Challenges in Getting Flash Drives Closer to CPU

Myoungsoo Jung and Mahmut Kandemir
Department of CSE, The Pennsylvania State University
{mj@cse.psu.edu, kandemir@cse.psu.edu}

Abstract

The PCI Express Solid State Disks (PCIe SSDs) blur the difference between block and memory access semantic devices. Since these SSDs leverage PCIe bus as storage interface, their interfaces are different from conventional memory system interconnects as well as thin storage interfaces. This leads to a new SSD architecture and storage software stack design. Unfortunately, there are not many studies focusing on the system characteristics of these emerging PCIe SSD platforms. In this paper, we quantitatively analyze the challenges faced by PCIe SSDs in getting flash memory closer to CPU and study two representative PCIe SSD architectures (from-scratch SSD and bridge-based SSD) using state-of-the-art real SSDs from two different vendors. Our experimental analysis reveals that 1) while the from-scratch SSD approach offers remarkable performance improvements, it requires enormous host-side memory and computation resources which may not be acceptable in many computing systems; 2) the performance of the from-scratch SSD significantly degrades in a multi-core system; 3) redundant flash software and controllers should be eliminated from the bridge-based SSD architecture; and 4) latency of PCIe SSDs significantly degrade with their storage-level queueing mechanism. Finally, we discuss system implications including potential PCIe SSD applications such as all-flash array.

1 Introduction

Over the past few years, NAND Flash-based Solid State Disks (SSDs) are widely employed in various computing systems ranging from embedded systems to enterprise-scale servers to high-performance computing systems, thanks to their high performance and low power consumption. Even though SSDs were originally meant to be a block device replacement or a storage cache that works along with slow spinning disks, their performance has bumped to standard thin storage interfaces such as SATA 6Gpbs, which, at this point, blurs the difference between block and memory access semantic devices. Figure 1a plots the bandwidth trends for the thin interfaces versus various SSDs in real world. While the bandwidth of SATA interface has increased from 150MB/s to 600MB/s over a decade, SSDs have improved their bandwidth by four times during the same period. As a result of this remarkable performance improvement, both industry and academia started to consider taking SSDs out from the I/O controller hub (i.e., Southbridge) and locate them as close to the CPU side as possible (see Figure 1b). Clearly, PCIe SSDs are by far one of the easiest ways to integrate flash memory into the processor-memory complex (i.e., Northbridge), which requires no cabling or connections to other I/O devices involved in handling flash memory. By exploiting the benefits of the PCIe interface, latencies are expected to be kept as close to DRAM levels as possible. However, since these SSD technologies consider the PCIe bus as a storage interface, their interfaces are different

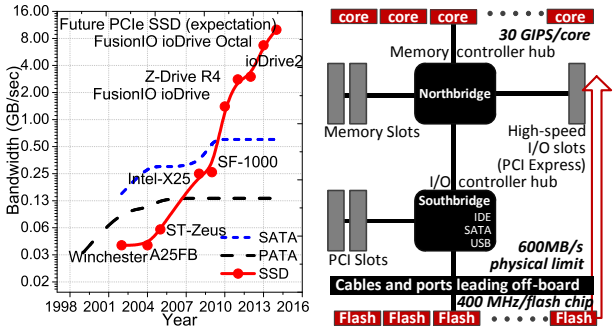


Figure 1: Bandwidth trends over time for the thin interfaces versus SSDs (a), and flash storage integration into a place closer to CPU (b).

from conventional memory interconnects at Northbridge as well as thin storage interfaces at Southbridge. The data movement management and underlying flash storage management under this new SSD interface makes PCIe SSDs a pivotal milestone in the evolution of SSD architectures and accompanying software stack designs. Unfortunately, the system characteristics of these new emerging PCIe SSD platforms has received, so far, little attention in the literature, and challenges behind these SSDs and software technologies remain largely unexplored. Further, the public datasheets of SSDs give very little information.

