GC-Steering: GC-Aware Request Steering and Parallel Reconstruction Optimizations for SSD-Based RAIDs

Suzhen Wu, *Member, IEEE*, Weidong Zhu, *Student Member, IEEE*, Yingxin Han, Hong Jiang, *Fellow, IEEE*, Bo Mao, *Member, IEEE*, Zhijie Huang, *Member, IEEE*, and Liang Chen

***Abstract*—Solid-state disk (SSD)-based redundant array of independent disks (RAIDs) have been widely deployed in high-end enterprize 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 introduce 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 article, we proposed GC-aware request steering (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 enterprize 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 local GC and global GC schemes. Moreover, the GC-Steering scheme also significantly reduces the average response times by an average of 62.3% during RAID reconstruction than the normal state.**

***Index Terms*—Garbage collection (GC), parallel reconstruction, request steering, solid-state disk (SSD)-based redundant array of independent disks (RAIDs).**

Manuscript received May 22, 2019; revised July 29, 2019, October 13, 2019, and December 13, 2019; accepted February 7, 2020. Date of publication February 17, 2020; date of current version November 20, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant U1705261, Grant 61972325, Grant 61872305, and Grant 61702569, in part by U.S. NSF under Grant CCF-1704504 and Grant CCF-1629625, and in part by the Fuzhou Science and Technology Project under Grant 2019-G-55. This article was recommended by Associate Editor C. Yang. *(Corresponding author: Bo Mao.)*

Suzhen Wu, Weidong Zhu, and Yingxin Han are with the Computer Science Department, Xiamen University, Xiamen 361005, China (e-mail: suzhen@xmu.edu.cn).

Hong Jiang and Zhijie Huang are with the Computer Science and Engineering Department, University of Texas at Arlington, Arlington, TX 76019 USA (e-mail: hong.jiang@uta.edu).

Bo Mao is with the Software School, Xiamen University, Xiamen 361005, China (e-mail: maobo@xmu.edu.cn).

Liang Chen is with the College of Engineering, Fuzhou Institute of Technology, Fujian 350506, China.

Digital Object Identifier 10.1109/TCAD.2020.2974346

I. Introduction

According to IDC's latest report on the enterprize storage market, the flash-based storage category generated $2.1 billion in sales during the first quarter of 2018 which is a year-over-year increase of nearly 55 percent, outpacing the overall enterprize storage market [1]. Flash-based solid-state disks (SSDs) have emerged as alternatives to hard disk drives (HDDs), increasingly replacing or coexisting with HDDs in desktops, enterprize storage systems, and large-scale data centers [2]–[4]. Flash-based SSDs are made of semiconductor chips which offer many benefits, such as low-power consumption, high robustness to vibrations and temperature, and most importantly, high small-random-read performance. However, SSDs also have some disadvantages compared with HDDs, such as high cost per GB, the garbage collection (GC) induced performance degradation and the limited number of program/erase (P/E) cycles [5].

Besides the read and write operations that are common in both HDDs and SSDs, GC is a key operation which is time-consuming within flash-based SSDs. Generally speaking, the granularity of a read or write operation is a page (2 KB–4 KB), and an erase operation, which generally writes all 1 s, is a block (128 KB–256 KB) consisting of multiple pages (e.g., 64 or 128). The processing time of an erase operation is an order of magnitude more than that of a read or write operation [5], [6]. On the other hand, the flash translation layer (FTL) within SSDs performs out-of-place writes instead of in-place write that HDDs use, which results in a large number of invalid pages within flash blocks. To free up more flash space for subsequent write data, these blocks are selected to be erased by the GC operation. It first copies all the valid pages in a victim block into a free block and then erases the victim block, which is an obviously time-consuming process. The user I/O performance is significantly affected by the GC operations due to the severe contention between the external user I/O requests and the internal GC-induced requests [7]–[11]. Moreover, the internal GC-induced performance degradation is also the primary cause of slowdowns for flash-based storage systems, which can cause significant tail latency [12], [13].

Along with the increasing requirements of the performance, capacity, and reliability of the high-performance computing and enterprize storage systems, applying the redundant array of independent disks (RAIDs) [14] algorithm to SSD-based disk arrays is promising and feasible [15], [16]. Though the disks within a disk array are independent, the performance is restricted by the slowest device in the array. Therefore, the intermittent GC operations of the individual SSDs within an SSD-based RAID will cause serious performance variability and degradation [12], [17], [18]. Kim *et al.* [19] found that the uncoordinated GC operations of the individual SSDs significantly degraded the performance of SSD-based RAIDs.

Moreover, recent studies have revealed that flash-based disks also show significant probability of failures in large-scale data centers. The analysis on the data collected over a period of nearly four years from a majority of flash-based SSDs at Facebook data centers reveals that SSD failures are relatively common events with 4.2%–34.1% of SSDs reporting uncorrectable errors [20]. Another study on flash reliability based on the data collected over a six-year period on SSDs used in Google data centers finds that 1%–2% of flash-based SSDs are replaced annually due to the suspected hardware problems over the first four years in the production [21]. These failure studies also imply that it is important and urgent to investigate and improve the failure-recovery process for SSD-based RAIDs.

To address both the performance and reliability issues of SSD-based RAIDs alluded to above, we propose GC-aware request steering (GC-Steering) scheme that is aware of the GC process within an SSD-based RAID. The main idea behind GC-Steering is to fully exploit the workload characteristics and utilize the prereserved 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/recovery operations on the system performance and reliability. GC-Steering proactively migrates "hot" read data to the staging space, thus the subsequent read requests addressed to an SSD currently in the GC state can be alternatively serviced by the staging space without being interfered with the ongoing GC process. For the incoming write requests addressed to an SSD currently in the GC state, GC-Steering temporally stores them to the staging space. As a result, the contention between the external user I/O requests and the internal GC/recovery-induced I/O requests is significantly alleviated, if not completely eliminated. Moreover, the recovery performance is also significantly accelerated because: 1) the contention between user I/O requests and reconstruction-induced I/O requests is alleviated by temporarily redirecting all write requests and popular read requests originally targeting at the degraded SSD-based RAID to the staging space and 2) the parallel access characteristics of flash-based SSDs are exploited during the failure-recovery process.

We implement a lightweight prototype of GC-Steering by integrating it into the Linux software RAID module. To assess the effectiveness of the GC-Steering scheme, we conduct extensive trace-driven experiments driven by a wide range of HPC-like and realistic enterprize workloads. The evaluation results show that the GC-Steering scheme reduces the average response times than the local GC (LGC) and global GC (GGC) schemes by an average of 63.3% and 65.8%, respectively. Moreover, the GC-Steering scheme also significantly reduces the average response times by an average of 62.3% during the failure recovery period than that during the normal state.

In summary, this article makes the following contributions.

1) By fully exploits the workload characteristics and utilizes the staging space in SSD-based RAID, GC-Steering alleviates the negative impact of the GC operations on the performance and reliability of SSD-based RAIDs.
2) To the best of our knowledge, GC-Steering is the first study on fully understanding and improving the failure-recovery process in SSD-based RAIDs. It improves the reconstruction efficiency by the proposed request steering and parallel recovery techniques.
3) We conduct extensive experiments on a lightweight GC-Steering prototype and the evaluation results show that GC-Steering significantly reduces the average response times in both normal operational period and reconstruction period, thus improving both the performance and reliability of SSD-based RAIDs.

The remainder of this article 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 article and point out the directions for future research in Section VI.

