Improving the SSD Performance by Exploiting Request Characteristics and Internal Parallelism

Bo Mao, *Member, IEEE*, Suzhen Wu, *Member, IEEE*, and Lide Duan, *Member, IEEE*

***Abstract*—With the explosive growth in the data volume, the I/O bottleneck has become an increasingly daunting challenge for big data analytics. It is urgent and important to introduce high-performance flash-based solid state drives (SSDs) into the storage systems. However, since the existing systems are primarily designed for conventional magnetic hard disk drives, directly incorporating SSDs in the existing systems cannot fully exploit SSDs' performance advantages. In this paper, we propose a new I/O scheduler for SSDs, namely Amphibian, that exploits the high-level request characteristics and low-level parallelism of flash chips to improve the performance of SSD-based storage systems. Amphibian includes two performance enhancement schemes: 1) size-based request ordering, which prioritizes requests with small sizes in processing and 2) garbage collection (GC)-aware request dispatching that delays issuing requests to flash chips that are in the GC state. These two schemes employed in Amphibian significantly reduce the average waiting times of the requests from the host. Our extensive evaluation results derived from three types of SSDs show that, compared with the existing I/O schedulers, Amphibian greatly improves both throughput and average response times for SSD-based storage systems, thus improving the I/O performance of the systems.**

***Index Terms*—Garbage collection (GC)-aware, I/O scheduler, internal parallelism, request characteristics, solid state drive (SSD).**

I. Introduction

HARD disk drives (HDDs) have become the performance wall of storage systems due to the slowness of their mechanical positioning nature. Recently, flash-based solid state drives (SSDs) have become an attractive alternative to HDDs, drawing a great deal of attention in both academia and industry [1]–[3]. In addition to being employed in mobile devices and desktop/laptop computers, SSDs are also increasingly applied in high performance computing and enterprise environments. However, with the existing systems primarily designed for magnetic HDDs, directly replacing HDDs with SSDs in the existing systems cannot fully exploit the performance advantages of high-performance SSDs.

Among the different I/O layers in a storage system, the I/O scheduler is highly critical to the performance of the system. The I/O scheduling algorithm used in the I/O scheduler directly affects the working efficiency of HDDs and SSDs. Traditional I/O scheduling algorithms are merely designed for HDDs: they are address-based, and try to sort the requests in a way that minimizes the head seeking distances. Since HDDs are mechanical devices accessing data through the head movement, their response times are closely related to the addresses of incoming access requests [4], [5]. Sequential accessing can effectively reduce the cost of the head tracking movement, and is thereby favored by HDDs. For example, in the SCAN (elevator) disk-scheduling algorithm, the disk head moves in one direction until it reaches the edge of the disk when servicing requests, thus avoiding frequent head movements and the unnecessary seeking time [5].

Different from HDDs, SSDs are based on semiconductor chips and have no mechanical parts. In SSDs, data signals are transmitted completely through circuits without using any mechanical heads. Therefore, traditional I/O scheduling algorithms designed for HDDs may be suboptimal for SSDs. Compared with HDDs, SSDs provide a large variety of benefits, including low power consumption, high robustness to vibrations and temperature, and, most importantly, high small-random-read performance. However, SSDs also have a number of disadvantages, such as high cost, low small-random-write performance, and limited lifetime [1]. Apart from the asymmetric performance of reads and writes in flash-based SSDs, our evaluation, together with some previous studies, also show that the response time of a certain request is linear to the request size in SSDs [4]. Moreover, the inherent garbage collection (GC) operations in SSDs also significantly affect the user I/O performance [6]–[8]. Therefore, without deep design considerations specifically for SSDs, the performance advantages of flash-based SSDs cannot be fully exploited.

Based on the above observations, we propose a novel I/O scheduler for flash-based SSDs, called Amphibian, to improve the I/O performance for SSD-based storage systems by exploiting both high-level request characteristics and low-level internal parallelism of flash chips. At the high-level, Amphibian utilizes the asymmetric read and write performance

Manuscript received December 13, 2016; revised March 3, 2017; accepted April 14, 2017. Date of publication April 25, 2017; date of current version January 19, 2018. This work was supported in part by the National Natural Science Foundation of China under Grant 61472336 and 61402385, in part by the Key Laboratory of Information Storage System, Ministry of Education of China, in part by the Huawei Innovation Research Program, and in part by the National Science Foundation Computing and Communication Foundations under Grant 1566158. This paper was recommended by Associate Editor Z. Shao. *(Corresponding author: Suzhen Wu.)*

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

S. Wu is with the Computer Science Department, Xiamen University, Xiamen 361005, China (e-mail: suzhen@xmu.edu.cn).

L. Duan is with the Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX 78249 USA (e-mail: lide.duan@utsa.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2697961

characteristics of flash-based SSDs, prioritizing read requests over write requests to improve the overall performance. Furthermore, it processes the requests with smaller data sizes ahead of those with larger sizes in the I/O waiting queue to reduce the average waiting time of the requests. At the low-level, by identifying which flash chips are in the GC state, Amphibian employs a GC-aware request dispatching scheme to delay issuing requests to flash chips that are in the GC state, thus fully exploiting the internal parallelism of SSD chips. The evaluation results conducted on three types of SSDs show that, compared with the state-of-the-arts, Amphibian improves both throughput and average request response times significantly. Consequently, the I/O performance of the SSD-based storage systems is improved.

To the best of our knowledge, Amphibian is the first I/O scheduler that exploits both request characteristics and internal parallelism of SSDs. This paper achieves the following contributions.

1) We quantitatively demonstrate that the average response time of flash-based SSDs approximately grows linearly with the data size of the request. Thus, we propose a size-based request ordering scheme for flash-based SSDs to reduce the queuing times of the requests.
2) We propose a GC-aware request dispatching scheme that exploits the internal parallelism of flash-based SSD chips. The proposed scheme reduces the conflicts between the user requests and the internal GC-induced I/O traffic.
3) We conduct extensive evaluations with both benchmark and trace-driven experiments to show the effectiveness of Amphibian. The evaluation results show that Amphibian can achieve up to 28.3% improvement in system throughput, and 18.0% improvement in request response time on average.

The rest of this paper is organized as follows. Background and motivation are presented in Section II. We describe the design details of Amphibian in Section III. Performance evaluation is presented in Section IV, and the related work is discussed in Section V. We finally conclude this paper in Section VI.

II. Background and Motivation

In this section, we present two key characteristics of flash-based SSDs in contrast to those of magnetic HDDs based on our experiments and analysis. These observations motivate our proposed new I/O scheduler for flash-based SSDs.

A. SSD Basics

Unlike mechanical HDDs, flash-based SSDs are made of semiconductor chips and do not have moving parts (i.e., mechanical positioning parts) [9]. In addition to high energy-efficiency and high random-read performance, flash-based SSDs have the following unique characteristics different from HDDs [1], [9].

First, the read performance and write performance of flash-based SSDs are asymmetric. To better describe the performance of NAND flash chips, Table I compares the random read, program (write), and erase times for three types of SSD cells: 1) SLC, which stores a single bit of data per cell; 2) MLC, which stores two bits per cell; and 3) TLC, which stores three bits per cell. In other words, MLC can store twice the amount of data compared to SLC. However, the read performance of MLC is much slower than that of SLC; and the program performance of MLC is also much slower than that of SLC [10], [11]. Among the three, TLC chips have the lowest read/program/erase performance [10]. As can be seen, the read operation is the fastest, while the erase operation is two orders of magnitude slower than the read. Although the write operation is faster than the erase operation, it is still 10–20 times slower than the read operation.

TABLE I
Read/Program/Erase Times for SLC/MLC/TLC Flash Chips [10]

Operations	SLC chips	MLC chips	TLC chips
Random Read	20us	40us	80us
Program	260us	900us	2.3ms
Erase	2ms per block	2ms per block	10ms per block

Second, flash-based SSDs exhibit a unique characteristic known as "erase-before-write" that requires a whole flash block (consisting of multiple pages) be erased before any part of it can be rewritten. As a result, for a write operation to any part of a block, all other valid data in the block need to be read out first and then stored, together with the new written data, to another free block. Due to the sheer size of a block, an erase operation typically takes a time in the range of milliseconds, which is one or two orders of magnitude slower than the read operation [12], [13]. Consequently, SSDs demonstrate poor performance when servicing small random-write requests.

Third, GC operations in SSD significantly affect the user I/O performance [7], [8], [14]. GC eliminates the need to erase the whole block prior to a write operation. Instead, it marks the block that needs to be erased as "garbage," and performs whole block erase as space reclamation before the block becomes free and can be rewritten. GC accumulates data blocks previously marked for deletion, performs a whole block erasure on each garbage block, and returns the reclaimed space for reuse. In practice, when the number of free blocks in an SSD is smaller than the preset threshold, the valid pages in the victim blocks (i.e., to be erased) must be copied to a different free block and the victim blocks are erased to be new free blocks. The GC process increases the queueing times of the user requests, thus significantly degrading both read and write performance.

B. Response Time Versus Request Size

In order to understand the relation between the response time and the user request size for HDDs and SSDs, we use the IOmeter tool [15] to measure the average response times of an HDD (WDC WD1600AAJS) and an SSD (Intel X25-E 64 GB) under different request sizes. Fig. 1 shows the normalized results.