In this paper, we quantitatively analyze the challenges PCIe SSDs face in getting flash memory closer to the CPU side and study two representative PCIe SSD architectures and flash software stacks therein: 1) *from-scratch PCIe SSD architecture* and 2) *bridge-based PCIe SSD architecture*. The from-scratch PCIe SSD is built from bottom to top by employing FPGA- or ASIC-based native PCIe controller(s). In contrast, the bridge-based PCIe SSD leverages the conventional high-performance SSD controller(s) by employing an on-board PCIe-to-SAS (or -SATA) bridge controller. Unlike the latter, the from-scratch SSD further optimizes the flash software stack in order to maximize the storage and data processing efficiency. To characterize these two different architectures, we performed a comprehensive set of experiments using two state-of-the-art PCIe SSDs from two different vendors. To the best of our knowledge, our data analysis and presented resource management characteristics on PCIe SSDs are not reported in the literature so far and not studied well in the past. Our main contributions can be summarized as follows:

• *Characterizing the performance of the emerging PCIe SSD architectures.* We observe that the latency and throughput of the from-scratch PCIe SSD outperforms the bridge-based PCIe SSD, which is opposite to the impression one could get from the datasheets of these SSDs. Specifically, the from-scratch SSD offers on average 29% and 39% shorter latency, and provides 21% and 81% bet-

ter throughput on reads and writes, respectively. In addition, the from-scratch SSD offers stable write performance in terms of both latency and throughput under heavy write-intensive workloads, while the bridge-based SSD exhibits some sort of write cliff [12], which is a significant performance drop caused by garbage collections.

• *Analyzing host-side resource usages on different flash memory storage stacks.* Even though the from-scratch SSD offers better and sustained performance, it overly consumes host-side resources in terms of memory and computational power, which might be unacceptable in many cases. Specifically, the from-scratch SSD needs about 10 GB host-side memory space for I/O services, whereas the bridge-based PCIe SSD requires only 0.6 GB at most. In addition, it consumes 80% of CPU cycles in completing I/O requests, whereas the latter only needs 23% computation power for the same I/O services.

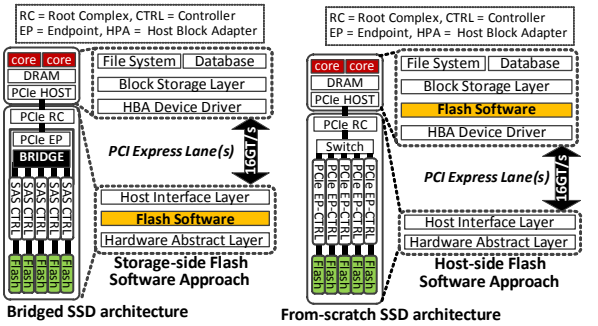
• *Addressing the challenges brought by PCIe SSDs as shared resources.* We observed that the performance of both bridged-based SSD and from-scratch SSD significantly degrades as we increase the number of I/O processing workers. While the host-side resource consumption of the bridge-based SSD is not impacted by the number of workers, the from-scratch SSD requires more host-side memory space and more CPU cycles (32% and 160%, respectively). We also found that these emerging SSDs exhibit about 100 times longer latency with a device-level queue method compared to the one with the legacy mode.

2 Bringing SSDs Closer to CPU

To bring flash drivers closer to the process-memory complex, one needs to achieve shorter latency values with higher throughput than conventional SSD devices. Because of the adapter form-factor of PCIe SSD platforms, which allows them to allocate more space in employing multiple flash packages and SSD controllers, PCIe SSDs are in the much better position to reap the benefits of higher parallelism compared to conventional SAS/SATA SSDs. In addition to high PCIe bus capacity, this advantage of parallelism enables PCIe SSDs reduce latency while increasing throughput, as compared to single SAS/SATA SSDs. Further, they also improve the performance of the flash software stack. In this section, we explain two representative PCIe SSD architectures and corresponding flash software stacks.

2.1 PCIe Architecture

Bridge-based PCIe SSD. As shown on the left side of Figure 2a, the bridge-based SSD employs multiple traditional SSD controllers, each of which handling the underlying flash packages like a single SAS/SATA SSD. These SSD controllers are also connected to a PCIe-to-SAS (and -SATA) bridge controller, which interconnects upper external PCIe link and under internal SAS link. The bridge controller internally converts the PCIe protocol to the SAS protocol (or vice versa) so that it can leverage existing SSD technologies and offer high compatibility. In addition, the bridge controller stripes the incoming I/O requests over multiple SSD controllers, which is similar to what RAID controllers do to improve storage-level parallelism. Consequently, the bridge-based SSD architecture can expose an aggregated SAS/SATA SSD performance to the PCIe root complex (RC) device, which connects the internal PCIe fabric, composed of one or multiple bridge controllers, to the processor-memory complex.



(a) Bridge-based SSD (b) From-scratch SSD

Figure 2: High-level views of our PCIe SSD architectures and their software stacks.