II. Background and Motivation

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

A. Adverse Impact of GC on SSD-Based Storage

Due to the unique physical features of NAND flash, read, and write operations are performed on a minimal unit of page while erase operations are performed on a unit of block. Especially, write requests are serviced out-of-place rather than in-place since data can only be written to erased pages (*also known as*, 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 GC process that significantly affects the user I/O performance of SSD-based storage systems [7], [13], [22], [23]. Fig. 1(a) shows the microscopic analysis of the average response times driven by the Financial workload on a single SSD. The SSD is filled up before the evaluations and we can see that larger latencies are occurred due to the frequent GC operations.

On the other hand, due to the limited number of erase cycles of each memory cell in a block, GC also significantly affects the reliability (endurance) of flash-based SSDs.

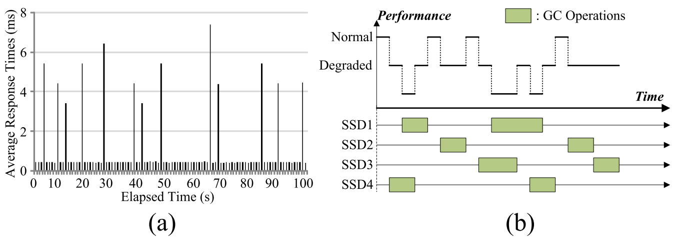


Fig. 1. Example of the performance impact of GC operations on a (a) single SSD and (b) SSD-based RAID5.

Generally, a single-level-cell (SLC) memory cell can withstand up to 100 000 write cycles (or erase cycles) before failing while a 2-bit multilevel-cell (MLC) memory cell can typically withstand up to 10 000 write cycles before failing [5]. What is worse, a 3-bit triple-level-cell (TLC) memory cell can only sustain about 1000 write cycles before failing, meaning that the write endurance of TLC flash is much lower than that of SLC/MLC flash chips. Therefore, how to address the GC-induced performance and reliability problems has become a critically important challenge when deploying flash-based SSDs into HPC and enterprize storage systems.

This negative GC impact on an SSD-based RAID is arguably much more significant than on an individual SSD [24]–[26]. For example, the frequency of GC operations in an SSD-based RAID consisting of multiple SSDs, say N, is at least N times higher than that in the latter. Consequently, user I/O requests have a much higher probability to be blocked or interfered with GC-induced internal I/O requests. Previous study by Kim *et al.* [19] has revealed that the uncoordinated LGC process on individual SSDs in an SSD-based RAID degrades the whole system performance and causes a serious performance variability of SSD-based RAIDs. Fig. 1(b) shows an example which depicts how the default LGC scheme affects the performance of SSD-based RAIDs. In an extreme case, such as one illustrated in the figure, the uncoordinated but consecutive occurrences of the GC processes among the individual SSDs 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 directly affecting the system availability [27]. Moreover, the GC-induced performance degradation also causes significant tail latency problem [28]–[30].

The first solution to address the problem is GGC proposed by Kim *et al.* [19] which forces all the SSDs in an SSD-based RAID to initiate the GC operations at the same time. However, GGC makes the SSD-based RAID unavailable for the incoming user I/O requests during the GC period. The reason is that all the SSDs in an SSD-based RAID are busy with dealing with the internal 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 also lead to SLA and SLO violations for many applications, particularly those required to be available 24/7. Thus, it is desirable to design a new SSD-based RAID scheme to be available to service the user I/O requests all the time. The key challenge is to reduce the interference between the external user I/O requests and the internal GC/recovery-induced internal I/O requests.

B. Failure-Recovery for SSD-Based RAIDs

Failure recovery is critically important for RAID in case of a disk failure. Existing failure recovery algorithms are mainly designed for HDD-based RAIDs. For HDDs, sequential read/write operations are much faster than their random counterparts. Thus, existing failure-recovery algorithms try to make the reconstruction I/O requests sequential on HDDs within HDD-based RAIDs. However, different from HDDs, flash-based SSDs are made of semiconductor chips and have abundant internal access parallelism [31]. By exploiting the internal parallelism of flash-based SSDs, random accesses can be performed much more efficiently on SSDs than on HDDs [32], [33]. Moreover, flash-based SSDs have the read-write performance asymmetry, in which the read performance is much higher (about 10×) than the write performance. It implies 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 active 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 [34]. The SSD controller failure makes the whole SSD unusable and thus triggering the RAID reconstruction process immediately within SSD-based RAIDs. Some studies on the data collected from a large number 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 and 1%–2% of SSDs being replaced annually due to suspected hardware problems [20], [21]. These studies and findings make it abundantly clear why it is critically important and urgent to accelerate the failure-recovery process for SSD-based RAIDs.

C. Trace Characteristics and Motivation

It is important to understand the workload characteristics for the storage system design, particularly for flash-based storage systems where the unique features of flash memory and their interactions with workloads accentuate this importance [35]. For this purpose, we categorize and differentiate flash pages into the following three distinct types based on how data stored in a page is accessed by the applications.

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

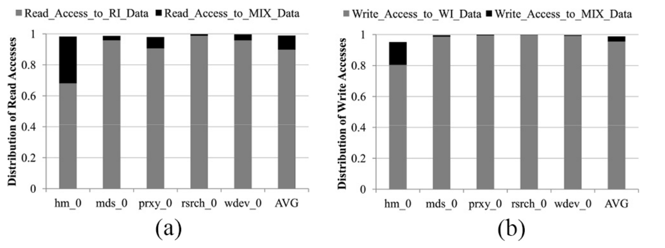


Fig. 2. Distribution of (a) read and (b) write requests on the three types of data pages in MSR traces.

TABLE I
WORKLOAD CHARACTERISTICS

Type	**Traces**	**Trace Characteristics**		
		Read Ratio	IOPS	Avg. Req. Size
HPC-like	HPC_W	20.1%	477	510.5 KB
	HPC_R	79.9%	477	510.5 KB
OLTP	Fin1	32.8%	52	11.9 KB
MSR enterprise	hm_0	35.5%	134	8.3 KB
	mds_0	11.9%	201	7.2 KB
	prxy_0	2.7%	86	2.5 KB
	rsrch_0	9.3%	173	8.7 KB
	wdev_0	20.1%	189	9.4 KB

Fig. 2 shows the access patterns on these three types of data pages in Microsoft Research Cambridge (MSR) traces as that shown in Table I. Fig. 2(a) illustrates the distribution of read requests among the RI data pages and MIX data pages and Fig. 2(b) shows the distribution of write requests among WI data and MIX data pages. The bars are not 100% means that some write requests may update the RI data pages (Write_Access_to_RI_Data) and some read requests may access the WI data pages (Read_Access_to_WI_Data), due to the RI and WI data are defined to be 90%, not 100%. We can see that an average of 89.8% read requests access the RI data pages and an average of 95.5% write requests access the WI data pages. Only a small portion of the requests access the MIX data pages. These observations are also consistent with those reported in the previous studies [36]–[38]. More importantly, these findings on the workload characteristics imply that the hot read data blocks are not frequently updated by write requests.

Based on the above analysis and device characteristics of flash-based SSDs and SSD-based RAIDs in HPC and enterprize environments, it is clearly that the design of SSD-based RAIDs is fundamentally different from that of HDD-based RAIDs to significantly improve performance and reliability of SSD-based RAIDs. This, combined with awareness and understanding of the workload characteristics, motivates design of GC-Steering that proactively migrates the hot data blocks within an SSD-based RAID to a prereserved 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/recovery-induced I/O requests. Moreover, the internal parallelism of flash-based SSDs should be fully exploited to further accelerate the failure recovery process. This helps improve the user I/O performance and RAID failure-recovery efficiency simultaneously.