The experimental results show that, for the HDD, the response time increases very slowly with the increasing user request size. As shown in Fig. 1(a), when the request size

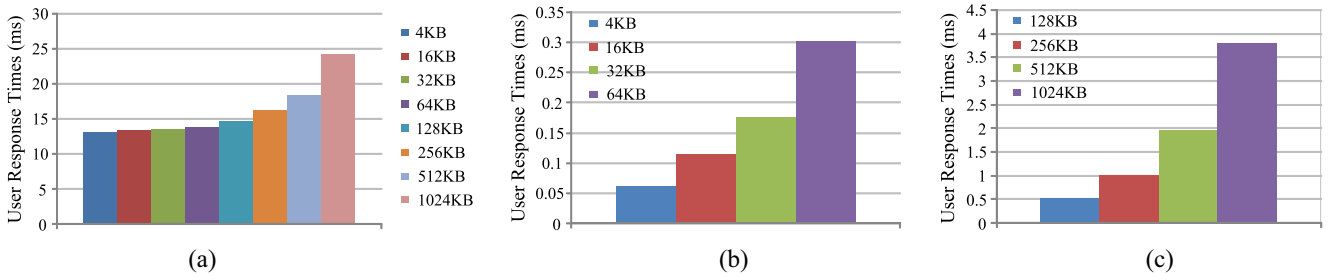

Fig. 1. Response times driven by random accesses of different request sizes for (a) HDD and (b) and (c) SSD.

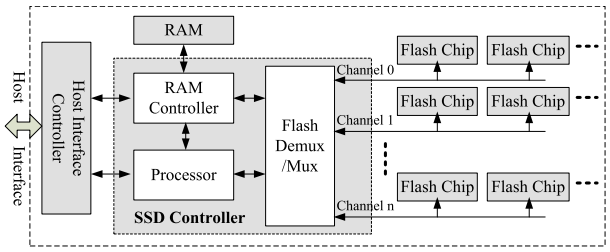

Fig. 2. Overview of the internal parallelism for a typical SSD consisting of multiple flash chips.

increases from 1 to 128 KB, the average response time of the HDD almost remains unchanged. This is because, for random accesses in HDDs, the data seek time and rotational latency are much larger than the data transfer time, dominating the response time to the request [16]. For the same reason, most of the I/O schedulers designed for HDDs sort the user requests by their physical addresses in order to utilize the data locality.

In contrast, from Fig. 1(b) and (c), we can see that the average response time of the SSD grows nearly linearly with the increased request size. Since SSDs are not mechanical devices, the read and write operations transmit the signal completely through circuits. Neither disk head seeking nor rotational spinning is performed. Thus, the data transfer time, which is directly related to the request size, is the main part of the user response time.

C. Internal Parallelism of SSDs

To increase the storage capacity, multiple flash chips are integrated into an SSD. Fig. 2 depicts an schematic overview of a typical SSD architecture. Each of the *n* independent channels is shared by multiple flash chips. SSDs are an inherently highly parallelized architecture [17]. It comprises multiple units, including pages, blocks, planes, channels and packages. Different constituent operational units can operate in parallel, thus providing the potential to achieve better performance.

Existing research on the SSD parallelism include discussing internal design alternatives [1] and issuing concurrent requests to SSDs to exploit their inherent parallelism for better performance [3], [18]–[20]. Our goal in this paper is to exploit the internal parallelism to avoid issuing requests to the flash chips that are in the GC state. Issuing requests to the flash chips that are in the GC state not only increases the waiting time of the requests, but also worsens the contention between the user I/O requests and the background GC operations.

When deploying SSDs in high performance computing and enterprise storage systems, the existing I/O system software cannot fully utilize the performance advantages of the flash-based SSDs [21], [22]. In addition to the asymmetric read and write performance of SSDs, our evaluation results have shown that, for flash-based SSDs, the response times of the requests with the same request type are linear with the request sizes. Moreover, the inherent GC operations in SSDs significantly affect the user I/O performance. Therefore, it is necessary to redesign the system software to take into consideration SSD-specific characteristics, including the asymmetric read–write performance, the size-latency relationship, and the GC-induced performance degradation. However, existing I/O schedulers for SSDs only consider the request fairness [18], [23], [24] or the parallelism characteristics [25]–[28]. They are unaware of the high-level request sizes and the low-level GC operations.

Based on the above observations and analysis, we propose Amphibian, a new I/O scheduler for flash-based SSDs, to improve the I/O performance of SSD-based storage systems. Amphibian exploits both high-level request characteristics and low-level internal parallelism to improve the I/O performance. On the one hand, it preferentially processes the small requests in the I/O waiting queue to reduce the average waiting times of the requests. On the other hand, it utilizes the asymmetric read/write performance feature and the internal parallelism characteristics of the flash-based SSDs to avoid issuing the user requests to the flash chips that are in the GC state, thus alleviating the contention between the user requests and the GC operations to improve the overall system performance.

III. Amphibian

In this section, we first outline the main principles guiding the design of Amphibian. Then we present a system overview of Amphibian, followed by a description of the request type-based queueing, request size-based ordering, and GC-aware request dispatching in Amphibian.

A. Design Principles

The design of Amphibian aims to achieve high performance, high applicability, and high portability, as explained below.

1) *High Performance:* Amphibian strives to reduce the user average response times by exploiting both request characteristics and internal parallelism in the I/O scheduler algorithm for flash-based SSDs, thus reducing the waiting time in the I/O queue for the user I/O requests.

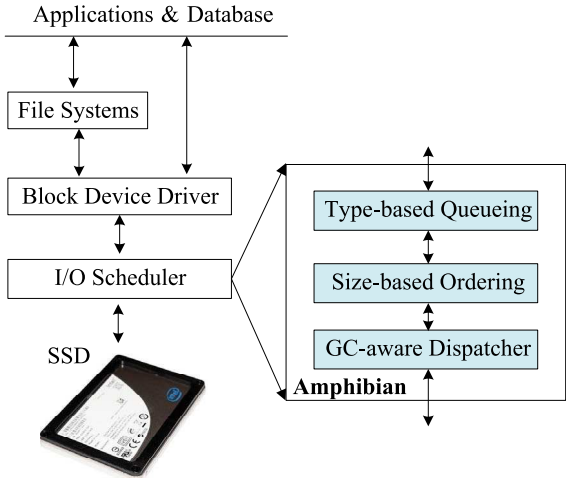

Fig. 3. System overview of Amphibian on the I/O path.

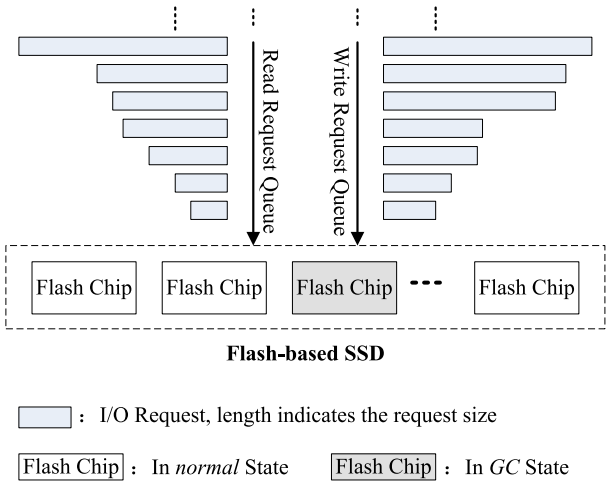

Fig. 4. Request type-based queueing in Amphibian.

2) *High Applicability:* Amphibian exploits the external request characteristics and the internal parallelism, both of which are common features for commercial SSDs. Therefore, Amphibian is applicable to all SSDs, including SATA SSDs and PCI-e SSDs. Our evaluations also validate the applicability of Amphibian.
3) *High Portability:* The asymmetric read–write performance, the size-latency relation and the GC operations are common for most commercial SSDs. Amphibian can be easily extended to any existing SSD-based I/O schedulers, such as FIOS [29] and ParDispatcher [28], to further improve the system performance. Moreover, Amphibian can also be ported to the SSD-based disk arrays to improve the system efficiency.

B. System Overview of Amphibian

Fig. 3 shows a system overview of our proposed Amphibian on the I/O path of the SSD-based system. Amphibian, which is located between the block device layer and the device driver layer, determines the request service order according to the request characteristics. As shown in Fig. 3, on the top level of Amphibian, user requests are enqueued into the I/O scheduling module, based on the types and sizes of the requests. On the bottom level of Amphibian, the scheduling strategy selects a request to be serviced next. The Amphibian scheduling algorithm not only improves the throughput of the SSD-based storage system, but also reduces the average response time.

The main goal of Amphibian is to reduce the average user response times of the SSD-based storage system. It consists of three functional modules: 1) type-based queueing; 2) size-based ordering; and 3) the GC-aware request dispatcher. The *type-based queueing* module separates the read and write requests into different queues. The *size-based ordering* module is responsible for sorting requests in both read and write request queues based on the request sizes, prioritizing requests with small sizes. The *GC-aware request dispatcher* module issues the requests to the corresponding flash chips while avoiding issuing the requests to the flash chips that are in the GC state. The first two modules exploit the high-level request characteristics, i.e., request types and sizes, while the third module exploits the internal parallelism of SSDs. By fully exploiting both high-level request characteristics and internal parallelism, the incoming requests can execute much more efficiently.