From-scratch PCIe SSD. One of challenges behind the bridge-based SSD architecture is the high performance overheads in internally converting different protocols and in processing I/Os, using the indirect control logic, from CPU to flash memory. Motivated by this, the from-scratch PCIe SSDs have been built from bottom to top by directly interconnecting the NAND flash interface and the external PCIe link, as shown in Figure 2b. Since PCIe is a set of point-to-point links, the connection between the PCIe RC and the flash interface is implemented by one or more switch devices, each internally handling multiple PCIe endpoints (EPs). The PCIe EP has independent upstream and downstream buffers, which control the in-bound or out-bound I/O requests in front of the flash memory. This scalable architecture can easily expand the storage capacity by putting more flash chips into its PCIe network topology and straightforwardly expose true NAND flash memory performance to the upper processor-memory subsystem. These PCIe EPs and switches are typically implemented by FPGA or ASIC as a form of native PCIe controller, and the flash software can be optimized to reduce latency and offer better throughput, as discussed below.

2.2 Flash Software Stack

Storage-side Flash Firmware. Typically, the flash control modules are implemented in the storage side for most conventional SSDs and bridge-based PCIe SSDs as "flash firmware". In this storage-side flash software stack, a hardware abstraction layer (HAL) handles low-level NAND flash commands and manages the I/O bus for moving data between SSD controller and internal registers of individual flash memories, as depicted on the right side of Figure 2a. On top of the HAL, the main flash software modules are built, which include the flash translation layer (FTL), buffer cache, wear-leveler and garbage collector. Among the flash software modules, the FTL is the core logic in managing flash memory, and translates addresses from virtual to physical. Finally, there is a host interface logic atop the flash software, which is mainly responsible for the protocol conversion, parsing requests, and scheduling them. This conventional flash software stack lets SSDs expose the underlying flash memory to the processor-memory complex without any host-side storage stack modification.

Host-side Flash Software Module. Flash software could manage the underlying flash memory more efficiently if it is possible to access the host-level resources such as file system and incoming I/O request information. Con-

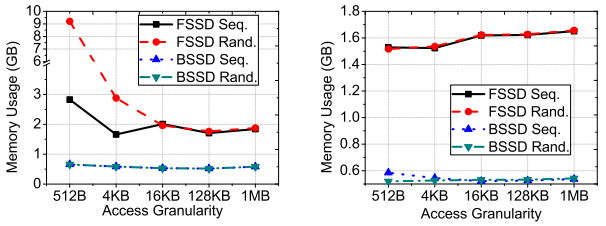
	From-scratch SSD	Bridge-based SSD
Code-name	(FSSD)	(BSSD)
Interface	PCIe 2.0 x4	PCIe 2.0 x4
Flash Software Module	Host-side kernel driver	Storage-side firmware
Price	$2490	$2152
Controller Type	Xilinx FPGA	SAS-to-PCIe Bridge
Storage capacity	430GB	400GB
Write Bandwidth	700MB/sec	750MB/sec
Read Bandwidth	1GB/sec	1.4GB/sec
512B I/O Latency	45μsec	65μsec
Flash Type	QDP MLC	eMLC
Internal DRAM	Publicly N/A	2GB
Debut	2012 Q2	2012 Q3

Table 1: Important characteristics of the tested PCIe SSDs.

sequently, there exist several prior proposals that try to migrate the flash software to the host-side, as illustrated on the right side of Figure 2b. In addition, by implementing the flash software modules on the host side, we can 1) unify indirect flash software logic [1, 15, 7] and 2) overlap storage and data processing times by exploiting abundant host-side computation and memory resources [13, 10]. Specifically, [7] proposes virtual storage layer (VSL) and direct file system (DFS) by migrating the flash software module from the storage side to the host side (especially FTL), so that it can optimize data accesses as well as offer extensive OS support. [15] unifies FTL and the host-side file system to remove indirect address mapping, and [13] moved the internal buffer cache to the host-side to improve performance when targeting write-intensive workloads. [10] migrates garbage collector and page allocator [11] from SSD to the host-side software stack. Thanks to this flash software module migration, a from-scratch SSD can maximize throughput while reducing latency.