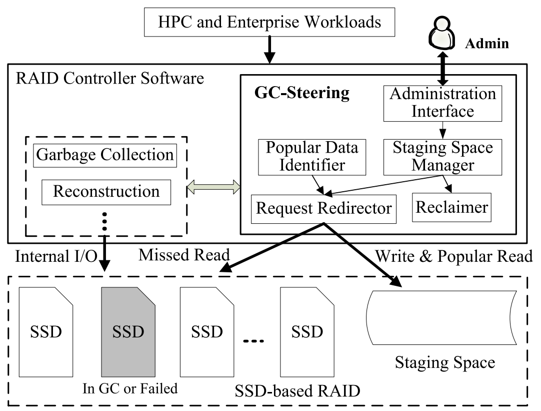


Fig. 3. System overview of GC-Steering within the RAID controller software for SSD-based RAIDs.

III. DESIGN OF GC-STEERING

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

A. System Overview

The design objectives of GC-Steering are to improving both the performance and reliability of SSD-based RAIDs by addressing GC/recovery-induced interference. To achieve the goals, 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/recovery-induced I/O requests. Furthermore, by redirecting many I/O requests away from the degraded SSD-based RAID to the staging space during RAID reconstruction and exploiting the parallelism within flash-based SSDs, both the user performance degradation and reconstruction performance degradation caused by the failure-recovery process can be alleviated. Currently, GC-Steering is based on SSD-based RAID5 organization.

Fig. 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, Fig. 3 illustrates how GC-Steering is augmented to the RAID controller software with five key functional components: 1) administration interface; 2) popular data identifier; 3) staging space manager; 4) request redirector; and 5) reclaimer. *Administration Interface* provides an interface for system administrators to configure the GC-Steering design options. *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

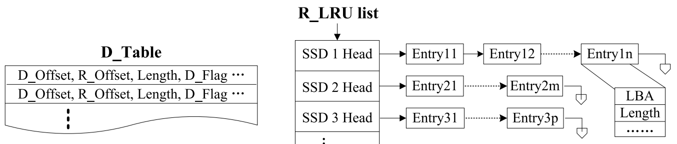


Fig. 4. Data structures of GC-Steering.

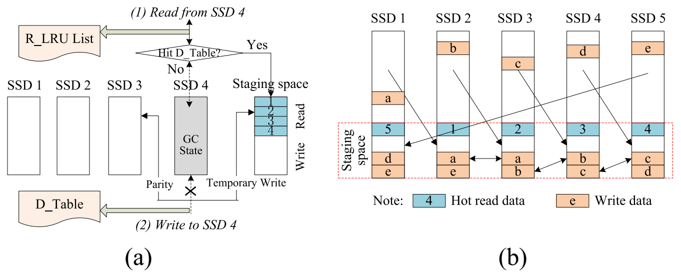


Fig. 5. I/O processing workflow and the data layout in GC-Steering. (a) I/O processing workflow. (b) Data layout in GC-Steering if the staging space is the reserved space of each SSD within RAID.

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.

To provide a flexible design space, the staging space can be different persistent configurations of SSD-based storage devices. For example, the persistent configurations can be a dedicated SSD or the reserved space of each SSD within an SSD-based RAID. Moreover, in order to recover data on the failed disk to the staging space during RAID reconstruction, the size of the staging space must be the same or larger than that of an SSD within the RAID. The staging space also can be shared by multiple RAIDs within the data center. 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 and is deactivated during the normal period for SSD-based RAIDs.

B. Data Structures

Two important data structures are designed in GC-Steering to record the redirected data blocks and identify popular read data, namely, *D_Table* and *R_LRU*, as shown in Fig. 4. The first data structure is D_Table which is a log table to manage the redirected data in the staging space. It contains the logs of all redirected data blocks, including the following important variables.

1) *D_Offset* and *R_Offset* indicate the offsets of the redirected data block in the SSD-based RAID and the staging space, respectively.
2) *Length* indicates the length of the redirected data block.
3) *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*).

The second data structure is R_LRU which is an LRU-style list to identify the popular read data blocks on each SSD. It 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. GC-Aware Request Processing Workflow

GC operations within SSD-based RAIDs conflict with the user requests directly and degrade the system performance. 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. Fig. 5(a) 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 serviced by the corresponding SSDs within the SSD-based RAID. R_LRU is updated accordingly to record the popular read data.

Fig. 5(a) also shows how GC-Steering deals with the incoming write requests. For the write requests, if they are 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. 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 the SSD-based 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. The entry in the D_Table will be deleted upon the write data is reclaimed back to the SSD-based RAIDs.

The data distribution is different in case of different staging space configurations. If the staging space is configured as a dedicated SSD, the data layout is the same as that shown in

Fig. 5(a). If the staging space is the reserved space in each SSD within the SSD-based RAID, the data layout is quite different. Fig. 5(b) shows how the read data and write data are stored in the reserved space of each SSD within RAID. First, the hot read data is stored in an interleaved fashion and organized in an RAID0-style for performance. 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 an 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.

Upon the completion of the GC operations within SSDs, 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. Moreover, once the write data is reclaimed from the staging space to the original RAID, the whole flash block is erased. Thus, the GC management in the staging space is simple and efficient without data coping overhead.

D. Parallel Reconstruction Workflow

When an SSD fails within SSD-based RAIDs, the RAID reconstruction process is immediately initiated. The replacement SSD can be a newly added SSD (indicated by ①) or a staging space (indicated by ②), as shown in Fig. 6(a). 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 hits 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

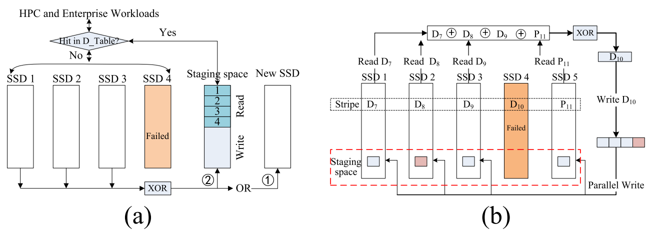


Fig. 6. (a) Reconstruction workflow and (b) parallel reconstruction enhancement in GC-Steering when using the reserved space on each SSD within RAID as the replacement SSD.

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 possible configurations, i.e., a dedicated SSD or the reserved space on each SSD within the SSD-based RAID. If the staging space is a dedicated SSD, the degraded RAID performs the traditional RAID reconstruction workflow. Because SSDs have asymmetric read and write performance characteristics, with the read operation being much faster than the write operation, the writing of the reconstructed data to the newly added SSD can become a performance bottleneck in traditional RAID reconstruction for SSD-based RAIDs.

If the staging space is the reserved space on each SSD within SSD-based RAID, the parallel reconstruction workflow is performed by the degraded RAID, as shown in Fig. 6(b). In this case, the reconstructed data of the failed SSD is written in parallel to the staging space. Although the read areas and the write areas within each SSD are 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 [32], [33]. Moreover, it must be noted that the staging space is organized in a RAID5-style for redundancy with a smaller stripe unit size after the previously redirected data stored in it is reclaimed back. The stripe unit size of the staging space is defined as the original stripe unit size divided by (N-2), where N is the number of SSDs in an SSD-based RAID. In this way, the reconstructed data block can be written in parallel to the staging space, as that illustrated in Fig. 6(b). As a result, the performance bottleneck due to writing the reconstructed data can be eliminated for SSD-based RAIDs with the reserved-space design. If a second SSD fails, the data in the staging space of the failed SSD is first reconstructed to the 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. In case of a power failure, the data stored in the staging space is not lost due to the persistency characteristics of flash storage. Upon the power is recovered, the data stored in the staging space is still accessible.