C. Request Type-Based Queueing

The read performance and write performance of flash-based SSDs are asymmetric. The read performance is much better than the write performance [30]. In addition, read operations are synchronous in the application layer. Upon pending read requests, user applications will be blocked until the requested data is returned to the applications. On the other hand, write operations are usually asynchronous and do not block the user applications. If a read request needs to wait for the completion of a write request, the critical read latency is unnecessarily increased. Motivated by this observation, Amphibian prioritizes read requests over write requests, preventing read requests from being blocked by write requests and thereby reducing the average waiting time of read requests in the I/O queue. Consequently, this type-based queueing module inserts an incoming request to the read request queue or the write request queue based on the request's type.

Fig. 4 shows the request type-based queueing in Amphibian. If the read request queue is not empty, Amphibian processes read requests first. Only when there are no read requests, Amphibian will process the write requests in the write request queue. This method ensures that read requests are never blocked by write requests. However, it may cause a write request wait too long. In order to prevent such write starvation from occurring, Amphibian sets a threshold value as the longest waiting time for write requests. When a write request's waiting time reaches the threshold, the write request cannot be blocked by read requests anymore and will be immediately processed by Amphibian. The threshold scheme is also used in the deadline I/O scheduler for HDDs [5]. It is worth noting that request type-based queueing has been

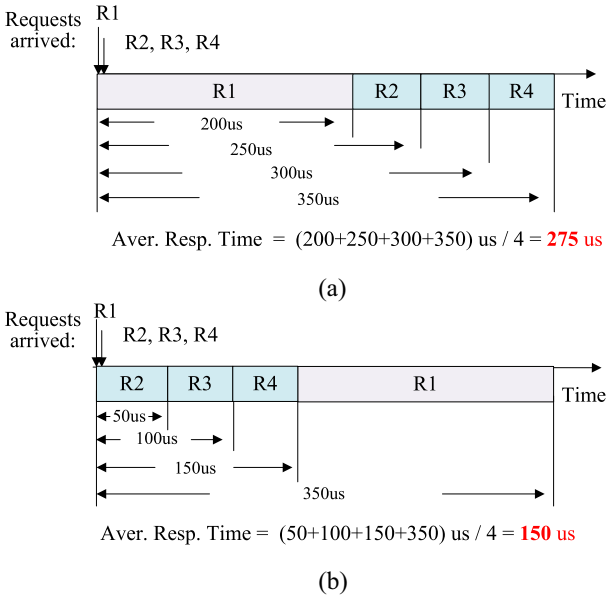

Fig. 5. Example of request size-based ordering. (a) Noop. (b) Amphibian.

used before [25], [29], [31] for SSD-based I/O schedulers. We utilize it here to facilitate the other modules in Amphibian.

D. Request Size-Based Ordering

The request size-based ordering is inspired by the shortest-job-first scheduling policy [5]. In conventional HDD-based I/O schedulers, the shortest jobs are not determined by the request sizes, but the physical addresses of user requests. This is because HDDs are mechanical devices whose response times are determined by the seek times and the rotational delays. In contrast, the response time of SSDs is nearly linear with the request size, as elaborated in Section II-B. A smaller request size indicates a shorter response time. Hence, processing requests with larger sizes first will make the other requests wait a longer time. Therefore, giving higher priorities to requests with small sizes will significantly reduce the waiting time of the small requests in the request queues. It must be noted that if a request arrives with a high priority tag, the request will be processed immediately.

The size-based ordering module sorts user requests in each of the two request queues according to the request sizes. The request dispatcher issues the requests to the back-end SSDs according to the sorted request order from the corresponding request queue. The request size-based ordering module in Amphibian can effectively reduce the waiting time of the requests in the request queues, thus reducing the average response time of SSD-based storage systems. Fig. 5 gives a comparison of the request processing order between Noop and Amphibian. There are one 16-KB request (i.e., $R1$), which takes 200 us to complete, and three 4-KB requests (i.e., $R2$–$R4$), each taking 50 us to complete. These four requests arrive at nearly the same time with $R1$ being slightly earlier. In accordance with the first come first served policy that Noop uses, the request service order is $R1$–$R4$. As shown in Fig. 5(a), the total service time is 1100 us and the average response time of Noop is 275 us. With the request size-based ordering scheme, Amphibian sorts the requests based on the request sizes, resulting in a request service order of $R2$, $R3$, $R4$, and $R1$. As a result, the total service time is 750 us and the average response time is 150 us, as shown in Fig. 5(b). Thus, Amphibian reduces the average response time by 45.5%, compared with the Noop-based storage system.

Prioritizing requests with small sizes reduces the average response time of the SSD-based storage system. However, similar to write starvation, Amphibian may cause requests with large sizes wait too long. In order to avoid the starvation of large requests, Amphibian sets different timeout thresholds for the read and write requests, respectively. When the waiting time of the requests in a queue reaches the preset timeout threshold, Amphibian processes the corresponding request immediately. By default, the threshold is 5000 ms for write requests and 500 ms for read requests which is similar to that in the deadline I/O scheduler. In determining which queue to be processed first, Amphibian also follows the read priority strategy to process the requests from the read queue first.

E. Garbage Collection-Aware Dispatcher

A flash chip in an SSD can be in either the normal state or the GC state. Apart from read and write operations, GC operations also significantly affect the SSD performance. When a flash chip is in the GC state, requests issued to the flash chip must wait until the GC process completes. Thus, the service times for such requests are extremely high [7], [32]. Fig. 6 shows the microscopic analysis of the average response times of three realistic traces on an Intel DCS3700 200-GB SSD. The details of the three traces are described in Section IV. Initially, the SSD is filled with the written data. It is easy to see periodic, frequent high latencies occurring due to the GC operations. These high latencies are orders of magnitude higher than those in the normal state [8]. Prior studies demonstrate similar findings [23], showing that GC can render the SSD performance significantly and in a variable and unpredictable way. The Solid State Storage Initiative of SNIA has initiated a project named "Understanding SSD Performance Project" [33], and has found that the response times are increasingly dramatic due to the GC operations. All these studies have revealed that GC processing has a significantly impact on the system performance.

In order to avoid issuing GC-conflicted requests, the GC-aware request dispatcher module in Amphibian monitors and identifies flash chips within the SSD that are in the GC state. To observe the significant performance degradation due to GC operations, the GC-aware request dispatcher continuously monitors the response times of each flash chip. When the response time of a flash chip is abnormally increased, the GC-aware request dispatcher temporarily blocks the user I/O requests to that flash chip. Then, the GC-aware request dispatcher module issues three new read requests to that flash chip. Based on the response times of the three new read requests, the GC-aware request dispatcher module can identify whether the flash chip is in the GC state.

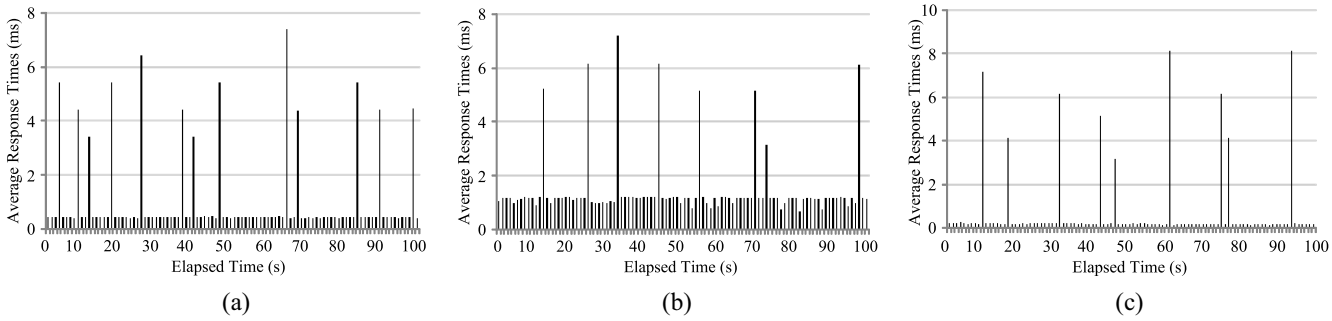

Fig. 6. Microscopic analysis of the average response times driven by the realistic traces. (a) Financial1. (b) Prn_0. (c) Prxy_0.

For new generation SSDs, GC operations are explicitly exposed to upper file systems via ioctl APIs, such as the TRIM command. TRIM is beneficial to all SSDs regardless of what kind of GC is used [34]. The TRIM command enables an operating system to notify SSD which pages no longer contain valid data. It enables SSD to handle GC operations more efficiently. For a file delete operation, the operating system marks the file's data pages as invalid pages, and then sends a TRIM command to the SSD to perform the GC operations on these invalid pages to release free space for subsequent write data.

Moreover, because flash controllers always act like a black box to host systems, a host controlled GC scheme is proposed for flash-based SSDs, such as open channel SSDs [35]. As a result, our proposed GC-aware request dispatcher can explicitly utilize these commands to identify flash chips processing GC operations. In our current implementation, Amphibian identifies GC operations based on request response times of a chip. Moreover, similar to ParDispatcher [28], Amphibian uses the space zone to mark the GC area since the internal mapping is unknown to the host system. Based on the request's address, the corresponding space zone is marked as GC-active state. If the LBAs of the requests fall within the space zone, these I/O requests will be affected. Moreover, retry operations are performed to check whether the marked GC area is still in GC state. The interval time between the retry operations is 1 s. If the requested data in the retry operations can be returned normally, the space zone is marked as normal state.

