristics and motivation

Understanding the workload characteristics is important for the design of storage systems, particularly for flash-based storage systems where the unique features of flash memory and their interactions with workloads accentuate this importance [21]. For this purpose, we categorize and differentiate SSD pages into the following three distinctive types based on how data stored in a page is accessed by the workloads:

(1) Read intensive data (RI): If almost all the accesses (>90%) to a data page are read requests, this page is defined as read intensive;
(2) Write intensive data (WI): If almost all the accesses (>90%) to a data page are write requests, this page is defined as write intensive;
(3) Mixed data (MIX): If the accesses to a data page are interleaved with reads and writes, this page is defined as mixed.

Figure 2 shows the access patterns on these three types of data pages in MSR traces as that shown in Table I. Figure 2(a) illustrates the distribution of read requests among the read intensive data pages and mixed data pages and Figure 2(b) shows the distribution of write requests among write intensive data and mixed data pages. An average of 89.8% read requests access the read intensive data pages and an average of 95.5% write requests access the write intensive data pages. Only a small part of the requests access the mixed data pages. These observations are also consistent with those reported in the previous studies [19]. More importantly, these workload

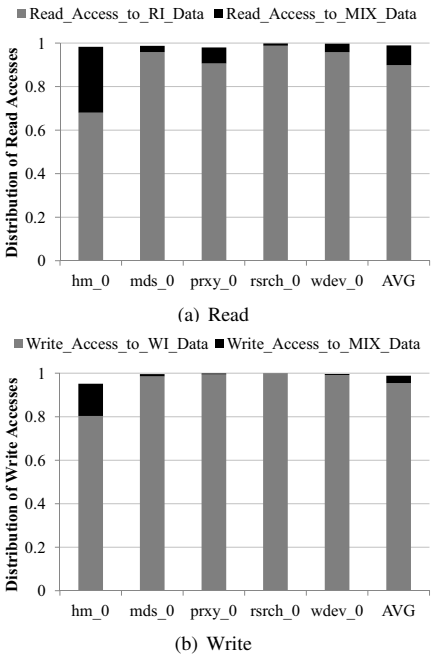


Fig. 2. Distribution of read/write requests on the three types of data pages.

characteristics imply that the hot read data blocks are not frequently updated by write requests.

The above analysis and observations of flash-based SSDs and SSD-based RAIDs in HPC and enterprise environments clearly indicate the paramount importance of understanding and significantly improving performance and reliability of SSD-based RAIDs that are fundamentally different from that of HDD-based RAIDs. This, combined with awareness of the workload characteristics, motivates design of GC-Steering that proactively migrates the hot data blocks within an SSD-based RAID to a pre-reserved space (*e.g.*, a staging space such as a dedicated SSD or the reserved space within individual SSDs of RAID) to significantly alleviate, if not entirely eliminate, the contention between the external user I/O requests and the internal GC-induced requests. This helps simultaneously improve the user I/O performance and RAID failure-recovery performance.

III. Design of GC-Steering

In this section, we first present a system overview of GC-Steering, followed by a description of the request processing workflow and the RAID reconstruction workflow in GC-Steering. The data consistency issues of GC-Steering are discussed at the end of this section.

A. System overview

The main idea behind GC-Steering is to temporarily redirect the popular read requests and all write requests originally addressed to any SSD in the GC state to a staging space, thus significantly reducing the contention between the external user I/O requests and the internal GC-induced I/O requests.

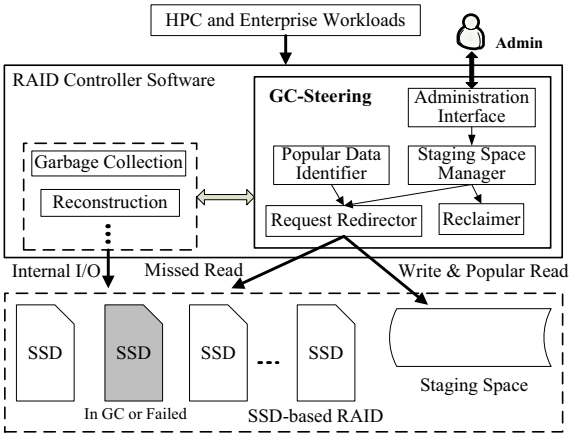


Fig. 3. System overview of GC-Steering within the RAID controller software.

Furthermore, by redirecting many requests away from the degraded SSD-based RAID to the staging space during RAID reconstruction, both the user performance degradation and reconstruction performance degradation caused by the failure-recovery process can be alleviated.

Figure 3 shows a system overview of GC-Steering. In our design, GC-Steering can be incorporated into any existing SSD-based RAID schemes, such as hardware RAID and software RAID. As an example, this figure illustrates how GC-Steering is augmented to the RAID controller software with five key functional components: Administration Interface, Popular Data Identifier, Staging Space Manager, Request Redirector and Reclaimer. *Popular Data Identifier* is responsible for monitoring the popularity of read data blocks to help *Staging Space Manager* migrate popular read data blocks to the staging space. *Staging Space Manager* is responsible for migrating popular data blocks from the operational SSD-based RAID to the staging space and managing the data layout of the redirected data in the staging space. *Request Redirector* is responsible for redirecting all write requests and popular read requests to the staging space during GC and reconstruction periods, while *Reclaimer* is responsible for reclaiming the write data back to the SSD-based RAID after the GC or reconstruction process completes.