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 nonvolatile 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 the GC-Steering scheme by comparing it against LGC and GGC schemes through extensive trace-driven evaluations.

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 16-GB memory. In this system, a SAMSUNG HE253GJ SATA HDD (250 GB) 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 120-GB SSDs.

In the evaluation, we compare the performance of GC-Steering with two state-of-the-art schemes, LGC and GGC [19], in terms of average response time and tail latency. The GC activities in SSDs are detected with the read profiling technique by issuing read requests and monitoring the read response [12], [31]. If these read requests are not returned within a timeout window, GC-Steering marks the SSD in the GC state. Retry operations are also performed to check whether the SSD is still in the GC state. Moreover, with the new LightNVM module in the Linux kernel and the open channel SSDs, the GC operations can be tracked and controlled by the host easily [39]. In practice, the RAID controller, such as the ones in pure flash array products from Pure Storage [40] or HP Nimble Storage [41], can track the GC activities and manage the data layout among flash devices directly. Thus, GC-Steering can be easily embedded into these flash array products to further improve system performance and reliability.

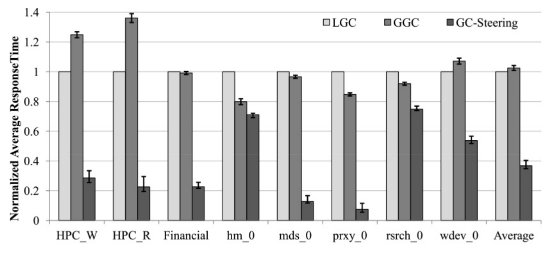
In the experiments, all the RAID schemes use the same number of SSDs to provide a fair comparison. To ensure that SSDs reach the steady state (i.e., with regular GCs) when new write requests arrive during the experimental running period, we fill the entire space on each SSD with valid data prior to measuring the performance, a common practice called simulation "warm-up" [19]. In the GGC scheme, when any one SSD within an 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 prereserved space of each SSD within the SSD-based RAID.

We use a mixture of HPC-like workloads and realistic enterprize-scale workloads to study the performance of our proposed the 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 enterprize-scale workloads, the Fin1 trace is collected from the OLTP applications running at a large financial institution [42]. The other traces are collected from storage volumes in an enterprize data center by MSR [43]. Since most MSR traces are one-day workloads with bursty and idle periods, we only choose 1-h traces with bursty periods to replay. These traces represent different access patterns in terms of read/write ratio, IOPS, and average request size, as summarized in Table I.

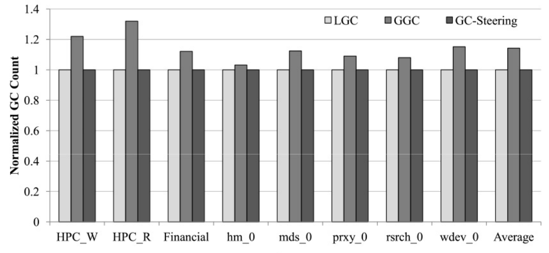
B. *Performance in the Normal State*

1) Results and Analysis: The first design goal of GC-Steering is to address the GC-induced performance degradation. We first conduct experiments on an RAID5 system consisting of five SSDs with a stripe unit size of 64 KB, driven by the different workloads in the normal operational state. Fig. 7(a) and (b) shows the comparisons of average response times and GC counts, 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,



(a)



(b)

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

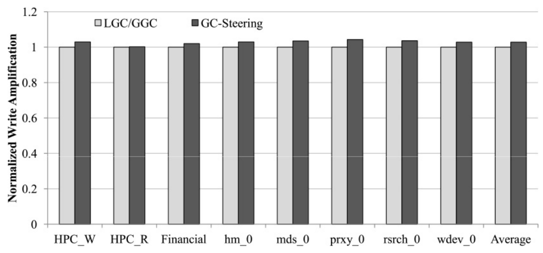


Fig. 8. Write amplification within GC-Steering due to the extra write requests to the staging space.

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 [13], 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 Fig. 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 enterprize-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 enterprize-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 an 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 Fig. 7(b). Moreover, under the two HPC-like workloads the tail latency in the GGC scheme is much more serious than that in the 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 [19]. On the other hand, GGC outperforms LGC in terms of average response time in the experiments driven by the enterprize-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.

During the GC period, GC-Steering temporally redirects write data and popular read data to the staging space which will incur the write amplification problem. Here, the write amplification is defined as the write requests performed in the block level, including these write requests issued to the staging space by GC-Steering, divide the user write requests. In order to investigate the write amplification problem, we intercept the write requests issued in the block device level to calculate the write amplification outside of SSDs. Fig. 8 shows the write amplification results. We can see that GC-Steering only increases write requests by an average of 2.8%, up to 4.3% for the write-intensive workload. The reason is that GC-Steering only redirects write data and popular read data during GC periods. In the normal state, GC-Steering performs the same as the LGC and GGC schemes. Thus, the write amplification problem associated with GC-Steering is negligible.

To investigate the impact of the three schemes on tail latency under the different workloads, we plot the I/O latencies at the 95th, 98th, 99th, and 99.9th percentiles, and the cumulative distributions of the request-response times in Figs. 9 and 10, respectively. First, GC-Steering consistently and significantly outperforms the other two schemes in the tail latency performance. While tail latency is mainly caused by the GC operations of SSDs within the RAID set, the GC-induced performance degradations are alleviated by the request redirections in the GC-Steering scheme. As a result, the percentage of requests with long latency is reduced accordantly. Second, GGC has a much more serious tail latency problem than the other two schemes under the HPC-like workloads and

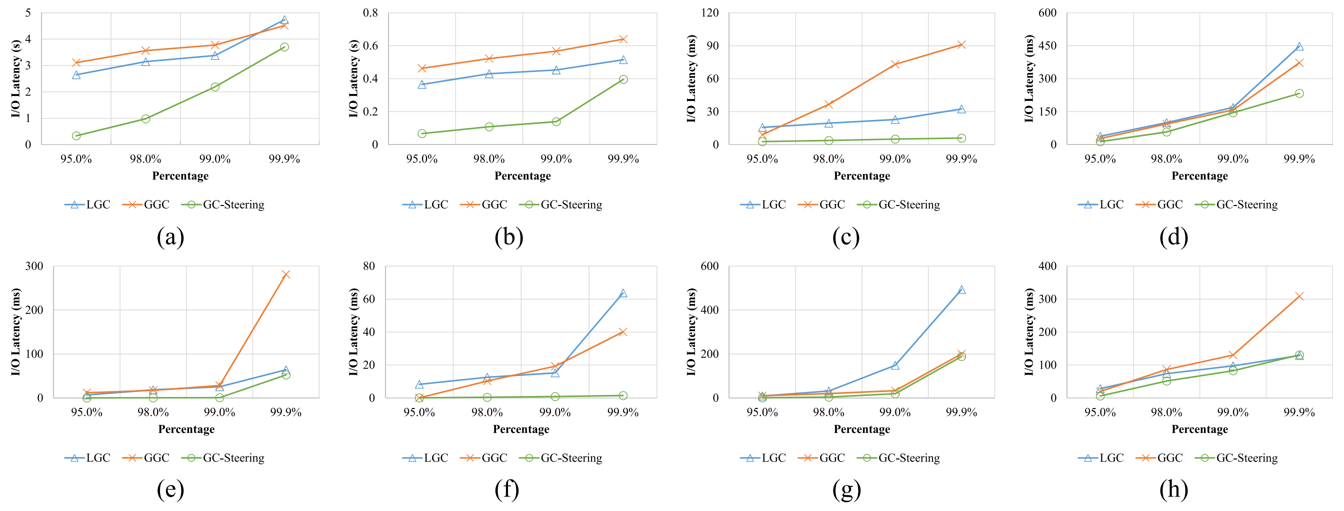


Fig. 9. I/O latencies at the 95th, 98th, 99th, and 99.9th percentiles achieved by the LGC, GGC, and GC-Steering schemes, the lower the better. GC-Steering is consistently the best performer. (a) HPC_W. (b) HPC_R. (c) Financial. (d) hm_0. (e) mds_0. (f) prxy_0. (g) rsrch_0. (h) wdev_0.