After identifying a flash chip or space zone within the SSD in the GC state, the GC-aware request dispatcher will keep all the requests targeting the identified flash chip or space zone in awaiting queue. In the meantime, Amphibian still dispatches requests to other flash chips that are in the normal state. These requests are processed in parallel on different flash chips or space zones, utilizing the saved issue bandwidth from delaying GC-conflicted requests. Consequently, they can be completed more quickly, and the throughput of the whole system is improved. Moreover, since the requests targeting the GC-active flash chips or space zones are queued in a host waiting queue, the SSD outstanding I/O feature is not affected by these pending requests. As an illustrative example, Fig. 7 shows an SSD device with an issue bandwidth of four for outstanding IO requests. Without the GC-aware request dispatcher module, the actual issue bandwidth will be reduced to two due to the other two requests (shown in grey) targeting a

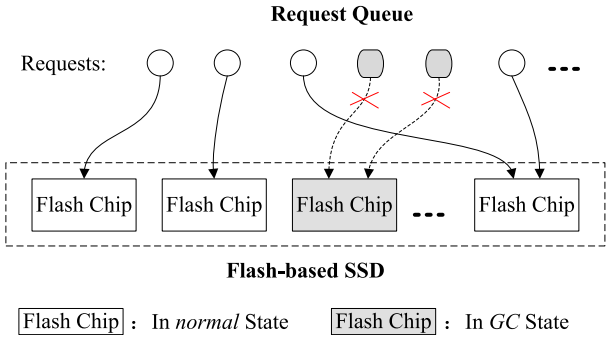

Fig. 7. Example of GC-aware request dispatching workflow in Amphibian.

flash chip in the GC state and thus being blocked. However, with the assistance from our proposed module, the I/O scheduler can identify that the target flash chip is in the GC state. Hence, the I/O scheduler can keep the two blocked requests in the waiting queue, and issue two other requests that target flash chips in normal state. In this way, the outstanding and parallel features of the SSDs are fully exploited to improve system performance and efficiency.

IV. Performance Evaluations

In this section, we first describe the experimental setup and evaluation methodology. Then we evaluate the performance of Amphibian through both benchmark and trace-driven experiments.

A. Experimental Setup and Methodology

We have implemented an Amphibian prototype as an independent module on top of the Linux I/O scheduler framework. The performance evaluation is conducted on a Dell PowerEdge T320 server with an Intel Xeon E5-2407 processor and 8-GB DDR memory. In order to examine the efficiency of Amphibian, three different types of SSDs, including two enterprise SATA SSDs and a high end PCI-e-based SSD, are evaluated in our experiments. The first one is an Intel X25-E Extreme SATA SSD 64 GB (for short, Intel X25-E SSD). The second one is an Intel DC S3700 SSD 400GB (for short, Intel DC S3700 SSD). These two SATA SSDs are connected to the PERC H710 SATA controller, representing SSDs used in the enterprise and datacenter environments. The third SSD device is an Intel 750 Series 400 GB PCI-e add-in card driver

TABLE II
EXPERIMENTAL SETUP

Machine	Intel Xeon E5-2407, 8GB RAM
OS	Linux 2.6.38
Device adapter	PERC H710 SATA controller
SSD module	Intel X25-E 64GB (X25-E) Intel DC S3700 400GB (DC S3700) Intel 750 Series PCI-e 400GB (750S PCI-e)
Benchmark tool	Postmark [36]
Trace replay tool	RAIDmeter [9]

TABLE III
WORKLOAD CHARACTERISTICS

Traces	Read Ratio	IOPS	Av. R. Size	Period
Fin1	32.8%	52	11.9KB	1h
Fin2	82.4%	127	6.2KB	1h
Web1	99.9%	335	8.0KB	1h
Web2	99.9%	331	15.1KB	1h
Web3	99.9%	218	11.8KB	1h
Proj_2	77.5%	81	40.2KB	2.4h
Prxy_1	65.3%	266	12.5KB	2.4h
Usr_1	85.5%	64	45.4KB	2.4h

(for short, Intel 750S PCI-e SSD) that is directly attached in the PCIe Gen3 slot. A separate hard disk is used to house the operating system (Linux kernel 2.6.38) and other software. The experimental setup is outlined in Table II.

Two trace-replay methods are generally used for performance evaluations: 1) open-loop and 2) closed-loop [37], [38]. The former has the potential to overestimate the user response time since the I/O arrival rate is independent of the underlying system and can therefore cause the request queue (and also the queueing delays) to grow rapidly when the system load is high. The opposite is true for the closed-loop method as the I/O arrival rate is dictated by the processing speed of the underlying system and the request queue is generally limited in length (i.e., equal to the number of the independent request threads).

To have a fair comparison, we use both an open-loop model (i.e., trace replay with RAIDmeter [9]) and a closed-loop model (i.e., Postmark benchmark [36]) to evaluate the performance of Amphibian [37], [38]. RAIDmeter [9] is a block-level trace replay software, capable of replaying traces and evaluating the I/O response time of storage devices. The traces used in our experiments are obtained from the Storage Performance Council [39] and Microsoft Research Cambridge [40]. Among them, two financial traces (Fin1 and Fin2) were collected from OLTP applications running at a large financial institution; three Web traces (Web1, Web2, and Web3) were collected from a machine running a Web search engine; and three enterprise traces (Proj_2, Prxy_1, and Usr_1) were collected from storage volumes in an enterprise data center by Microsoft Research Cambridge. These traces represent different access patterns in terms of read/write ratios, IOPS, and average request sizes. The workload parameters of these traces are summarized in Table III. Since these traces are collected from HDD-based storage systems, the I/O intensity is not enough to generate long I/O queue on flash-based storage systems. All the traces are scaled up by ten times and we replayed them the same for all the evaluated schemes.

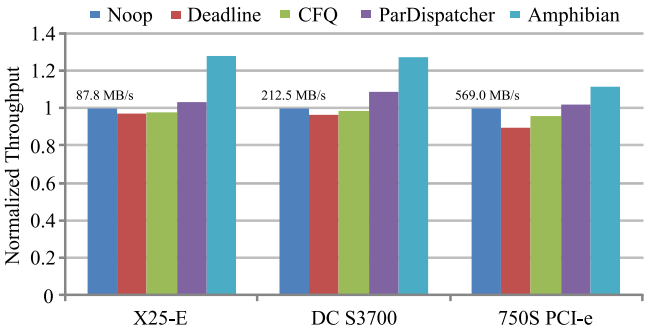

Fig. 8. Throughput results of the postmark benchmarks on the three Intel SSDs, normalized to that of the Noop scheme with the throughput values marked in the column.

In this paper, we compare the performance of Amphibian with three I/O schedulers employed by the Linux operating systems, i.e., Noop, deadline and CFQ, and the ParDispatcher scheduler [28], which leverages the parallelism by dispatching the requests to different logical space regions in an SSD. The I/O scheduler selection is configured by the command "*echo SCHEDULER-NAME > /sys/block/DEVICE-NAME/queue/scheduler*" where SCHEDULER-NAME is the I/O scheduler and DEVICE-NAME is the SSD device. In order to make a fair comparison among different schemes, the SSDs are filled with data before each experiment. In this way, the initial state of SSDs could be nearly the same for all the schemes.

B. Benchmark-Driven Evaluations

The first experiment is conducted using the Postmark benchmark [36]. It is designed to portray the performance of desktop applications, such as electronic mail, netnews, and Web-based commerce. We use the Postmark benchmark to generate an initial pool of random text files and image files ranging from a lower bound of 1 KB to an upper bound of 4 MB, then perform 100 000 transactions that include file read, file write, create and delete operations. The evaluation results of the Postmark benchmark are presented in Fig. 8. The x-axis refers to the different types of SSDs, and the y-axis is the throughput results normalized to the Noop scheme. The default queue length is set to 16.

First, we can see that the Noop scheduler achieves the best throughput among the three default Linux I/O schedulers. The reason is that the CFQ and deadline schedulers are designed specially for HDDs, and they are not suitable for SSDs with device characteristics different than HDDs. Therefore, the extra optimizations of the CFQ and deadline schedulers over the Noop scheduler will degrades system performance for SSDs. The results further validate that the HDD-optimized I/O schedulers are not suitable for SSDs. Second, Amphibian improves the throughput by 28.2%, 27.9%, and 11.6% over the Noop scheme, and by 23.8%, 17.5%, and 9.0% over the ParDispatcher scheme for the three Intel SSDs in analysis, respectively. This is because Amphibian is an SSD-optimized I/O scheduler taking both workload and SSD device characteristics into design consideration. Amphibian adopts the size-based ordering design, which is much more effective

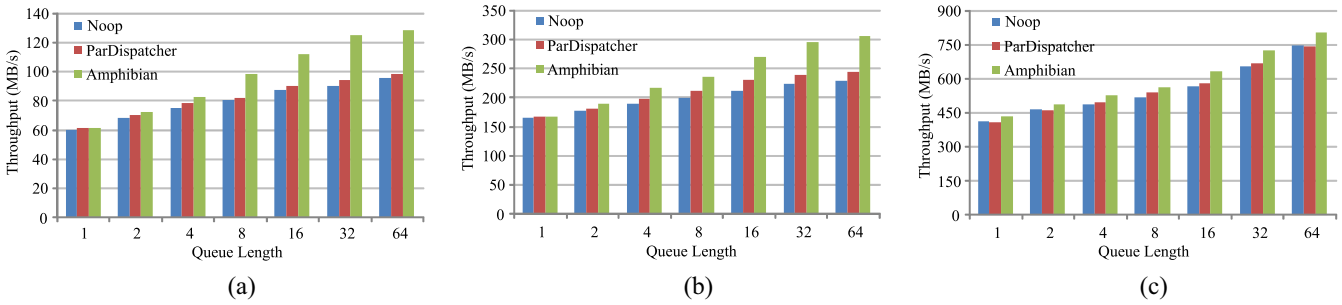

Fig. 9. Throughput results for postmark on the three Intel SSDs with respect to different queue lengths. (a) Intel X25-E SSD. (b) Intel DC S3700 SSD. (c) Intel 750S PCI-e SSD.

than the ParDispatcher and Noop schedulers, to prioritize requests with small sizes and thereby reduce the average service time. Moreover, Amphibian monitors the internal GC activities within flash chips, and avoids issuing user requests conflicting with the GC operations. Third, we can see that Amphibian consistently works well across different types of SSDs. It achieves the highest throughput in all the three SSDs examined. The Amphibian design takes into account the external requests characteristics and internal device parallelism, and is applicable to all types of flash-based SSDs. The performance results further validate the applicability design objective of Amphibian.