GC-Steering can redirect user I/O requests to different persistent configurations of SSD-based storage devices, for example, a dedicated SSD or the reserved space of each SSD within an SSD-based RAID. In what follows, we will illustrate the workflow based on these two design configurations. Moreover, GC-Steering is automatically activated when an SSD starts to process the GC operation or when the reconstruction thread is initiated, and is deactivated when all write data in the staging space is reclaimed back to the SSD-based RAID. Thus, GC-Steering does not significantly affect the normal operations for SSD-based RAIDs.

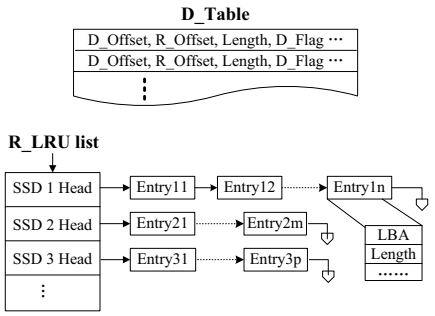


Fig. 4. Data structures of GC-Steering.

B. Data structures

GC-Steering relies on two important data structures to record the redirected data blocks and identify popular read data, namely, *D_Table* and *R_LRU*, as shown in Figure 4. D_Table, a log table to manage the redirected data in the staging space, contains the logs of all redirected data blocks, including four important variables. *D_Offset* and *R_Offset* indicate the offsets of the redirected data block in the SSD-based RAID and the staging space, respectively. *Length* indicates the length of the redirected data block and *Flag* indicates whether it is redirected read data block from the SSD-based RAID (*Flag is set to be false*) or redirected write data block from the user application(*Flag is set to be true*).

R_LRU, an LRU-style list to identify the popular read data blocks on each SSD, stores the information (*i.e.*, D_offset and Length of read data) of the most recently read requests for each SSD within the SSD-based RAID. Based on R_LRU, the popular read data can be identified and proactively migrated to the staging space. In order to reduce the space overhead of the staging space, not all popular read data blocks within the SSD-based RAID are migrated. In our current design, only up to 10% of popular data blocks are migrated. Moreover, when the popular data blocks in the SSD-based RAID are read by user applications, they are concurrently migrated to the staging space and D_Table is accordingly updated with *Flag* set to false. Thus, the migration overhead of popular data blocks is reduced without affecting the system performance of the SSD-based RAID.

C. Request processing workflow

When an SSD within the SSD-based RAID is dealing with the GC operation, the incoming user I/O requests addressed to that SSD are checked to determine whether they should be issued to that SSD or the staging space. Figure 5 shows the request processing workflow in GC-Steering for an incoming read request, assuming that SSD 4 is currently in the GC state. For a read request, GC-Steering first checks whether there is an entry associated with the requested data in D_Table or not. If such an entry is found in D_Table, the read request is serviced by the staging space. If not, the read request is processed by the SSD-based RAID. On the other hand, for a read request that is not addressed to the SSD in the GC state, it is directly

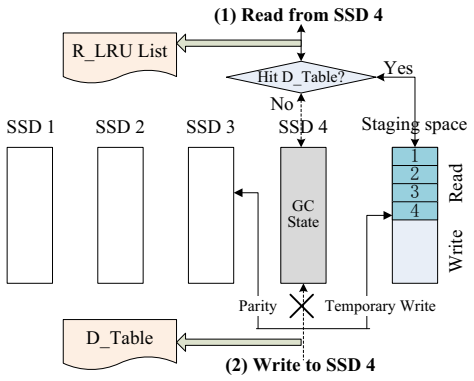


Fig. 5. The I/O processing workflow in GC-Steering.

serviced by the corresponding SSDs within the SSD-based RAID. R_LRU is updated accordingly to record the popular read data.

For all write requests addressed to the SSD currently in the GC state, GC-Steering uses the write redirection scheme to temporarily store the write data in the staging space. As shown in Figure 5, a write request addressed to the SSD currently in the GC state is replaced by a write request to the staging space. In order to maintain the reliability of the redirected write data, GC-Steering concurrently updates the corresponding parity to its correct position in the same stripe in the SSD-based RAID. In this case, if the redirected write data in the staging space is lost, it can be reconstructed from the surviving SSDs within RAID. When a write request is redirected to the staging space, a corresponding entry is created and added to D_Table. Consequently, the incoming read requests must be checked first in D_Table to keep the fetched data always up-to-date.

If the staging space is configured as a dedicated SSD, the data layout is the same as that shown in Figure 5. If the staging space is the reserved space in each SSD within the SSD-based RAID, the data layout is quite different. First, the hot read data is stored in an interleaved fashion and organized in a RAID0-style for redundancy purpose. Since the loss of the hot read data in the staging space does not cause data failure, RAID0, which does not incur any write amplification, is suitable for guaranteeing high performance and high reliability. Second, the write data is also stored in an interleaved way but organized in a RAID1-style for redundancy. RAID1 can provide high reliability against an SSD failure. Upon the failure of an SSD within RAID, the redirected write data in the reserved space of that SSD can be correctly recovered from its mirroring SSD. Although RAID1 has lower storage efficiency than RAID5, the staging space is only used temporally to store the redirected write data during GC or RAID reconstruction period, thus incurring low storage overhead.