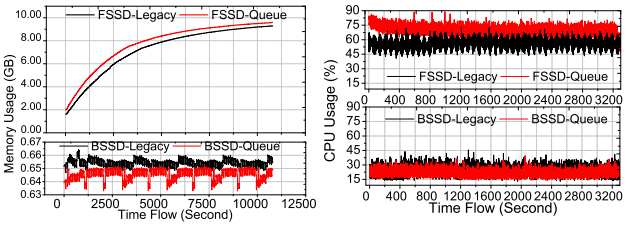
3 Experimental Setup

PCIe SSDs. We chose two most-recently-released, cutting edge PCIe SSDs from two different vendors. Since our goal is not to perform reverse engineering of these commercial products, we refer to each of them using a code-name – *FSSD* refers the from-scratch SSD, and *BSSD* refers to the bridge-based SSD. Our SSDs and their important characteristics are listed in Table 1. It should be noted that, even though these two architectures have been built based on very different deign concepts, both PCIe SSDs are geared toward offering shorter latency and better throughput, and are designed mainly for workstations.
System Configuration. Our experimental system is equipped with an Intel Quad Core i7 Sandy Bridge 2600 3.4 GHz processor and "16GB" memory (four 4GB DDR3-1333Mhz memory). In this system, all of the functions of the Northbridge reside on the CPU, and all SSDs we tested are connected to Sandy Bridge through the PCIe 2.0 interface. We executed all our tests in NTFS, and stored logs and output results into separate block devices in a full asynchronous fashion; *neither a system partition nor a file system is created on our SSD test-beds*. Note that this configuration allows each SSD test-bed to be completely separated from the evaluation scenarios and tools.
Measurement Tool. We modified an Intel open source storage tool, called *Iometer* [4], to capture time series of performance characteristics and host-side memory usage. To measure accurate memory usage at a given time, we added a module in calling *GlobalMemoryStatusEx* into Iomoter, which is an Window system function that allows users to retrieve the current state of both physical and virtual memory. In addition, in order to minimize interference between successive evaluations, our modified Iometer physically erases whole region of the underlying device through *secure erase command* in SMART, in every evaluation step for writes.



(a) Write (b) Read

Figure 3: Memory usage.



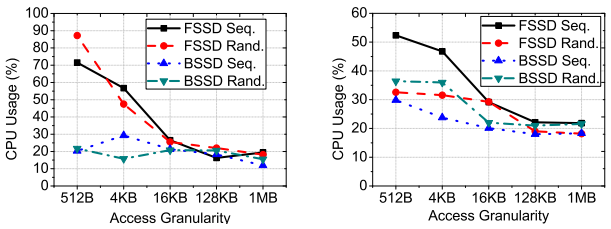
(a) Memory usage comparison (b) CPU usage comparison

Figure 4: Time series comparison for host resources usage between FSSD (top) and BSSD (bottom) with the default 512B block-size access operation.

4 Challenges in Resource Management

4.1 Memory Usage

Overall memory usage evaluation. One can see from Figure 3 that FSSD needs at least 2GB memory for writes and 1.5GB memory space for reads while BSSD requires only 0.6GB memory space regardless of the I/O type and size. As a result of flash software migration, the host side kernel drivers require memory space in loading their image and containing their in-memory structures. In addition, we believe that there are two main reasons why FSSD consumes on average sixteen times more memory space than BSSD. First, the direct file system [7] and migrated flash software [1, 15, 10] require host-side memory to maintain huge mapping tables. Second, the host-side write buffer cache consumes memory space in hiding the underlying flash memory complexity, such as garbage collections, endurance [2] and intrinsic latency variation [12]. For instance, as shown in Figure 3, FSSD's memory usage varies based on access granularity and pattern, whereas the BSSD's memory usage is not different by them. This is because, in the bridge-based SSD architecture, the table is implemented in the SSD, and data processing is only performed at the storage side.
Time series analysis. Figure 4a plots memory usage of FSSD (top) and BSSD (bottom) over time. In this test, we evaluated them with a 512B block access granularity since all the system-level operations are block-based, and the default block-size is 512B. In addition, we performed the memory usage test based on two different I/O access scenarios: 1) queue mode operation (using 128 queue entries) and legacy mode operation (submitting the request whenever the device is available to serve an I/O request). One can see from the figure that FSSD requires



(a) Write (b) Read

Figure 5: CPU usage.