To further understand the reasoning behind the improvement of Amphibian, we conduct a sensitivity study on different request queue lengths for the Postmark benchmark among the Noop, ParDispatcher, and Amphibian schemes. Fig. 9 shows the comparison results on the three SSDs. In this figure, we can see that with short queue lengths, such as 1 and 2, the three schemes perform almost the same. The potential performance benefit that can be achieved by exploiting the internal parallelism within flash chips is limited in short request queues. Similarly, the performance benefit of the GC-aware dispatcher is also limited because of infrequent GC operations in short queues. However, flash-based SSDs are usually deployed in enterprise and high-end computing environments where I/O accesses are intensive [41], [42]. We can see from Fig. 9 that, as the queue length increases, the improvement of Amphibian becomes more significant. A longer request queue can hold more user requests, leading to more room for the size-based ordering and also more frequent GC operations. Consequently, the performance benefit of Amphibian is more obvious with the increasing request queue length. When the queue length is 64, Amphibian improves the throughput by up to 35.9%, 33.1%, and 11.6%, compared to the Noop scheme on the three SSDs. Furthermore, this performance improvement trend can be seen in all three Intel SSDs, again validating the high portability of Amphibian.

C. Trace-Driven Evaluations

In addition to the benchmark-driven evaluations, we also conduct trace-driven evaluations on the three types of SSDs that have different performance characteristics. In the trace-driven evaluations, we collect the average response time as the performance metric for different I/O schedulers. Fig. 10 shows the average response times of different traces on the Intel X25-E SATA SSD; all results are normalized to that of the Noop scheme. We can see that, compared with the Noop scheme, Amphibian reduces the average response time by 12.4%, 6.4%, 23.5%, 28.2%, 25.0%, 33.3%, 28.2%, and 31.6% for the Fin1, Fin2, Web1, Web2, Web3, Proj_2, Prxy_1, and Usr_1 traces, respectively. On average, Amphibian outperforms the Noop scheme with an improvement of 22.9%. Compared to the ParDispatcher scheme, Amphibian reduces the average response times by 6.3%, 7.8%, 22.1%, 23.5%, 23.8%, 15.7%, 19.6%, and 10.6% for the evaluated traces, respectively. On average, Amphibian outperforms the ParDispatcher scheme by 16.4%.

Different traces have different read/write ratios and I/O intensity. Among these eight traces, Fin1, Fin2, Proj_2, Prxy_1, and Usr_1 are read/write mixed applications where the number of write requests is comparable with or larger than the number of read requests. Since processing write requests and internal GC operations takes longer than processing read requests, the read requests in these five traces need to wait a relatively longer time. In such cases, employing Amphibian reduces the waiting time of the read requests by giving them higher priorities and avoiding issuing requests to the flash chips in the GC state. Moreover, in both request queues, Amphibian processes small requests prior to large requests. Thus, it can further reduce the average response times by reducing the waiting time in the request queues. For read-intensive workloads, such as Web1, Web2, and Web3, Amphibian reduces the request response times using the size-based ordering scheme, thus reducing the queueing times of the small requests. Therefore, Amphibian is effective in reducing the request response times for both read-intensive and mixed read/write applications, thus being applicable to various applications.

The three types of SSDs we used have different performance characteristics, thus the performance improvements of Amphibian conducted on the three SSDs are also different. Fig. 11 shows the average response times results on the Intel DC S3700 SATA SSD driven by the different traces, normalized to that of the Noop scheme. Compared with the Noop scheme, Amphibian reduces the request response times by 17.6%, 8.7%, 16.3%, 13.6%, 19.0%, 35.1%, 17.6%, and 38.9% for the Fin1, Fin2, Web1, Web2, Web3, Proj_2, Prxy_1, and Usr_1 traces, respectively. On average, Amphibian outperforms the Noop scheme with an improvement of 20.1%.

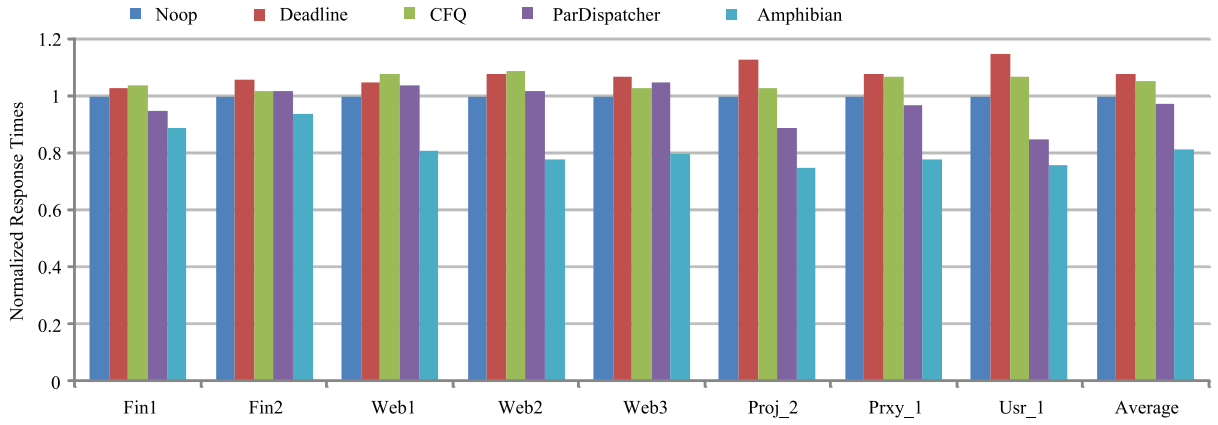

Fig. 10. Normalized average response times on the Intel X25-E SATA SSD driven by the different traces.

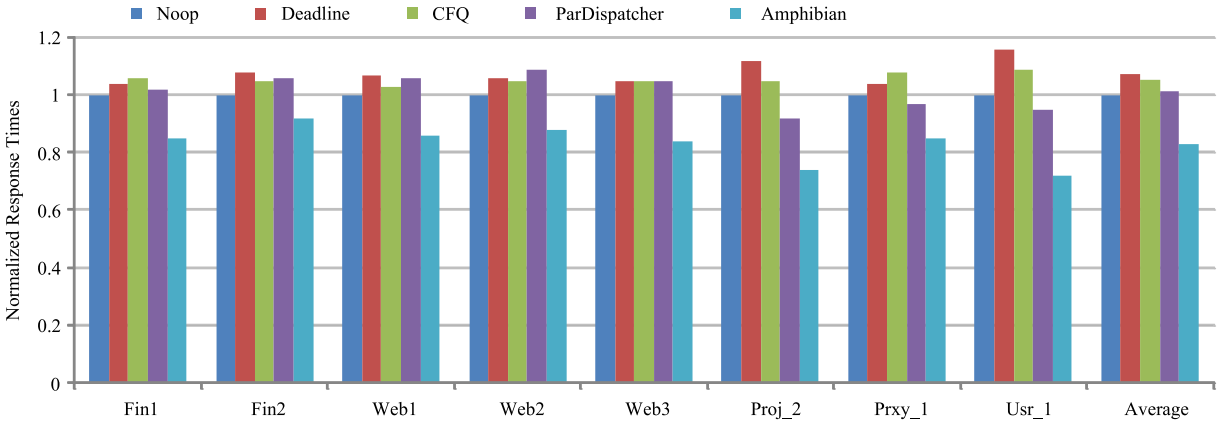

Fig. 11. Normalized average response times on the Intel DC S3700 SATA SSD driven by the different traces.