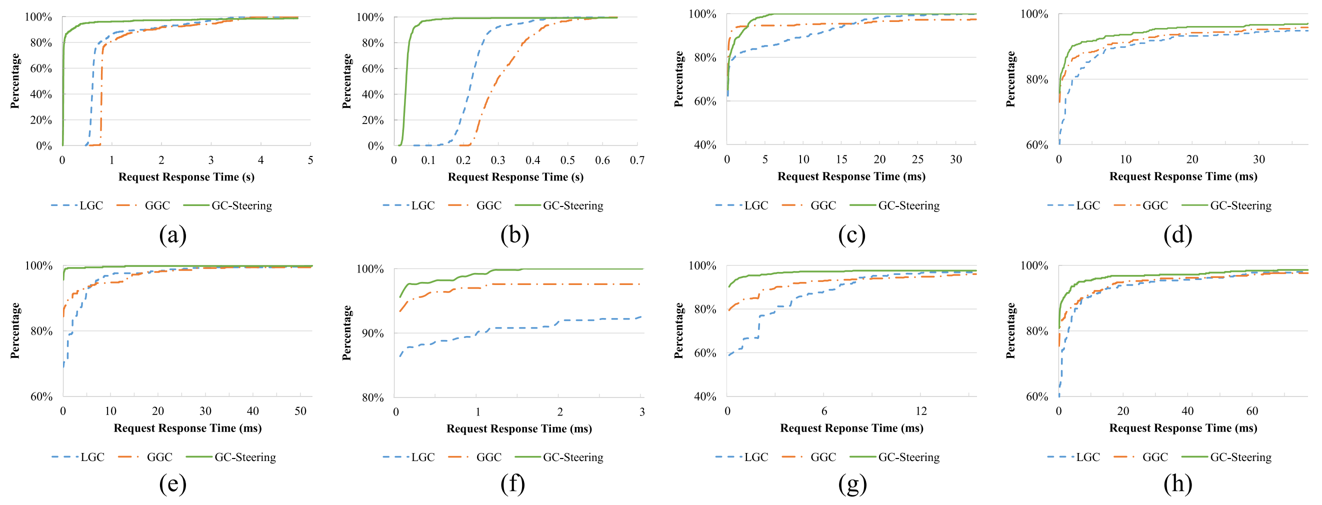


Fig. 10. Cumulative response time distributions for the LGC, GGC, and GC-Steering schemes driven by the different workloads, where the *x*-axis shows the request-response times and the *y*-axis indicates the fraction of requests with response times lower than the corresponding values on the *x*-axis. These results show why GC-Steering has the best tail-latency performance among all the schemes. (a) HPC_W. (b) HPC_R. (c) Financial. (d) hm_0. (e) mds_0. (f) prxy_0. (g) rsrch_0. (h) wdev_0.

Financial trace. The reason is that GGC forces all the SSDs to start their GC processes when any SSD within the RAID set initiates the GC process. During the GC period, the RAID set is unavailable and the user I/O requests will be waiting in the queue until all the SSDs complete their GC processes, which significantly increases the percentage of requests with long latencies, exacerbating the tail latency problem. The results are consistent with the previous results on average response times showing that the GGC scheme has inferior system performance to the LGC and GC-Steering schemes.

2) Sensitivity Study: The GC-Steering performance is likely influenced by several important factors, including the number of SSDs in an RAID, the stripe unit size and the design choice. Fig. 11 shows the sensitivity study on the response time results in GC-Steering driven by the different workloads.

Number of SSDs: To examine the sensitivity of GC-Steering to the number of SSDs in an RAID, we conduct experiments on RAID5 systems consisting of different numbers of SSDs (5 and 7) with a stripe unit size of 64 KB. Fig. 11(a) 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 an 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 an 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 hand, the performance improvement of GC-Steering under the HPC-like workloads is much more significant than that under the realistic enterprize workloads, because the former is 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 five SSDs with

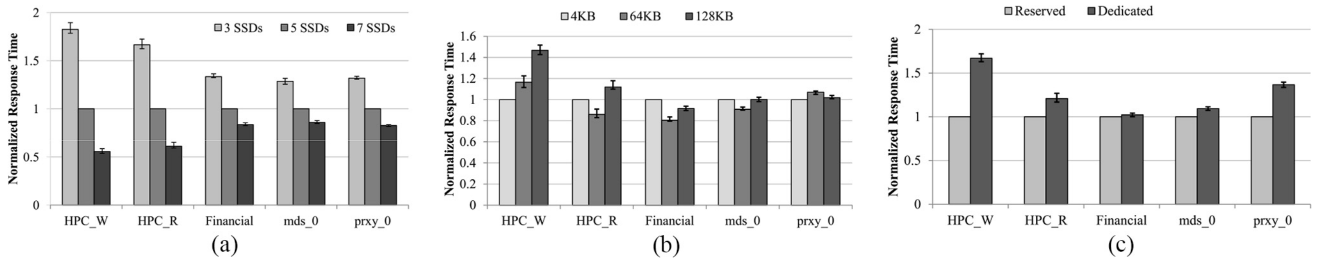


Fig. 11. Sensitivity study of the (a) number of SSDs, (b) stripe unit size, and (c) staging space design on the response time in GC-Steering.

stripe unit sizes of 4 KB, 64 KB, and 128 KB, respectively. Fig. 11(b) 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 [44], i.e., the access types (read or write) and request sizes, which is consistent with the results illustrated in Fig. 11(b). For these workloads, read/write ratios and request sizes are different and MIX, thus no such an optimal fixed stripe unit size is existed. For enterprize and HPC workloads, optimal stripe unit size should be carefully configured and is still an open problem for SSD-based RAIDs [45].

Design Choice of Staging Space: By default, we configure the prereserved 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 five SSDs with a stripe unit size of 64 KB and configure the two mentioned types of staging space. Fig. 11(c) 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 prereserved 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 prereserved 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 prereserved 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 six SSDs with a stripe unit size of 64 KB driven by the different workloads. For LGC, GGC, and GC-Steering (Dedicated), five 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

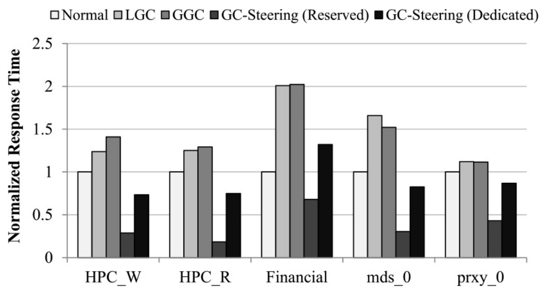


Fig. 12. Average response time during RAID reconstruction, normalized to the average response time when there is no reconstruction.