When an SSD in the GC state completes its GC operations, the redirected write data in the staging space will be reclaimed back to its correct location in RAID. Since the corresponding parity has already been updated in its correct position when the write data is redirected to the staging space, GC-Steering does not need to consider the corresponding parity when reclaiming the redirected write data. To ensure data consistency, the corresponding log entry of the reclaimed data is deleted from D_Table after the reclaim process completes. Moreover, to improve the efficiency of the reclaim process, the sequential data blocks in the staging space are first merged into a large data block before the reclaim process.

D. Reconstruction workflow

Upon the failure of an SSD within RAID, the RAID reconstruction process is immediately initiated. The replacement SSD can be a newly added SSD (indicated by ①) or the staging space (indicated by ②), as shown in Figure 6. If the replacement SSD is a newly added SSD, GC-Steering focuses on redirecting user I/O requests to the staging space during RAID reconstruction to improve both the reconstruction efficiency and user I/O performance. During RAID reconstruction, all write requests addressed to the degraded RAID are redirected to the staging space after determining whether they should overwrite their previous locations or write to new locations according to D_Table. On the other hand, for each read request, D_Table is first checked to determine whether the read data is in the staging space. If the read request hit D_Table, it is directly serviced by the staging space instead of the degraded RAID that is busy dealing with the failure-recovery process. Otherwise, it is serviced by the degraded SSD-based RAID. After the reconstruction process completes, the redirected write data is reclaimed back to the SSD-based RAID. To ensure data consistency, the corresponding entry of the reclaimed data in D_Table is deleted after the reclaim operation completes.

If the replacement SSD is the staging space, the previously redirected write data in the staging space must be first reclaimed back to the SSD-based RAID before initiating the reconstruction process. After completing the reclaim process, the data originally stored in the staging space is invalidated and the RAID reconstruction process is initiated. In the GC-Steering design, the staging space has two configurations, *i.e.*, a dedicated SSD or the reserved space of each SSD within SSD-based RAID. If the staging space is a dedicated SSD, the degraded RAID performs the traditional RAID reconstruction workflow. If the staging space is the reserved space of each SSD within SSD-based RAID, the parallel reconstruction workflow is performed by the degraded RAID. In this case, the reconstructed data of the failed SSD is written in parallel to the staging space. Although the read area and the write area within each SSD is interleaved, SSDs can process them concurrently without any disk head seek overhead that HDDs require. Therefore, the parallel access feature of SSDs can be fully exploited to improve the RAID reconstruction performance [5], [15]. Moreover, it must be noted that the staging space is organized with the same redundancy as the SSD-based RAID after the previously redirected data stored in it is reclaimed back. If a second SSD fails, the data in the staging space of the failed SSD is first reconstructed to the

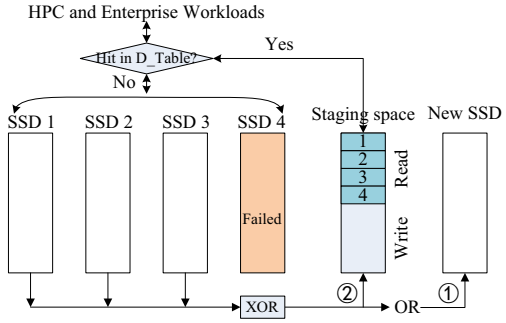


Fig. 6. The reconstruction workflow in GC-Steering.

newly added SSD, followed by reconstructing the remaining lost data of the failed SSD.

E. Data consistency

Data consistency in the GC-Steering design includes the following two aspects: (1) The redirected write data must be reliably stored in the staging space until the data reclaim process completes, and (2) The key data structure (*i.e.*, D_Table) must be safely stored.

First, the redirected write data must be reliably stored in the staging space. Since the staging space may also fail before the redirected write data is reclaimed, the redirected write data being stored in the staging space must be protected by a redundancy scheme. In GC-Steering, when writing data to the staging space, the corresponding parity in the same stripe is concurrently updated to its correct location to prevent data loss caused by a possible failure of the staging space. If the staging space is a dedicated SSD, the failure of the dedicated SSD does not cause data loss because the redirected write data can be reconstructed by the data and parity on the surviving SSDs. If the staging space is the reserved space of each SSD within SSD-based RAID, the write data is protected by the RAID1-style redundancy. Thus, the failure of a single SSD does not cause data loss. Moreover, the redirected write data is sequentially stored in the staging space with the append-only mode. When the redirected write data in the staging space is reclaimed, the contiguous blocks can be erased effectively to free up space for subsequent write data.

Second, to prevent the loss of D_Table in the event of a power supply failure or a system crash, GC-Steering stores the contents of D_Table in a non-volatile RAM (NVRAM). Since D_Table is in general small, it will not incur notable extra hardware cost to the RAID system. In order to reduce the write penalty due to D_Table updates, GC-Steering stores the contents of D_Table in battery-backed RAM, a *de facto* standard form of NVRAM. In this case, a small battery can delay shutdown until the content of D_Table in the RAM is safely saved to an area of SSDs. On the other hand, in order to improve the write performance by using the write-back strategy, NVRAM is commonly deployed in the RAID controller. Consequently, it is easy and reasonable to use NVRAM to store the contents of D_Table.

IV. Performance Evaluations

In this section, we first describe the experimental setup and methodology. Then we evaluate the effectiveness and performance of our proposed GC-Steering scheme by comparing it against the relevant state-of-the-art schemes LGC and GGC through extensive trace-driven evaluations, with HPC-like workloads and realistic enterprise workloads.