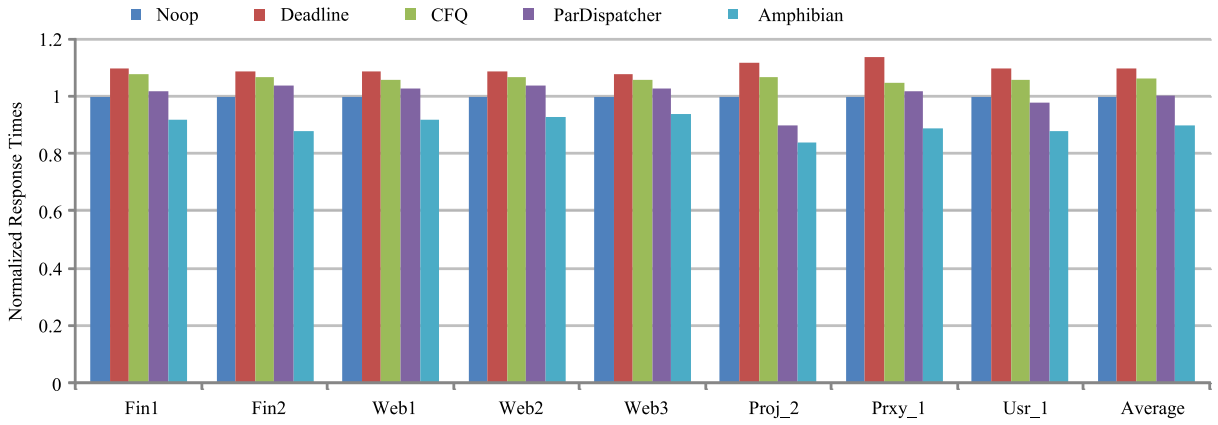

Fig. 12. Normalized average response times on the Intel 750S PCI-e SSD driven by the different traces.

Compared with the ParDispatcher scheme, Amphibian reduces the request response times by 16.7%, 13.2%, 18.9%, 19.3%, 20.1%, 19.6%, 12.4%, and 24.2% for the Fin1, Fin2, Web1, Web2, Web3, Proj_2, Prxy_1, and Usr_1 traces, respectively. On average, Amphibian outperforms the ParDispatcher scheme with an improvement of 18.0%.

Fig. 12 shows the average response times results on the Intel 750S PCI-e SSD driven by the different traces, normalized to that of the Noop scheme. Compared with the Noop scheme, Amphibian reduces the request response times by 8.7%, 13.4%, 8.7%, 7.5%, 6.4%, 19.0%, 12.4%, and 13.6% for the Fin1, Fin2, Web1, Web2, Web3, Proj_2, Prxy_1, and Usr_1 traces, respectively. On average, Amphibian outperforms the Noop scheme with an improvement of 11.1%. Compared with the ParDispatcher scheme, Amphibian reduces the request response times by 9.8%, 15.2%, 10.7%, 10.6%, 8.7%, 6.7%, 12.7%, and 10.2% for the Fin1, Fin2, Web1, Web2, Web3, Proj_2, Prxy_1, and Usr_1 traces, respectively. On average, Amphibian outperforms the ParDispatcher scheme with an improvement of 10.6%.

From Figs. 10–12, we can see that for all the three types of SSDs, Amphibian largely outperforms the Noop scheme

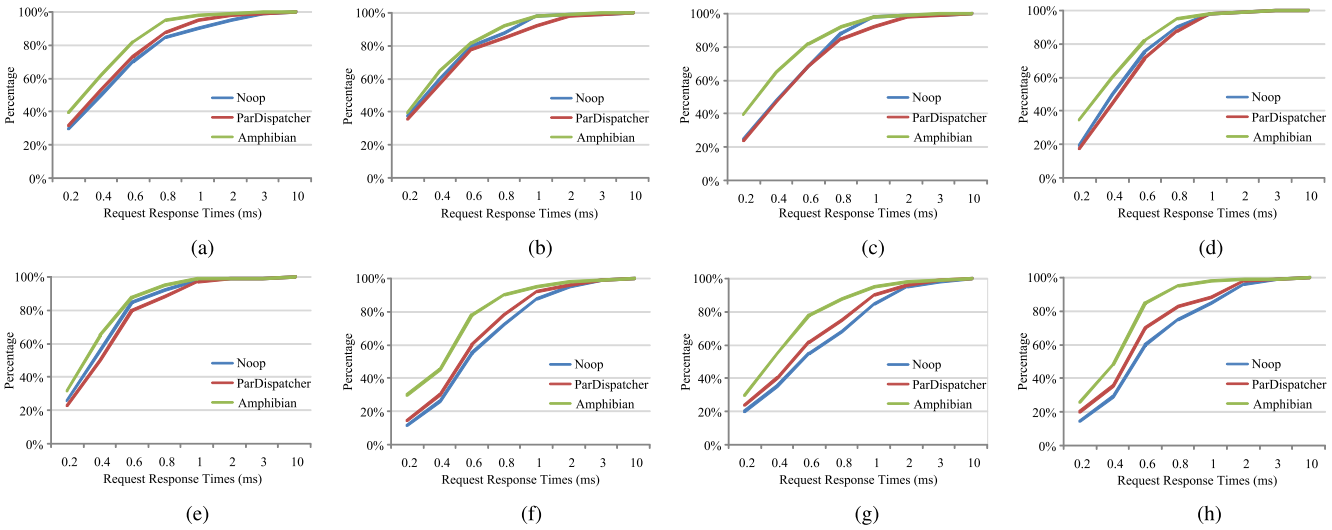

Fig. 13. Response time distributions for the Intel X25-E SATA SSD: the *x*-axis shows the request response times and the *y*-axis shows the fraction of requests with response times lower than some values in *x*-axis. (a) Fin1. (b) Fin2. (c) Web1. (d) Web2. (e) Web3. (f) Proj_2. (g) Prxy_1. (h) Usr_1.

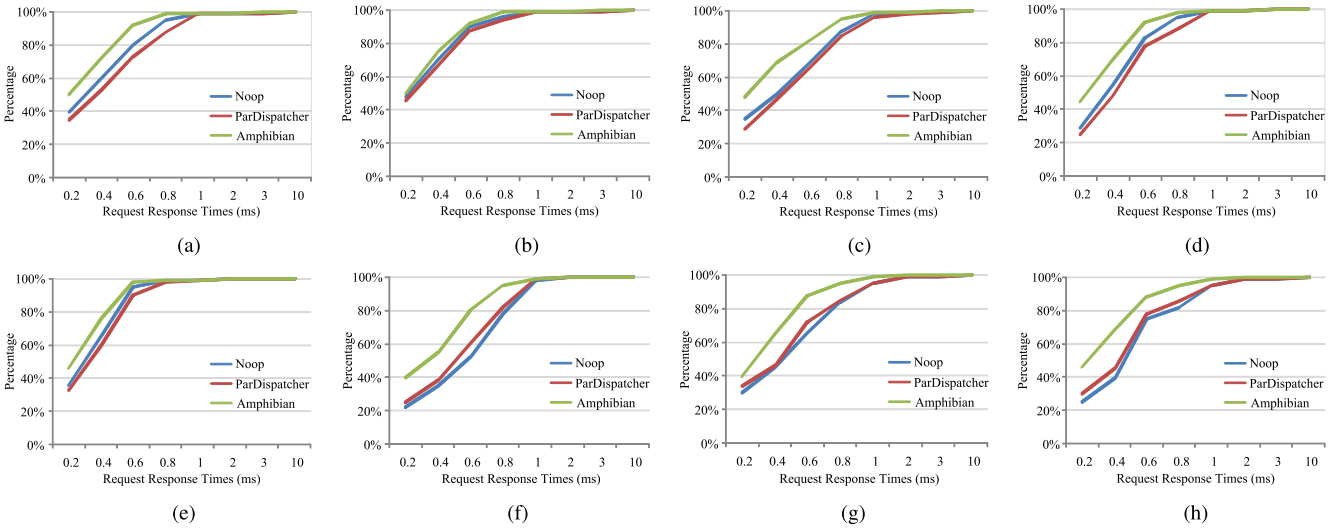

Fig. 14. Response time distributions for the Intel DC S3700 SATA SSD: the *x*-axis shows the request response times and the *y*-axis shows the fraction of requests with response times lower than some values in *x*-axis. (a) Fin1. (b) Fin2. (c) Web1. (d) Web2. (e) Web3. (f) Proj_2. (g) Prxy_1. (h) Usr_1.

in user response times. The evaluation results further validate high portability of Amphibian for both enterprise SSDs and high-end PCI-e SSDs. All of these SSDs are made of flash chips, sharing similar device properties, such as asymmetric read–write performance, request size-based responsiveness, and GC-influenced performance degradation. In addition, we see that the performance improvement of Amphibian on the 750S PCI-e SSD is lower than that on the other two SATA SSDs. This is because high-end PCI-e SSDs have a large on-board buffer cache that can hide some performance issues of the flash chips [18], [43]. More specifically, we observe that, on the Intel 750S PCI-e SSD, the random-read and random-write performance are almost the same during the early stage of a program, but the random-write performance degrades significantly later on after the on-board buffer cache is filled up.

From the results of the trace-driven evaluations, we can also see that the HDD-based I/O schedulers, such as CFQ and deadline, are not suitable for flash-based SSDs. The average response times of the CFQ and deadline schemes are much larger than that of the Noop scheme. The results are consistent with the results of the benchmark-driven evaluations, further validating that the HDD-optimized I/O schedulers are not suitable for flash-based SSDs. With flash-based SSDs being popular and widely deployed in I/O-intensive servers, such as enterprise and HPC environments, the software stack of the storage systems should be redesigned from optimizing the performance of HDDs to optimizing the performance of flash-based SSDs. Hence, the performance optimizations adopted in Amphibian is highly critical to exploit the performance advantages of flash-based SSDs, as opposed to the traditional I/O schedulers.