GC-Steering. Fig. 12 compares the three schemes in terms of average response time during the RAID reconstruction period, normalized to the response time when no reconstruction is underway. The RAID reconstruction bandwidth is set to range between 1 MB/s and 10 MB/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 10 MB/s. Consequently, the reconstruction times for the three schemes are almost the same. However, Fig. 12 shows that for the LGC and GGC schemes, the average user response time during RAID reconstruction is increased by an average of 45.6% and 47.3% above that in the normal state, respectively.

On the other hand, for GC-Steering with prereserved space on each SSD within an RAID (Reserved) and a dedicated SSD (Dedicated), the average response times are 62.3% and 10.1% lower than that in the normal state, respectively. The large discrepancy in improvement between the two GC-Steering configurations stems 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 an RAID accordingly. Second, the parallelism characteristics of flash-based SSDs are exploited during the RAID reconstruction process. In detail, the reconstruction-induced write operations can be performed in parallel on surviving SSDs [as that shown in Fig. 6(b)], thus alleviating the write bottleneck on the replacement SSD. Consequently, Reserved GC-Steering reduces the average response time more significantly than Dedicated GC-String during RAID reconstruction.

To evaluate how the maximum RAID reconstruction bandwidth affects reconstruction performance, we conduct

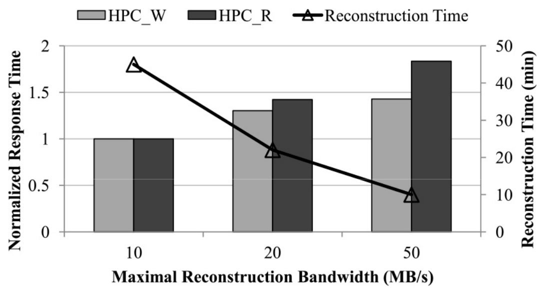


Fig. 13. Impact of the maximum reconstruction bandwidth on the average response time and reconstruction time, normalized to the average response time when maximum reconstruction bandwidth is 10 MB/s in GC-Steering.

experiments that measure reconstruction time and average response time as a function of different maximum reconstruction bandwidth, 10 MB/s, 20 MB/s, and 50 MB/s, respectively. Fig. 13 plots the experimental results on a RAID5 system consisting of five SSDs with a stripe unit size of 64 KB. The reconstruction time decreases with the increasing maximum RAID reconstruction bandwidth. The reason is that the Linux MD software favors the reconstruction process, keeping the RAID reconstruction speed always at the allowed maximum bandwidth. As a result, the user response time is sensitive to and increases proportionally with the increasing maximum reconstruction bandwidth [46] that significantly encroaches on the bandwidth available to user requests. For example, the average user response time increases by an average of 36.3% and 63.1% under the maximum reconstruction bandwidth of 20 MB/s and 50 MB/s compared with that under the maximum reconstruction bandwidth of 10 MB/s, respectively.

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-induced performance degradation problem. This article falls into the latter category and focuses on improving the performance and reliability of SSD-based RAIDs.

To address the first problem, Balakrishnan *et al.* [15] proposed 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 based on their observations that balancing the write traffic among SSDs of an SSD-based RAID5 has an unintended effect of causing correlated failures of these SSDs. 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. It essentially trades performance for reliability for SSD-based RAIDs. Flash-aware RAID [47] reduces the number of internal write operations caused by parity updates by using a delayed parity update strategy and a partial parity technique. The elastic striping and anywhere parity scheme [48] 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.* [49] proposed a grouping-based elastic striping scheme to separately write data chunks in different groups into SSDs by exploiting the workload characteristics.

TABLE II
Comparison of the Related Studies to GC-Steering

Schemes	Environment	Rebuild	Control GC ?
GCaR [12]	FTL within an SSD	No	No
TTFlash [24]			Rotated one by one
GGC [19]	SSD-based RAID0		Concurrently
GC-Steering	SSD-based RAID5/6	Yes	No

Logging is an effective technique to transform small random writes into large writes in storage systems. HPDA [16], an enhanced hybrid RAID4 disk array composed of both HDDs and SSDs, uses two HDDs to service as the dedicated logging parity device, thus avoiding the parity updates on the SSD-based RAID. Similar to HPDA, LDM [50] uses two mirrored HDDs as a logging write buffer that temporally absorbs the small write requests. EPLog [51] mitigates the parity update overhead by redirecting the parity traffic to a separate log HDD and uses the elastic parity construction scheme to eliminate the need for prereading data in parity computations. By converting small random writes into large writes, the frequency of parity updates is reduced, thus improving both the system performance and reliability.

However, none of the above studies considers the GC activities of SSDs in SSD-based RAIDs. Kim *et al.* [19] find that the uncoordinated GC operations on individual SSDs amplify the performance degradation of SSD-based RAIDs. To address the problem, they propose an RAID-level 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 [52] that optimizes the reconstruction performance for HDD-based RAIDs, GC-Steering effectively exploits the workload characteristics and the reserved space on each SSD within SSD-based RAIDs, to alleviate the negative performance and reliability impact of GC operations on SSD-based RAIDs. Importantly, GC-Steering does 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 treats individual constitute SSDs of an RAID as black boxes, thus oblivious of the SSD internal GC workflow, and instead focuses on the RAID reconstruction to improve both performance and reliability of SSD-based RAID5/6.

However, RAID reconstruction has not been studied for SSD-based RAIDs. 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 previous studies [46], [52], [53]. However, these studies on HDD-based RAID reconstruction try to either make the reconstruction I/O requests sequential on HDDs [54], [55] or alleviate the interference between user I/O requests and reconstruction I/O requests [46], [52]. 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, this article reveals that significant access parallelism can be exploited in SSD-based RAIDs reconstruction by leveraging the high random-access performance of flash-based SSDs, which is shown to be 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 [32], [33]. This article represents but a first phase of substantial work required to investigate efficient failure recovery algorithms for SSD-based RAIDs.

VI. Conclusion

Straightforwardly applying the RAID algorithms to SSD-based disk arrays can not fully exploit the device advantages of flash-based SSDs and may degrade the performance and reliability of SSD-based RAIDs. Especially, the GC operations being performed on individual SSDs can cause severe performance variability and tail latency in SSD-based RAIDs. Moreover, the traditional HDD-based RAID reconstruction algorithms are not suitable for SSD-based RAIDs. To address these problems, we propose the GC-Steering scheme to alleviate the GC impact on the performance and reliability of SSD-based RAIDs by exploiting both the flash device and workload characteristics to alleviate the contention between user I/O requests and GC/recovery-induced internal requests. The extensive evaluation on a lightweight prototype of GC-Steering shows that GC-Steering reduces the average response times than the state-of-the-art schemes LGC and GGC by an average of 63.3% and 65.8%, respectively. Moreover, GC-Steering also significantly reduces the user response time during the RAID reconstruction period by an average of 62.3%.

References

[1] (2018). *Worldwide Enterprise Storage Market*. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS43964118

[2] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating server storage to SSDs: Analysis of tradeoffs," in *Proc. 4th Eur. Conf. Comput. Syst. (EuroSys)*, Mar. 2009, pp. 145–158.