A. Experimental setup and methodology

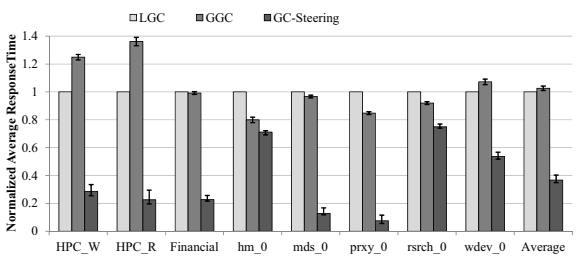
We implement a lightweight prototype of GC-Steering on top of the Linux software RAID (*i.e.*, Linux MD). All experiments are conducted on a Dell PowerEdge T320 node with an Intel Xeon E5-2407 CPU and 16GB memory. In this system, a SAMSUNG HE253GJ SATA HDD (250GB) is used to host the operating system (Ubuntu 14.04 with Linux kernel version 3.13), the Linux software RAID module and other software. An LSI Logic MegaRAID SAS 2208 controller is used to connect 7 Intel DC S3510 120GB SSDs.

In the evaluation, we compare the performance of GC-Steering with two state-of-the-art schemes, LGC and GGC [17], in terms of average response time and tail latency. The GC activities within SSDs can be detected by the RB-Explorer [32]. Moreover, all the RAID schemes use the same number of SSDs to provide a fair comparison. To ensure that SSDs reach the steady state (*i.e.*, with regular GCs) when new write requests arrive during the experimental running period, we fill the entire space on each SSD with valid data prior to measuring the performance, a common practice called simulation "warm-up" [17]. In the GGC scheme, when any one SSD within a RAID initiates its GC process, all the other SSDs of the RAID also initiate their GC processes. By default, the staging space in the GC-Steering prototype is the pre-reserved space of each SSD within the SSD-based RAID.

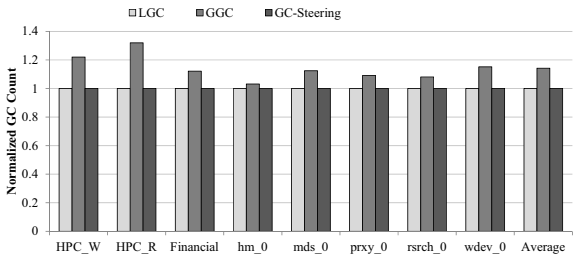
We use a mixture of HPC-like workloads and realistic enterprise-scale workloads to study the performance of our proposed GC-Steering scheme. For HPC-like workloads, we choose the read/write and bursty workloads, *i.e.*, the HPC_W and HPC_R traces, whose main characteristics are described in Table I. For realistic enterprise-scale workloads, the Fin1 trace is collected from the OLTP applications running at a large financial institution [3]. The other traces are collected from storage volumes in an enterprise data center by Microsoft Research Cambridge (MSR traces for short) [3]. Since most MSR traces are one-day workloads with bursty and idle periods, we only choose one-hour traces with bursty periods to replay. These traces represent different access patterns in terms of read/write ratio, number of requests and average request size, as summarized in Table I.

B. Performance in the normal state

(1) Results and analysis. We first conduct experiments on a RAID5 system consisting of 5 SSDs with a stripe unit size of 64KB, driven by the different workloads in the normal operational state. Figure 7(a) and Figure 7(b) show the comparisons of average response times and GC counts,



(a) The normalized average user response times



(b) The normalized GC counts

Fig. 7. Comparisons of normalized average response time and GC counts for LGC, GGC and GC-Steering.

TABLE I
THE TRACE CHARACTERISTICS.

Type	Traces	Trace Characteristics		
		Read Ratio	Num. of Req.	Avg. Req. Size
HPC	HPC_W	20.1%	500,000	510.5 KB
	HPC_R	79.9%	500,000	510.5 KB
MSR	Fin1	32.8%	5,334,987	11.9 KB
	hm_0	35.5%	3,993,316	8.3 KB
	mds_0	11.9%	1,211,034	7.2 KB
	prxy_0	2.7%	12,518,968	2.5 KB
	rsrch_0	9.3%	14,333,655	8.7 KB
	wdev_0	20.1%	1,143,261	9.4 KB

both normalized to those of LGC, among the LGC, GGC and GC-Steering schemes, respectively.

First, GC-Steering outperforms LGC and GGC in terms of average response time by 63.3% and 65.8% on average, respectively. The significant performance improvement comes from the fact that an average of 85.5% user I/O requests during the GC period are redirected to the staging space. Therefore, the contention between user I/O requests and GC-induced requests is significantly alleviated. From the point of view of user applications, the long latency caused by GC operations is significantly reduced [38], thus significantly reducing the average response time. On the other hand, GC-Steering performs better under write-intensive workloads (*e.g.*, HPC_W) than under read-intensive workloads (*e.g.*, HPC_R), because GC operations in SSD-based RAID are much more frequent under the former than that under the latter. Since GC-Steering works to steer user I/O requests away from SSDs in the GC state to reduce the GC impact, the more frequent the GC operations are, the more positive performance impact GC-Steering will have. Furthermore, GC-Steering does not reduce nor increase the GC count, as shown in Figure 7(b), because GC-Steering merely steers away user I/O requests during GC period without changing when, how and whether GC happens. This feature of GC-Steering makes it orthogonal to and ready to be integrated with existing GC optimizing schemes to further improve the overall system performance.