To further investigate the performance improvement of Amphibian in the examined traces, we plot the distributions of the request response times in Figs. 13–15 for the three SSDs, respectively. From these figures, we can see that Amphibian

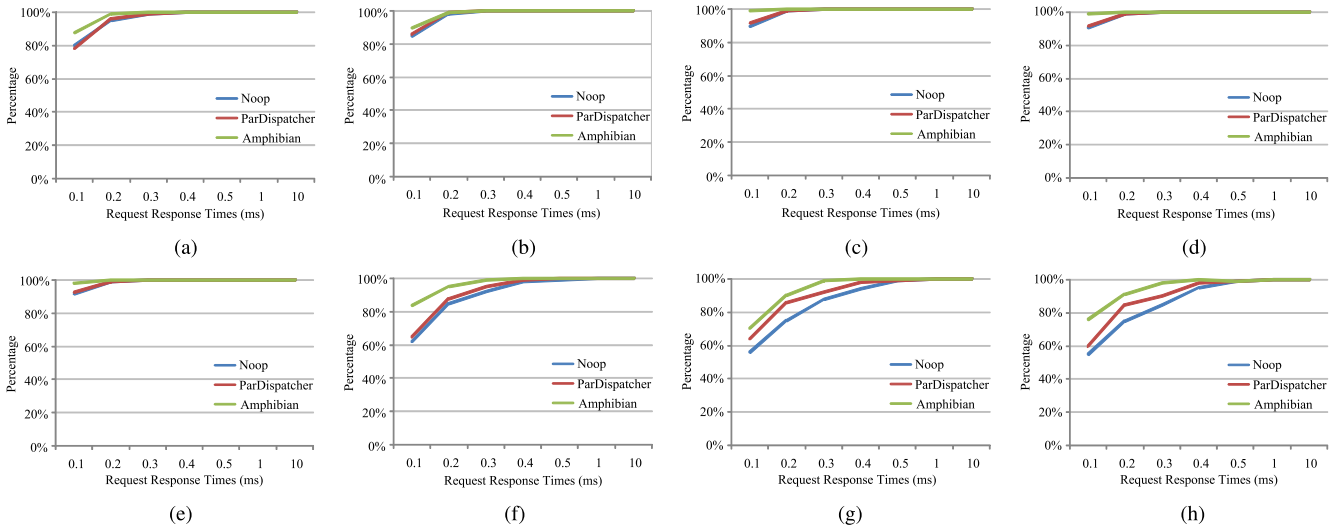

Fig. 15. Response time distributions for Intel 750S PCI-e SSD: the *x*-axis shows the request response times, and the *y*-axis shows the fraction of requests with response times lower than some values in *x*-axis. (a) Fin1. (b) Fin2. (c) Web1. (d) Web2. (e) Web3. (f) Proj_2. (g) Prxy_1. (h) Usr_1.

can process over 90% of the user requests in less than 1 ms in the two SATA SSDs and can process nearly 99% of the user requests in less than 0.2 ms for the two financial and three Web traces in the 750S PCI-e SSD. It is apparent that Amphibian performs much better than conventional schedules designed for HDD-based storage systems. The reason is that flash-based SSDs are made of flash chips and do not have mechanical parts, thus providing better responsiveness than HDDs. The response times on the PCI-e SSD are much lower than the SATA SSDs because the high-end PCI-e SSD has a large internal on-board buffer cache and some reserved flash space.

For all the three types of SSDs, a higher percentage of user requests are processed in Amphibian with low response times than the other two schemes. This also validates our observation that Amphibian achieves lower average response times than the Noop and ParDispatcher schemes in these traces. The reasons are threefold. First, by prioritizing read requests with small sizes, the average request waiting time in the I/O queue is reduced. Second, by processing small requests prior to large requests, the small requests can be processed much more effectively, thus further reducing the long waiting times of the small requests. Third, since Amphibian avoids issuing the user requests to the flash chips that are in the GC state, the user requests can be processed without the contention with the GC-induced internal requests. In this way, the user requests can be processed much more efficiently. Therefore, the request response times achieved by Amphibian are significantly lower than that achieved by the Noop and ParDispatcher schemes.

V. Related Work

Many studies have been conducted on I/O scheduling for HDD-based storage systems. HDD-based I/O schedulers, such as NOOP, deadline, and CFQ, have become standards in Linux kernels. However, none of them can work efficiently with SSDs, as validated by the previous studies [24], [28], [29], [31], [44] and our performance evaluations.

To address the problem, some studies have been conducted to improve the efficiency of I/O schedulers for SSDs. These studies can be classified into two categories. The first category is fairness-oriented I/O schedulers. For instance, Shen and Park [24], [29] proposed the FIOS and FlashFQ algorithms that take the fairness of the resource usage in SSDs into account. FIOS [29] schedules requests with the awareness of the read and write interference of SSDs, while FlashFQ [24] schedules requests with the awareness of the internal parallelism for different applications. FlashFQ relies on the request size to predict the response time, thus arranging the start-time fair queueing efficiently to improve the fairness among multiple concurrent tasks. In their evaluations, they only considered fixed request sizes, such as 4 or 128 KB; in contrast, real applications used mix of different request sizes and types. Moreover, FlashFQ exploits the internal parallelism to issue multiple concurrent I/O requests and avoids the unfairness resulting from the interference of the concurrent dispatched requests.

In contrast, the objective of Amphibian is high performance as opposed to fairness in FlashFQ. The different design objectives also lead to different design considerations. Different from FlashFQ, Amphibian exploits both request size and internal parallelism characteristics for the SSD I/O scheduler. Amphibian reorders the I/O requests based on the request sizes to reduce the long waiting times for the small-size requests, thus improving the overall system responsiveness. Moreover, Amphibian exploits the internal parallelism for the purpose of avoiding issuing the GC-conflicted requests on the flash chips that are in the GC-state. By reducing such GC-conflicted requests on the I/O queue, the overall system performance is improved.

The second category is performance-oriented I/O schedulers that aim at finding the maximized possibility of using the internal characteristics of SSDs in the block layer. For example, Wang *et al.* [28] proposed ParDispatcher that takes advantage of the internal parallelism of SSD. However, ParDispatcher only considers the logical space region, and

TABLE IV
COMPARISONS BETWEEN AMPHIBIAN AND THE STATE-OF-THE-ART

Schemes	**Request Type**	**Request Size**	**GC awareness**
FIOS [29]	√		
FlashFQ [24]	√		
ParDispatcher [28]	√		
Amphibian	√	√	√

cannot fully exploit the parallelism of flash-based SSDs that use the out-of-place update scheme. Thus, its performance improvement is limited, as shown in our performance evaluations. Gao *et al.* [25] proposed PIQ to minimize the access conflicts among the I/O requests in one batch by exploiting the rich parallelism of SSDs. Besides the above studies, Dunn and Reddy [44] proposed a new scheduler to avoid the penalty that is created during the new block writing. Kim *et al.* [31] proposed IRBW-FIFO and IRBW-FIFO-RP, which arrange the write-request into a logical block size bundle to improve the write performance.

However, none of these works are proposed to solve the request size ordering and internal GC conflict problems, as shown in Table IV. Our proposed Amphibian exploits the two key characteristics, i.e., the request size and the internal parallelism, to improve the I/O performance of the SSD-based storage system. Moreover, the fairness and performance objectives are orthogonal to each other in the design of the I/O scheduler for SSDs. In a concurrent system with multiple tasks, the fairness-oriented I/O schedulers, such as FlashFQ, achieve the fairness among tasks. In each task, the performance-oriented I/O schedulers, such as Amphibian, reduce the waiting times and response times of the I/O requests. Thus, the size-based ordering and GC-aware dispatching schemes in Amphibian are applicable to the I/O requests in each task in the FlashFQ scheme. Thus, Amphibian is orthogonal to and can be easily incorporated into any existing I/O scheduler algorithms.

On the other hand, the internal parallelism in SSDs is one of the important characteristics and is different from HDDs. A lot of studies have been conducted to exploit the internal parallelism for performance improvement [1], [3], [18], [19], [25], [28], [45]. Chen *et al.* [18] first conducted extensive experiments to show that the parallelism of SSDs is very important for the performance improvement. By exploiting the internal parallelism, their evaluation results show that the performance of the write operations has no relationship with their access patterns and is better than that of read operations. Hu *et al.* [19] studied the four level parallelism in SSDs and found that these four levels have different priorities in the exploration of the access latency and the system throughput. Compared with these studies, Amphibian exploits the internal parallelism by avoiding issuing the GC-conflicted requests to the flash chips in the GC state, thus reducing the long waiting times to improve the system performance.

VI. CONCLUSION

As the processor and memory speeds have increased over the HDDs, the I/O access latency has become a bottleneck of the system performance. Flash-based SSDs are promising in reducing the access latency in high performance computing environments and data centers. In this paper, we propose a new I/O scheduler for SSDs, called Amphibian, to exploit both high-level request characteristics and low-level internal feature of the flash chips and improve the performance of SSD-based storage systems. In our proposed Amphibian scheme, the size-based request ordering gives higher priorities to requests with small sizes; and the GC-aware request dispatching avoids issuing requests to the flash chips that are in the GC state. As a result, the average waiting times of the requests are reduced significantly. The extensive evaluation results show that Amphibian greatly outperforms the existing I/O schedulers in both throughput and request response times. Consequently, the I/O performance of SSD-based storage systems is improved.

We plan to explore several directions for the future work. First, we will take the fairness metric into account in the Amphibian design and conduct experiments to compare the efficiency of Amphibian and fairness-oriented I/O schedulers, such as FIOS and FlashFQ. Second, we will extend Amphibian and evaluate its efficiency on SSD-based disk arrays [46]. By treating an SSD in an SSD-based disk array as a flash chip in an SSD, the performance variability problem due to the GC operations remains the same.