[3] A. M. Caulfield *et al.*, "Understanding the impact of emerging non-volatile memories on high-performance, IO-intensive computing," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal. (SC)*, New Orleans, LA, USA, Nov. 2010, pp. 1–11.

[4] F. Chen, D. A. Koufaty, and X. Zhang, "Hystor: Making the best use of solid state drives in high performance storage systems," in *Proc. 25th ACM Int. Conf. Supercomput. (ICS)*, Tucson, Arizona, Jun. 2011, pp. 22–32.

[5] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proc. USENIX Annu. Tech. Conf. (USENIX)*, Boston, MA, USA, Jun. 2008, pp. 57–70.

[6] N. Shahidi, M. Arjomand, M. Jung, M. Kandemir, C. R. Das, and A. Sivasubramaniam, "Exploring the potentials of parallel garbage collection in SSDs for enterprise storage systems," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal. (SC)*, Salt Lake City, UT, USA, Nov. 2016, pp. 561–572.

[7] J. Lee, Y. Kim, G. M. Shipman, S. Oral, F. Wang, and J. Kim, "A semi-preemptive garbage collector for solid state drives," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Austin, TX, USA, Apr. 2011, pp. 12–21.

[8] Y. Oh, J. Choi, D. Lee, and S. H. Noh, "Caching less for better performance: Balancing cache size and update cost of flash memory cache in hybrid storage systems," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST)*, San Jose, CA, USA, Feb. 2012, p. 25.

[9] S. Park and K. Shen, "FIOS: A fair, efficient flash I/O scheduler," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST)*, San Jose, CA, USA, Feb. 2012, p. 13.

[10] G. Wu and B. He, "Reducing SSD read latency via NAND flash program and erase suspension," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST)*, San Jose, CA, USA, Feb. 2012, p. 10.

[11] (2017). *SNIA SSS Performance Test Specification (PTS) Testing Service*. [Online]. Available: http://www.snia.org/forums/sssi/ptstest

[12] S. Wu, Y. Lin, B. Mao, and H. Jiang, "GCaR: Garbage collection aware cache management with improved performance for flash-based SSDs," in *Proc. 30th Int. Conf. Supercomput. (ICS)*, Istanbul, Turkey, Jun. 2016, pp. 1–12.

[13] S. Yan *et al.*, "Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs," in *Proc. 15th USENIX Conf. File Storage Technol. (FAST)*, Santa Clara, CA, USA, Feb. 2017, pp. 15–28.

[14] D. A. Patterson, G. A. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proc. Int. Conf. Manag. Data (SIGMOD)*, Chicago, IL, USA, Jun. 1988, pp. 109–116.

[15] M. Balakrishnan, A. Kadav, V. Prabhakaran, and D. Malkhi, "Differential RAID: Rethinking RAID for SSD reliability," in *Proc. 5th Eur. Conf. Comput. Syst. (EuroSys)*, Paris, France, Apr. 2010, pp. 15–26.

[16] B. Mao *et al.*, "HPDA: A hybrid parity-based disk array for enhanced performance and reliability," *ACM Trans. Storage*, vol. 8, no. 1, p. 4, 2012.

[17] N. Jeremic, G. Mühl, A. Busse, and J. Richling, "The pitfalls of deploying solid-state drive RAIDs," in *Proc. 4th Annu. Int. Conf. Syst. Storage (SYSTOR)*, Haifa, Israel, May 2011, pp. 1–13.

[18] Y. Li, P. P. C. Lee, and J. C. S. Lui, "Analysis of reliability dynamics of SSD RAID," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1131–1144, Apr. 2016.

[19] Y. Kim, S. Oral, G. M. Shipman, J. Lee, D. A. Dillow, and F. Wang, "Harmonia: A globally coordinated garbage collector for arrays of solid-state drives," in *Proc. 27th IEEE Symp. Mass Storage Syst. Technol. (MSST)*, Denver, CO, USA, May 2011, pp. 1–12.

[20] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A large-scale study of flash memory failures in the field," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst. (SIGMETRICS)*, Portland, OR, USA, Jun. 2015, pp. 177–190.

[21] B. Schroeder, R. Lagisetty, and A. Merchant, "Flash reliability in production: The expected and the unexpected," in *Proc. 14th USENIX Conf. File Storage Technol. (FAST)*, San Jose, CA, USA, Feb. 2016, pp. 67–80.

[22] S. Wu, H. Li, B. Mao, X. Chen, and K.-C. Li, "Overcome the GC-induced performance variability in SSD-based RAIDs with request redirection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 5, pp. 822–833, May 2019.

[23] P. Yang *et al.*, "Reduce garbage collection overhead in SSD based on workload prediction," in *Proc. 11th USENIX Workshop Hot Topics Storage File Syst. (HotStorage)*, Renton, WA, USA, Jul. 2019, pp. 1–6.

[24] 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 *Proc. 14th USENIX Conf. File Storage Technol. (FAST)*, San Jose, CA, USA, Feb. 2016, pp. 263–276.

[25] S. Wu, B. Mao, Y. Lin, and H. Jiang, "Improving performance for flash-based storage systems through GC-aware cache management," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2852–2865, Oct. 2017.

[26] J. Kim, K. Lim, Y. Jung, S. Lee, C. Min, and S. H. Noh, "Alleviating garbage collection interference through spatial separation in all flash arrays," in *Proc. USENIX Annu. Techn. Conf. (USENIX)*, Renton, WA, USA, Jul. 2019, pp. 799–812.

[27] J. L. Hennessy and D. A. Patterson, "Towards availability benchmarks: A case study of software RAID systems," in *Computer Architecture: A Quantitative Approach*, 4th ed. Cambridge, MA, USA: Elsevier, 2006.

[28] J. Dean and L. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[29] J. Li, N. Sharma, D. R. K. Ports, and S. D. Gribble, "Tales of the tail: Hardware, OS, and application-level sources of tail latency," in *Proc. 5th ACM Symp. Cloud Comput. (SOCC)*, Seattle, WA, USA, Nov. 2014, pp. 1–14.

[30] H. Lu, B. Saltaformaggio, R. Kompella, and D. Xu, "vFair: Latency-aware fair storage scheduling via per-IO cost-based differentiation," in *Proc. 6th ACM Symp. Cloud Comput. (SOCC)*, Nov. 2015, pp. 125–138.