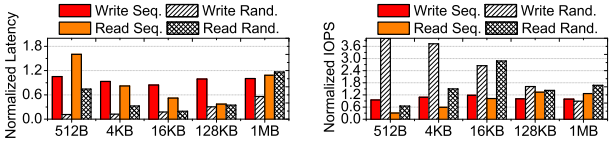
about 2GB memory space at the very beginning of the I/O process for both queue and legacy mode operations. Interestingly, as the I/O process progresses, the amount of memory usage keeps increasing in a logarithmic fashion and reaches about 10GB. It should be noted that, considering that the target system is a workstation, we believe that 10GB memory usage to manage only the underlying SSDs may not be acceptable in many applications. In contrast, as shown in the bottom part of the figure, BSSD keeps memory usage around 0.6GB over time. We believe that the reason why the memory usage of FSSD keep increasing over time is because of the host-side address mapping and caching. In particular, DFS/VFS [7] uses a B-tree structure to map addresses between the physical and virtual spaces, which tends to increase the memory requirements of the mapping information by adding more node entries to serve incoming I/O requests. We also believe that the huge memory usage is primarily caused by host-side buffer caching.

4.2 CPU Usage

Overall CPU usage evaluation. Figures 5a and 5b give CPU usage on the host-side in serving reads and writes, respectively. Similar to memory usage analysis, FSSD requires computation power about three times more than BSSD, except for cases where access granularity is larger than 16KB. We conjecture that one of main reasons why FSSD requires higher CPU usage (52%∼87%) for finer granular I/O accesses is that smaller size I/O requests leads to an increase in the size of the address mapping table lookup and update (or cache lines of host-side buffer). In contrast, BSSD only consumes 20%∼30% for the same I/O services. This is because the mapping table lookup and update processing are performed on the storage-side.
Time series analysis. Figure 4b compares the CPU usage of FSSD (top) and BSSD (bottom) under the workloads that exhibit high number of default block size accesses. As before, we evaluated our PCIe SSD test-beds with both legacy and queue mode operations. FSSD consistently consumes 60% of the cycles on the host-side CPU with legacy mode operations, and I/O service with queue mode operation requires 50% more CPU cycles than the legacy mode. We believe that a CPU usage over 60% for just I/O processing can degrade overall system performance. In contrast, BSSD only uses about 20% CPU cycles irrespective of the I/O operation mode.

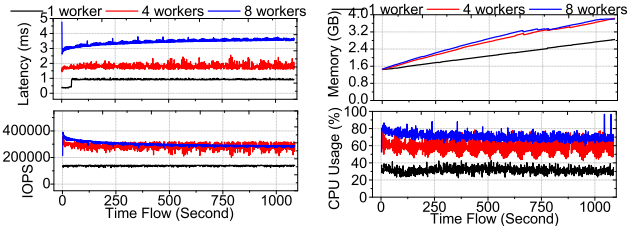
4.3 Challenges in System Performance

Overall performance comparison. It is hard to directly compare the microscopic performance characteristics on the two different SSD architectures since their flash software and platforms have different optimization



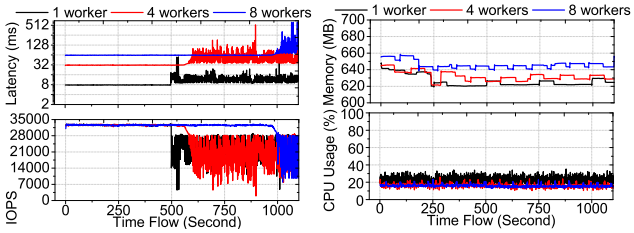
(a) Latency (b) IOPS

Figure 6: Latency and throughput comparison. Note that all the latency and throughput of BSSD values are normalized to corresponding values of FSSD.



(a) Performance (b) Resource

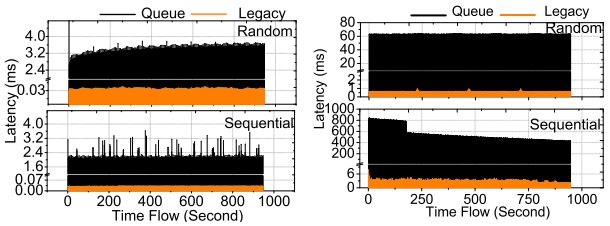
Figure 7: FSSD performance characteristics on the multi-core environment.