Second, GGC outperforms LGC for the realistic enterprise-scale workloads, but the reverse is true for the HPC-like workloads. More specifically, LGC outperforms GGC in average response time by 24.9% and 36.1% for the HPC_W and HPC_R workloads, respectively. The reason is that the two HPC-like workloads have larger average request size and higher I/O intensity than the realistic enterprise-scale workloads, resulting in much higher GC frequency in the experiments driven by the former than in the latter. In GGC, once an SSD within a RAID initiates its GC process, all the other SSDs of the RAID must start their GC processes no matter how much free space is available in them. Therefore, the total GC count of GGC is much larger than that of LGC, as shown in Figure 7(b). Moreover, under the two HPC-like workloads the tail latency in the GGC scheme is much more serious than that in LGC scheme. The results are different from those in the original GGC study because we are using different platforms. We use RAID-5 as the baseline and use a real RAID-5 system with Intel SSDs in the experiments, instead of a RAID-0 with an SSD-extended DiskSim simulator used in the GGC study [17]. On the other hand, GGC outperforms LGC in terms of average response time in the experiments driven by the enterprise-scale workloads, albeit by only 6.7% on average. The reason is that even though GGC forces all SSDs to conduct GC operations simultaneously to alleviate the contention between user I/O requests and GC-induced I/O requests, the user access latency in GGC during the coordinated GC period, in which GCs in all SSDs are simultaneously activated, is much higher than that in LGC, which offsets some, but not all performance gains of GGC.

(2) Sensitivity study. The GC-Steering performance is likely influenced by several important factors, including the number of SSDs in a RAID, the stripe unit size and the design choice.

Number of SSDs. To examine the sensitivity of GC-Steering to the number of SSDs in a RAID, we conduct experiments on RAID5 systems consisting of different numbers of SSDs (5 and 7) with a stripe unit size of 64KB. Figure 8 shows the experimental results for GC-Steering, indicating that the average response time decreases with the number of SSDs in a RAID. The reason is that more SSDs in a RAID system imply higher parallelism for the I/O process, concurrently reducing the number of I/O requests and the total write data addressed to individual SSDs. Consequently, the GC count of each SSD is reduced accordingly, resulting in reduced average response time. More SSDs in a RAID imply not only a lower GC count, but also reductions of the I/O latency and queueing in each SSD, further reducing the average response time. On the other

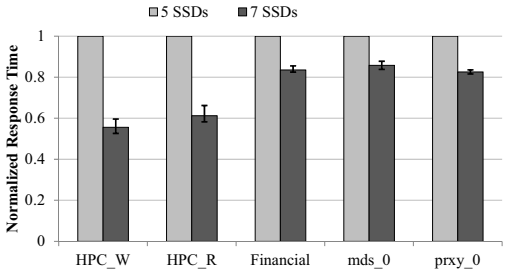


Fig. 8. Impact of the number of SSDs on the response time in GC-Steering.

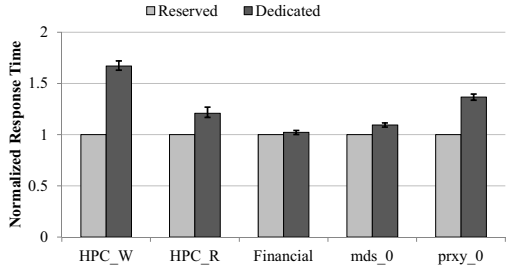


Fig. 10. Impact of the staging space on the response time in GC-Steering.

hand, the performance improvement of GC-Steering under the HPC-like workloads is much more significant than that under the realistic enterprise workloads, because the former are much more intensive than the latter.

Stripe unit size. To examine the impact of the stripe unit size of the RAID system in GC-Steering, we conduct experiments on a RAID5 system consisting of 5 SSDs with stripe unit sizes of 4KB, 64KB and 128KB, respectively. Figure 9 shows that no clear and consistent patterns seem to emerge about the relationship between the average response time and the stripe unit size in GC-Steering. Previous studies have revealed that optimal stripe unit size is highly depended on workload characteristics, *i.e.*, the access types (read or write) and request sizes, which is consistent with the results illustrated in Figure 9.

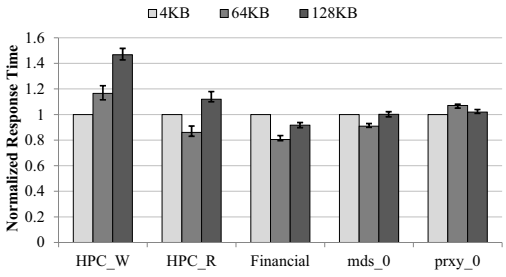


Fig. 9. Impact of the stripe unit size on the response time in GC-Steering.