ACKNOWLEDGMENT

The authors would like to thank all the ASTL members for their continues discussion and support on this paper.

REFERENCES

[1] N. Agrawal *et al.*, "Design tradeoffs for SSD performance," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, Boston, MA, USA, Jun. 2008, pp. 57–70.
[2] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding intrinsic characteristics and system implications of flash memory based solid state drives," in *Proc. Joint Int. Conf. Meas. Model. Comput. Syst. (SIGMETRICS/Performance)*, Seattle, WA, USA, Jun. 2009, pp. 181–192.
[3] C. Dirik and B. Jacob, "The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization," in *Proc. 36th Int. Symp. Comput. Archit. (ISCA)*, Austin, TX, USA, Jun. 2009, pp. 279–289.
[4] 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.
[5] C. Ruemmler and J. Wilkes, "An introduction to disk drive modeling," *IEEE Comput.*, vol. 27, no. 3, pp. 17–28, Mar. 1994.
[6] M. Jung, R. Prabhakar, and M. T. Kandemir, "Taking garbage collection overheads off the critical path in SSDs," in *Proc. ACM/IFIP/USENIX 13th Int. Conf. Middleware (Middleware)*, Montreal, QC, Canada, Dec. 2012, pp. 164–186.
[7] G. Wu and X. 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, pp. 117–124.
[8] 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.
[9] B. Mao *et al.*, "HPDA: A hybrid parity-based disk array for enhanced performance and reliability," *ACM Trans. Storage*, vol. 8, no. 1, pp. 1–22, 2012.
[10] L. M. Grupp, J. D. Davis, and S. Swanson, "The bleak future of NAND flash memory," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST)*, San Jose, CA, USA, Feb. 2012, pp. 17–24.
[11] L. M. Grupp, J. D. Davis, and S. Swanson, "The Harey tortoise: Managing heterogeneous write performance in SSDs," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, San Jose, CA, USA, Jun. 2013, pp. 79–90.

[12] C. Min, K. Kim, H. Cho, S. Lee, and Y. Eom, "SFS: Random write considered harmful in solid state drives," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST)*, San Jose, CA, USA, Feb. 2012, pp. 139–154.
[13] X. Zhang, J. Li, H. Wang, K. Zhao, and T. Zhang, "Reducing solid-state storage device write stress through opportunistic in-place delta compression," in *Proc. 14th USENIX Conf. File Storage Technol. (FAST)*, Santa Clara, CA, USA, Feb. 2016, pp. 111–124.
[14] J. Lee, Y. Kim, G. M. Shipman, S. Oral, and J. Kim, "Preemptible I/O scheduling of garbage collection for solid state drives," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 2, pp. 247–260, Feb. 2013.
[15] (2013). *IOmeter Project*. [Online]. Available: http://www.iometer.org/
[16] 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. 1–21, 2016.
[17] A. R. Abdurrab, T. Xie, and W. Wang, "DLOOP: A flash translation layer exploiting plane-level parallelism," in *Proc. 27th IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, Boston, MA, USA, May 2013, pp. 908–918.
[18] F. Chen, R. Lee, and X. Zhang, "Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing," in *Proc. 17th Int. Conf. High-Perform. Comput. Archit. (HPCA)*, San Antonio, TX, USA, Feb. 2011, pp. 266–277.
[19] Y. Hu *et al.*, "Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity," in *Proc. 25th Int. Conf. Supercomput. (ICS)*, Tucson, AZ, USA, Jun. 2011, pp. 96–107.
[20] M. Jung and M. T. Kandemir, "Sprinkler: Maximizing resource utilization in many-chip solid state disks," in *Proc. 20th IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Orlando, FL, USA, Feb. 2014, pp. 524–535.
[21] M. Jung and M. Kandemir, "Revisiting widely held SSD expectations and rethinking system-level implications," in *Proc. ACM SIGMETRICS/Int. Conf. Meas. Model. Comput. Syst. (SIGMETRICS)*, Pittsburgh, PA, USA, Jun. 2013, pp. 203–216.
[22] Y. Lu, J. Shu, and W. Wang, "ReconFS: A reconstructable file system on flash storage," in *Proc. 12th USENIX Conf. File Storage Technol. (FAST)*, Santa Clara, CA, USA, Feb. 2014, pp. 75–88.
[23] J. Lee *et al.*, "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.
[24] K. Shen and S. Park, "FlashFQ: A fair queueing I/O scheduler for flash-based SSDs," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, San Jose, CA, USA, Jun. 2013, pp. 67–78.
[25] C. Gao *et al.*, "Exploiting parallelism in I/O Scheduling for access conflict minimization in flash-based solid state drives," in *Proc. 30th Int. Conf. Massive Storage Syst. Technol. (MSST)*, Santa Clara, CA, USA, Jun. 2014, pp. 1–11.
[26] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. T. Kandemir, "HIOS: A host interface I/O scheduler for solid state disks," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Minneapolis, MN, USA, Jun. 2014, pp. 289–300.
[27] S.-Y. Park, E. Seo, J.-Y. Shin, S. Maeng, and J. Lee, "Exploiting internal parallelism of flash-based SSDs," *IEEE Comput. Archit. Lett.*, vol. 9, no. 1, pp. 9–12, Jan. 2010.
[28] H. Wang *et al.*, "A novel I/O scheduler for SSD with improved performance and lifetime," in *Proc. 29th IEEE Symp. Massive Storage Syst. Technol. (MSST)*, Long Beach, CA, USA, May 2013, pp. 1–5.
[29] 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, pp. 155–170.
[30] 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)*, Santa Clara, CA, USA, Feb. 2016, pp. 125–132.
[31] J. Kim *et al.*, "Disk schedulers for solid state drivers," in *Proc. Int. Conf. Embedded Softw. (EMSOFT)*, Grenoble, France, Oct. 2009, pp. 295–304.
[32] 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)*, Nantes, France, Feb. 2017, pp. 15–28.
[33] (Apr. 2017). *Understanding SSD Performance Project of SNIA*. [Online]. Available: http://www.snia.org/forums/sssi/pts
[34] H. Yeom, "From black box to grey box: Is it feasible for flash?" in *Proc. Oper. Syst. Support Next Gener. Large Scale NVRAM (NVRAMOS)*, Jeju-do, South Korea, Oct. 2014, pp. 1–10.
[35] M. Bjørling, J. Gonzalez, and P. Bonnet, "LightNVM: The linux open-channel SSD subsystem," in *Proc. 15th USENIX Conf. File Storage Technol. (FAST)*, Santa Clara, CA, USA, Feb. 2017, pp. 359–373.
[36] (Apr. 2017). *Filesystem Benchmarking With Postmark From NetApp*. [Online]. Available: http://www.shub-internet.org/brad/FreeBSD/postmark.html
[37] M. P. Mesnier *et al.*, "//TRACE: Parallel trace replay with approximate causal events," in *Proc. 5th USENIX Conf. File Storage Technol. (FAST)*, San Francisco, CA, USA, Feb. 2007, pp. 153–167.
[38] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open versus closed: A cautionary tale," in *Proc. 3rd USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, San Jose, CA, USA, Apr. 2006, pp. 239–252.
[39] (2010). *OLTP Trace From UMass Trace Repository*. [Online]. Available: http://traces.cs.umass.edu
[40] (2008). *MSR Cambridge Traces*. [Online]. Available: http://iotta.snia.org/tracetypes/3
[41] 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 Int. Conf. Supercomput. (ICS)*, Tucson, AZ, USA, Jun. 2011, pp. 22–32.
[42] 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)*, Nuremberg, Germany, Mar. 2009, pp. 145–158.
[43] M. Jung, W. Choi, J. Shalf, and M. T. Kandemir, "Triple-A: A non-SSD based autonomic all-flash array for high performance storage systems," in *Proc. 19th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, Salt Lake City, UT, USA, Mar. 2014, pp. 441–454.
[44] M. Dunn and A. Reddy, "A new I/O scheduler for solid state devices," M.S. thesis, Comput. Eng., Texas A&M Univ., College Station, TX, USA, 2009.
[45] W. Wang and T. Xie, "PCFTL: A plane-centric flash translation layer utilizing copy-back operations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3420–3432, Dec. 2015.
[46] Y. Kim *et al.*, "Coordinating garbage collection for arrays of solid-state drives," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 888–901, Apr. 2014.

Bo Mao (S'08–M'10) 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 40 publications in international journals and conferences. His current research interests include storage system, cloud computing, and big data.

Suzhen Wu (S'09–M'10) 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 2014. She has over 40 publications in international journals and conferences. Her current research interests include computer architecture and storage system.

Lide Duan (S'09–M'14) received the bachelor's degree in computer science from Shanghai Jiao Tong University, Shanghai, China, in 2006, and the Ph.D. degree in computer engineering from Louisiana State University, Baton Rouge, LA, USA, in 2011.

He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Texas at San Antonio (UTSA), San Antonio, TX, USA. Prior to joining UTSA in 2014, he was a Senior Design Engineer with AMD, Sunnyvale, CA, USA, researching on future x86-based high performance and low power CPU microarchitecture design and performance modeling. In addition, he had an internship with Lawrence Livermore National Laboratory, Livermore, CA, USA, in 2011. His current research interests include computer architecture with a current focus on nonvolatile memory systems and energy efficiency of emerging computer architectures.