[31] B. Mao and S. Wu, "Exploiting request characteristics and internal parallelism to improve SSD performance," in *Proc. 33rd IEEE Int. Conf. Comput. Design (ICCD)*, New York, NY, USA, Oct. 2015, pp. 447–450.
[32] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding intrinsic characteristics and system implications of flash memory based solid state drives," in *Proc. 11th ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst. (SIGMETRICS)*, Seattle, WA, USA, Jun. 2009, pp. 81–192.
[33] 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 *Proc. 15th USENIX Conf. File Storage Technol. (FAST)*, San Jose, CA, USA, Feb. 2016, pp. 271–284.
[34] (2017). *Samsung Report*. [Online]. Available: http://news.cnet.com/8301-13924_3-9876557-64.html
[35] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu, "WARM: Improving NAND flash memory lifetime with write-hotness aware retention management," in *Proc. 31st Int. Conf. Massive Storage Syst. Technol. (MSST)*, Santa Clara, CA, USA, Jun. 2015, pp. 1–14.
[36] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Trans. Storage*, vol. 4, no. 3, p. 10, 2008.
[37] Y. Yang and J. Zhu, "Write skew and Zipf distribution: Evidence and implications," *ACM Trans. Storage*, vol. 12, no. 4, p. 21, 2016.
[38] Q. Li *et al.*, "Access characteristic guided read and write cost regulation for performance improvement on flash memory," in *Proc. 14th USENIX Conf. File Storage Technol. (FAST)*, San Jose, CA, USA, Feb. 2016, pp. 125–132.
[39] M. Bjørling, J. Gonzalez, and P. Bonnet, "LightNVM: The Linux open-channel SSD subsystem," in *Proc. 15th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2017, pp. 359–374.
[40] (2017). *Data Reduction in Pure Storage*. [Online]. Available: https://www.purestorage.com/products/purity/purity-reduce.html
[41] (2017). *With Nimble, Less Is More*. [Online]. Available: https://www.nimblestorage.com/its-all-about-data-reduction/
[42] (2017). *OLTP Application I/O*. [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage
[43] (2017). *Block I/O Traces in SNIA*. [Online]. Available: http://iotta.snia.org/tracetypes/3
[44] F. Salmasi, H. Asadi, and M. GhasemiGol, "Impact of stripe unit size on performance and endurance of SSD-based RAID arrays," *Scientia Iranica*, vol. 20, no. 6, pp. 1978–1998, 2013.
[45] S. Wu, W. Yang, B. Mao, and Y. Lin, "MC-RAIS: Multi-chunk redundant array of independent SSDs with improved performance," in *Proc. 15th Int. Conf. Algorithms Archit. Parallel Process. (ICA3PP)*, Nov. 2015, pp. 18–32.
[46] S. Wu, H. Jiang, and B. Mao, "Proactive data migration for improved storage availability in large-scale data centers," *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2637–2651, Sep. 2015.
[47] S. Im and D. Shin, "Flash-aware RAID techniques for dependable and high-performance flash memory SSD," *IEEE Trans. Comput.*, vol. 60, no. 61, pp. 80–92, Jan. 2011.
[48] J. Kim, J. Lee, J. Choi, D. Lee, and S. Noh, "Improving SSD reliability with RAID via elastic striping and anywhere parity," in *Proc. 43th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DSN)*, Budapest, Hungary, Jun. 2013, pp. 1–12.
[49] Y. Pan, Y. Li, Y. Xu, and Z. Li, "Grouping-based elastic striping with hotness awareness for improving SSD RAID performance," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DSN)*, Rio de Janeiro, Brazil, Jun. 2015, pp. 160–171.
[50] S. Wu, B. Mao, X. Chen, and H. Jiang, "LDM: Log disk mirroring with improved performance and reliability for SSD-based disk arrays," *ACM Trans. Storage*, vol. 12, no. 4, pp. 1–22, 2016.
[51] Y. Li, H. Chan, P. P. C. Lee, and Y. Xu, "Elastic parity logging for SSD RAID arrays," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DSN)*, Toulouse, France, Jun. 2016, pp. 49–60.
[52] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao, "WorkOut: I/O workload outsourcing for boosting the RAID reconstruction performance," in *Proc. 7th USENIX Conf. File Storage Technol. (FAST)*, San Francisco, CA, USA, Feb. 2009, pp. 239–252.
[53] N. Wang, Y. Xu, Y. Li, and S. Wu, "OI-RAID: A two-layer RAID architecture towards fast recovery and high reliability," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DSN)*, Toulouse, France, Jun. 2016, pp. 61–72.
[54] M. Holland, "On-line data reconstruction in redundant disk arrays," Dept. Elect. Comput. Eng., Carnegie Mellon Univ., Pittsburgh, PA, USA, Rep. CMU-CS-94-164, Apr. 1994.
[55] L. Tian *et al.*, "PRO: A popularity-based multi-threaded reconstruction optimization for RAID-structured storage systems," in *Proc. USENIX Conf. File Storage Technol. (FAST)*, San Jose, CA, USA, Feb. 2007, pp. 277–290.

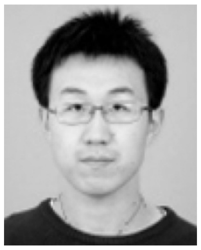
Suzhen Wu (Member, IEEE) received the B.E. and Ph.D. degrees in computer science and technology and computer architecture from the Huazhong University of Science and Technology, Wuhan, China, in 2005 and 2010, respectively.

She has been an Associate Professor with the Computer Science Department, Xiamen University, Xiamen, China, since August 2014. She has over 50 publications in journal and international conferences, including the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the *ACM Transactions on Storage*, FAST, LISA, ICS, ICCD, MSST, ICDCS, DATE, SRDS, and IPDPS. Her research interests include computer architecture and storage system.

Dr. Wu is a member of ACM.

Weidong Zhu (Student Member, IEEE) received the B.E. degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2016. He is currently pursuing the master's degree with the Computer Science Department, Xiamen University, Xiamen, China.

His research interests include flash-based storage systems, SSD-based disk arrays, key-value store, and data deduplication.



Yingxin Han is currently pursuing the master's degree with the Computer Science Department, Xiamen University, Xiamen, China.

His research interests include flash-based storage systems and SSD-based disk arrays.

Hong Jiang (Fellow, IEEE) received the B.Sc. degree in computer engineering from the Huazhong University of Science and Technology, Wuhan, China, in 1982, the M.A.Sc. degree in computer engineering from the University of Toronto, Toronto, Canada, in 1987, and the Ph.D. degree in computer science from Texas A&M University, College Station, TX, USA, in 1991.

He is currently the Chair and the Wendell H. Nedderman Endowed Professor with the Computer Science and Engineering Department, University of Texas at Arlington, Arlington, TX, USA. His present research interests include computer architecture, computer storage systems and parallel I/O, high-performance computing, big data computing, cloud computing, and performance evaluation. He has over 300 publications in major journals and international conferences in the above areas, including the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON COMPUTERS, the PROCEEDINGS OF IEEE, TACO, TOS, ISCA, MICRO, ATC, FAST, EUROSYS, and SOCC.

Dr. Jiang is a member of ACM.

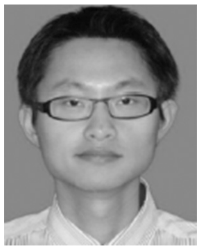
Bo Mao (Member, IEEE) received the B.E. degree in computer science and technology from Northeast University, Shenyang, China, in 2005, and the Ph.D. degree in computer architecture from the Huazhong University of Science and Technology, Wuhan, China, in 2010.

He is an Associate Professor with the Software School, Xiamen University, Xiamen, China. He has over 50 publications in international journals and conferences, including the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the *ACM Transactions on Storage*, FAST, LISA, ICS, ICCD, MSST, ICDCS, DATE, SRDS, Cluster, and IPDPS. His research interests include storage system and cloud computing.

Dr. Mao is a member of ACM.

Zhijie Huang (Member, IEEE) received the Ph.D. degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2016.

He is currently a Postdoctoral Research Associate with the Computer Science and Engineering Department, University of Texas at Arlington, Arlington, TX, USA. He has published many papers in major journals and international conferences, including the IEEE TRANSACTIONS ON COMMUNICATIONS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE COMMUNICATIONS LETTERS, INFOCOM, and ISIT. His research interests include coding theory and its application, dependable and secure systems, storage systems and parallel I/O, and embedded systems.



Liang Chen received the B.E. and master's degrees in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2006 and 2008, respectively.

He is currently a Lecture with the College of Engineering, Fuzhou Institute of Technology, Fujian, China. His research interests include flash-based SSDs and SSD-based disk arrays.