Design choice of staging space. By default, we configure the pre-reserved space (short for Reserved) of each SSD within the SSD-based RAID as the staging space in GC-Steering. In our design, the staging space in GC-Steering can also be configured with a dedicated SSD (short for Dedicated). To examine the impact of the different types of staging space on the GC-Steering performance, we conduct experiments on a RAID5 system consisting of 5 SSDs with a stripe unit size of 64KB and configure the two mentioned types of staging space. Figure 10 shows that the average response time with the staging space configured by a dedicated SSD is longer than that with the staging space configured with the pre-reserved space of each SSD within RAID. The reasons are twofold. First, flash-based SSDs, consisting of concurrently accessible chips (*e.g.*, parallel channels) and without any mechanically moving parts like those in HDDs, offer much more parallelism than that with a dedicated SSD to service both the normal user I/O requests and the redirected user I/O requests, thus reducing the average response time. Second, a staging space configured with the pre-reserved space of each SSD has much more SSDs to service the normal user I/O requests during non-GC periods than that configured with a dedicated SSD. The dedicated SSD is idle when there are no GC operations in all SSDs within a RAID, but the pre-reserved space of each SSD is not.

C. Reconstruction performance

The other design goal of GC-Steering is to improve the RAID reconstruction efficiency. We conduct experiments on a RAID5 system consisting of 6 SSDs with a stripe unit size of 64KB driven by the different workloads. For LGC, GGC and GC-Steering (Dedicated), 5 SSDs service the user I/O requests and the remaining SSD acts as the replacement SSD for all three schemes and jointly used as the staging space for GC-Steering. Figure 11 compares the three schemes in terms of average response time during RAID reconstruction, normalized to the response time when no reconstruction is underway. The RAID reconstruction bandwidth is set to range between 1MB/s and 10MB/s. In the experiments, we find that the Linux MD software favors the reconstruction process but not user I/O requests, thus the reconstruction speed is always at the maximum of 10MB/s. Consequently, the reconstruction times for the three schemes are almost the same. However, Figure 11 shows that for the LGC and GGC schemes, the average response time during RAID reconstruction is increased by an average of 45.6% and 47.3% more than that in the normal state, respectively.

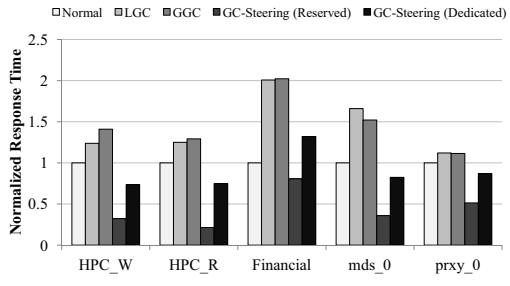


Fig. 11. Average response time during RAID reconstruction, normalized to the average response time when there is no reconstruction.

On the other hand, for GC-Steering with pre-reserved space of each SSD within a RAID (Reserved) and a dedicated SSD (Dedicated), the average response times are 55.7% and 10.1% less than that in the normal state, respectively. The large discrepancy in improvement between the two GC-Steering configurations comes from two factors. First, there are more SSDs to service user I/O requests during RAID reconstruction for Reserved GC-Steering than for Dedicated GC-Steering, thus reducing the number of user I/O requests addressed to each SSD within a RAID accordingly. Second, the reconstruction-induced write operations can be performed in parallel on surviving SSDs, thus alleviating the write bottleneck on the replacement SSD. Consequently, Reserved GC-Steering reduces the average response time significantly more than Dedicated GC-String during RAID reconstruction.

V. Related Work

Most of the existing studies on SSD-based RAIDs focus on the following two issues: (1) parity update problem and (2) GC-incurred performance degradation problem. Our work belongs to the latter category and focuses on improving the performance and reliability of SSD-based RAIDs.

Balakrishnan *et al.* [2] propose Diff-RAID to distribute the parity unevenly across the disk array and proactively replace the SSD degraded the fastest to improve the reliability of SSD-based RAID5. However, Diff-RAID does not reduce the number of parity updates on the SSDs and the performance degrades further due to the skewed parity updates. Flash-aware RAID [11] uses a delayed parity update strategy and a partial parity technique to reduce the number of internal write operations. The elastic striping and anywhere parity scheme [16] reconstructs new stripes with updated data chunks without updating the old parity chunks to reduce the parity update operations. Based on the elastic striping scheme, Pan *et al.* [25] propose a grouping-based elastic striping scheme to separately write data chunks in different groups into SSDs by exploiting the workload characteristics. Logging is an effective technique to transform small random writes into large writes and widely studied for SSD-based disk arrays, such as HPDA [22], LDM [37], and EPLog [20]. However, all the these studies do not consider the GC activities of SSDs in SSD-based RAIDs.

GCaR [36] scheme changes the cache replacement policy in FTL to be aware of the underline GC activities. Tiny-Tail Flash [8] uses GC-tolerant read and GC-tolerant flush in FTL to alleviate the GC-induced performance degradation. However, the GC-tolerant read introduces many more read requests which increase the read/write interactions and queue length within an SSD [30], [33]. Kim *et al.* [17] find that the uncoordinated GC operations on individual SSDs amplify the performance degradation of SSD-based RAIDs and propose a RAID-level Global Garbage Collection (GGC) mechanism to reduce the performance variability for SSD-based RAIDs. However, GGC forces all SSDs in an SSD-based RAID to process the GC operations at the same time, rendering the SSD-based RAID unavailable to service the applications during the coordinated GC period. Inspired by the study of I/O Workload Outsourcing [34] that optimizes the reconstruction performance for HDD-based RAIDs, GC-Steering effectively exploits the workload characteristics and the reserved space of each SSD within SSD-based RAIDs, to alleviate the negative performance and reliability impact of GC operations on SSD-based RAIDs. Importantly, GC-Steering dose not block the applications at any time, which is much more acceptable to end users than GGC. Moreover, our evaluation platform and configurations, with real SSDs and RAID-5, are quite different from those of the GGC paper (SSD-extended DiskSim and RAID-0). Table II summarizes studies most closely related to GC-Steering. Different from the existing studies, GC-Steering does not control the GC workflow within SSDs and considers the RAID reconstruction to improve both performance and reliability of SSD-based RAID5/6.

TABLE II
A comparison of the related studies to GC-Steering.

Schemes	*Environment*	*Rebuild*	*Control GC?*
GCaR [36]	FTL within an SSD	No	No
TTFlash [8]			Rotated one by one
GGC [17]	SSD-based RAID0		Concurrently
GC-Steering	SSD-based RAID5/6	Yes	No

On the other hand, to the best of our knowledge, GC-Steering is the first study to consider the failure-recovery for SSD-based RAIDs in the literatures. HDD-based RAID reconstruction has been studied extensively in the literature [34], [35]. However, the previous studies on HDD-based RAID reconstruction try to either make the reconstruction I/O requests sequential on HDDs [10] or alleviate the interference between user I/O requests and reconstruction I/O requests [34]. The evaluations of GC-Steering demonstrate that alleviating the interference between user I/O requests and reconstruction I/O requests can also improve the RAID reconstruction efficiency for SSD-based RAIDs. Moreover, our study reveals that the parallel reconstruction by exploiting the high-random-performance of SSDs is highly beneficial for SSD-based RAIDs. GC-Steering is consistent with the existing performance optimizations on SSD-based storage systems by randomizing the I/O requests [15]. Our current study represents but a first phase of substantial work required to investigate efficient reconstruction algorithms for SSD-based RAIDs.

VI. Conclusion

With the rapid development and wide deployment of flash-based SSDs, SSD-based RAIDs have become one of the most effective ways to provide high-performance and highly-reliable storage systems. Unlike HDDs with mechanical moving parts, the inherent GC operations in SSDs can seriously affect the performance and reliability of SSD-based RAIDs. Consequently, Straightforwardly applying the RAID algorithm to SSDs, without fully considering the unique characteristics of flash-based SSDs, can lead SSD-based RAIDs to fall significantly short of the performance and reliability promised

by the SSD technology. The GC operations being performed on individual SSDs can cause severe performance variability and tail latency in SSD-based RAIDs. Moreover, the traditional RAID reconstruction algorithms are not suitable for SSD-based RAIDs. To address both the performance and reliability problems, in this paper, we propose the GC-Steering scheme to alleviate the GC impact on the performance and reliability of SSD-based RAIDs by exploiting the flash device and workload characteristics to alleviate the contention between user I/O requests and GC-induced internal requests. The extensive evaluation on a lightweight prototype of GC-Steering shows that GC-Steering outperforms the state-of-the-art schemes LGC and GGC by 63.3% and 65.8% on average in terms of average response time, respectively. Moreover, GC-Steering also significantly reduces the user response time during RAID reconstruction by an average of 55.7% when driven by HPC-like workloads and realistic enterprise workloads.

GC-Steering is an ongoing research project and we are currently exploring several directions for the future work. First, we will investigate how to utilize GC-Steering to alleviate the parity update problem of SSD-based RAIDs by using more applications to evaluate the effectiveness of GC-Steering. Second, we will investigate how GC-Steering affects the performance and reliability simultaneously for other RAID levels, such as RAID1 and RAID6.

ACKNOWLEDGEMENT

This work is supported by the National Natural Science Foundation of China under Grant No. 61772439, No. U1705261, No. 61472336, and No. 61402385, the US NSF under Grant No. CCF-1704504 and CCF-1629625.

REFERENCES

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy. Design Tradeoffs for SSD Performance. In *USENIX'08*, Jun. 2008.

[2] M. Balakrishnan, A. Kadav, V. Prabhakaran, and Dahlia Malkhi. Differential RAID: Rethinking RAID for SSD Reliability. In *EuroSys'10*, Apr. 2010.

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

[4] H. Breih. How Flash Memory Will Affect Tomorrow's Automobiles. In *Flash Memory Submit*, Aug. 2017.

[5] F. Chen, D. Koufaty, and X. Zhang. Understanding Intrinsic Characteristics and System Implications of Flash Memory based Solid State Drives. In *SIGMETRICS/Performance'09*, Jun. 2009.

[6] J. Colgrove, J. Davis, J. Hayes, E. Miller, C. Sandvig, R. Sears, A. Tamches, N. Vachharajani, and F. Wang. Purity: Building Fast, Highly-Available Enterprise Flash Storage from Commodity Components. In *SIGMOD'15*, Jun. 2015.

[7] J. Dean and L. Barroso. The Tail at Scale. *Communications of the ACM*, 56(2):74–80, 2013.

[8] M. Hao, G. Soundararajan, D. Kenchammana-Hosekote, A. Chien, and H. Gunawi. The Tail at Store: A Revelation from Millions of Hours of Disk and SSD Deployments. In *FAST'16*, Feb. 2016.

[9] J. Hennessy and D. Patterson. Towards Availability Benchmarks: A Case Study of Software RAID Systems. *Computer Architecture: A Quantitative Approach, Fourth edition*, 2006.

[10] M. Holland. On-Line Data Reconstruction in Redundant Disk Arrays. *Carnegie Mellon Ph.D. Dissertation CMU-CS-94-164*, Apr. 1994.

[11] S. Im and D. Shin. Flash-Aware RAID Techniques for Dependable and High-Performance Flash Memory SSD. *IEEE Transcations on Computers*, 60(61):80–92, 2011.

[12] N. Jeremic, G. Mühl, A. Busse, and J. Richling. The pitfalls of deploying solid-state drive RAIDs. In *SYSTOR'11*, May. 2011.

[13] M. Jung, W. Choi, J. Shalf, and M. Kandemir. Triple-A: A Non-SSD Based Autonomic All-Flash Array for Scalable High Performance Computing Storage Systems. In *ASPLOS'14*, Apr. 2014.

[14] M. Jung, R. Prabhakar, and M. Kandemir. Taking Garbage Collection Overheads Off the Critical Path in SSDs. In *Middleware'12*, Dec. 2012.

[15] H. Kim, D. Shin, Y. Jeong, and K. Kim. SHRD: Improving Spatial Locality in Flash Storage Accesses by Sequentializing in Host and Randomizing in Device. In *FAST'17*, Feb. 2017.

[16] J. Kim, J. Lee, J. Choi, D. Lee, and S. Noh. Improving SSD Reliability with RAID via Elastic Striping and Anywhere Parity. In *DSN'13*, Jun. 2013.

[17] Y. Kim, S. Oral, G. Shipman, J. Lee, D. Dillow, and F. Wang. Harmonia: A Globally Coordinated Garbage Collector for Arrays of Solid-state Drives. In *MSST'11*, May 2011.

[18] J. Lee, Y. Kim, G. Shipman, S. Oral, F. Wang, and J. Kim. A Semi-Preemptive Garbage Collector for Solid State Drives. In *ISPASS'11*, Apr. 2011.

[19] Q. Li, L. Shi, C. Xue, K. Wu, C. Ji, Q. Zhuge, and E. Sha. Access Characteristic Guided Read and Write Cost Regulation for Performance Improvement on Flash Memory. In *FAST'16*, Feb. 2016.

[20] Y. Li, H. Chan, P. Lee, and Y. Xu. Elastic Parity Logging for SSD RAID Arrays. In *DSN'16*, Jun. 2016.

[21] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu. WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management. In *MSST'15*, Santa Clara, CA, Jun. 2015.

[22] B. Mao, H. Jiang, S. Wu, L. Tian, D. Feng, J. Chen, and L. Zeng. HPDA: A Hybrid Parity-based Disk Array for Enhanced Perfromance and Reliability. *ACM Transactions on Storage*, 8(1):Artical 4, 2012.

[23] J. Meza, Q. Wu, S. Kumar, and O. Mutlu. A Large-Scale Study of Flash Memory Failures in the Field. In *SIGMETRICS'15*, Jun. 2015.

[24] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating Server Storage to SSDs: Analysis of Tradeoffs. In *EuroSys'09*, Mar. 2009.

[25] Y. Pan, Y. Li, Y. Xu, and Z. Li. Grouping-based Elastic Striping with Hotness Awareness for Improving SSD RAID Performance. In *DSN'15*, Jun. 2015.

[26] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *SIGMOD'88*, Jun. 1988.

[27] Samsung defends flash drive reliability. http://news.cnet.com/8301-13924_3-9876557-64.html.

[28] B. Schroeder, R. Lagisetty, and A. Merchant. Flash Reliability in Production: The Expected and the Unexpected. In *FAST'16*, Feb. 2016.

[29] N. Shahidi, M. Arjomand, M. Jung, M. Kandemir, C. Das, and A. Sivasubramaniam. Exploring the Potentials of Parallel Garbage Collection in SSDs for Enterprise Storage Systems. In *SC'16*, Nov. 2016.

[30] D. Skourtis, D. Achlioptas, N. Watkins, C. Maltzahn, and S. Brandt. Flash on Rails: Consistent Flash Performance through Redundancy. In *USENIX ATC'14*, Jun. 2014.

[31] SNIA SSS PTS. http://www.snia.org/forums/sssi/ptstest.

[32] H. Sun, X. Qin, H. Jiang, J. Huang, and C. Xie. RB-Explorer: An Accurate and Practical Approach to Write Amplification Measurement for SSDs. *IEEE Transactions on Computers*, 64(4):1133–1148, 2015.

[33] S. Wells. Avoiding Costly Read Latency Variations in SSDs Through I/O Determinism. In *Flash Memory Submit*, Aug. 2017.

[34] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao. WorkOut: I/O Workload Outsourcing for Boosting the RAID Reconstruction Performance. In *FAST'09*, Feb. 2009.

[35] S. Wu, H. Jiang, and B. Mao. Proactive Data Migration for Improved Storage Availability in Large-Scale Data Centers. *IEEE Transactions on Computers*, 64(9):2637–2651, 2015.

[36] S. Wu, Y. Lin, B. Mao, and H. Jiang. GCaR: Garbage Collection aware Cache Management with Improved Performance for Flash-based SSDs. In *ICS'16*, Jun. 2016.

[37] S. Wu, B. Mao, X. Chen, and H. Jiang. LDM: Log Disk Mirroring with Improved Performance and Reliability for SSD-based Disk Arrays. *ACM Transactions on Storage*, 12(4):1–22, 2016.

[38] S. Yan, H. Li, M. Hao, H. Tong, S. Sundararaman, A. Chien, and H. Gunawi. Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs. In *FAST'17*, Feb. 2017.