(a) Performance (b) Resource

Figure 8: BSSD performance characteristics on the multi-core environment.

techniques. For example, we observed that BSSD's random writes with default block-size accesses exhibit 7.2 times better performance compared to sequential writes, which is opposite to common expectation on most modern SSDs. We believe that this is because BSSD puts incoming default-block size I/O requests into its internal 2GB DRAM buffer and additional non-volatile SRAM [3], but forwards the large sized I/O requests to the underlying flash memory, which in turn shows the unexpected performance. Consequently, we compare the overall performances of BSSD and FSSD. Figures 6a and 6b compare the latency and IOPS between FSSD and BSSD. We see that most latency values observed with BSSD are on average 39% worse than FSSD, which is opposite to the information one could obtain from the datasheets of these SSDs (see Table 1). We believe multiple controllers, indirect address mapping modules, and protocol conversion overheads of BSSD on data path from CPU to flash memory collectively contribute to this longer latency.
Multi-core system environment. To evaluate the performance impact in a multi-core system environment, we executed different number of I/O processing workers (ranging between 1 and 8) on FSSD and BSSD in parallel (with 512B granularity random access upon 128 queue entries). The results considering FSSD and BSSD as shared resources are plotted in Figures 7 and 8. The latency of both FSSD and BSSD increases as we increase the number of



(a) FSSD (b) BSSD

Figure 9: Queueing latency comparison observed by FSSD (a) and BSSD (b).

workers. Specifically, latency values with eight workers on FSSD and BSSD are worse than four workers by 118 % and 108%, respectively and worse than single worker by 289% and 704%, respectively. Throughput trends are a bit different compared to latency trends. While BSSD has no IOPS benefits by increasing the number of workers, the IOPS of FSSD increases. The IOPS of four workers are 2.2 times better than single worker evaluation. However, the advantage of many workers decreases because of the higher memory and CPU usages. In contrast, BSSD shows similar IOPS and host-side resource usages irrespective of the number of workers employed.

Queuing latency. Device-level queueing mechanisms are one of the crucial components, which can improve storage throughput. For example, NVMe offers 64K queue entries [5]. SAS/SATA [14, 6] also provides a device-driven queue mechanism, which allows the storage devices to determine the order of I/O request executions without any host-side software interrupts. However, we observed that the latency values with a queuing method significantly drop irrespective of the SSD architecture. As shown in Figure 9, the random and sequential write latencies of FSSD are longer than legacy mode latencies by about 106 times. Similarly, BSSD resulted in 99 times worse latency with the queue mode operation than the one with legacy mode. We believe that these significant latency drops with the queue mode operation would be a problematic obstacle to bring flash memory closer to the memory-processor subsystem.

Garbage collections. Figures 7a and 8a (with the default block size random accesses upon 128 queue entries) also reveal the difference between FSSD and BSSD regarding the management of the underlying garbage collections (GCs). While FSSD offers very sustained performance, the latency and throughput values of BSSD drop starting with the half of I/O execution. This is mainly because of GCs, which are a series of SSD internal tasks reading data from old flash block(s), writing them to new block(s), and erasing the old block(s). This performance drop caused by GCs is also referred to as *write cliff* [12]. One of the reasons behind the sustained performance of FSSD is the ample host-side buffer and the optimized flash software stack. It should be noted that BSSD also employs a 2GB internal memory as buffer, but it cannot hide the GC overheads, which means that the sustained performance of FSSD does not solely come from the available buffer.

5 System Implication and Future Work

Co-operative approach. In summary, we observed that, while the performance of from-scratch SSD is better than the bridge-based SSD, the former requires huge host-side resources, which may not be acceptable in many cases. We believe that an approach that partially migrates flash software functionalities from SSD to the host-side can be a promising mid-way option in achieving higher performance and lower host-side resource consumption. For example, FTL partitioning [8] moves only the address mapping module rather than moving the whole FTL cores (e.g., buffer cache, wear-leveler). Similarly the middleware and firmwmare cooperative approaches [10] only move the garbage collector, and I/O scheduler [9] is aware of internal parallelism from the storage-side to the host-side, which requires less system memory resources.

All-flash storage arrays. PCIe SSD based all-flash arrays or SSD RAID systems can directly experience the host-side resource challenges we demonstrated so far. For example, if system designers build a 5-RAID system based on FSSD, it requires approximately 50GB memory space, and they have to carefully design the processors to manage the underlying five FSSDs. If the designers build the RAID with BSSD and intend to improve performance by striping all incoming I/O requests over the five BSSDs, they need to carefully manage GCs globally because the probability that the system could suffer from a straggler performing GCs significantly increases.

Future work. One can envision a a variety of flash software implementations on the two representative SSD architectures studied. In our future work, we want to explore such implementations. We also plan to analyze the system characteristics related to the host-side computational resources such as interrupt handling methods, buffer management strategies and zero-copy efficiency upon the PCIe SSD architectures.

References

[1] ARPACI-DUSSEAU, A. C., ET AL. Removing the costs of indirection in flash-based ssds with namelesswrites. In *Proc. of HotStorage* (2010).

[2] BOBOILA, S., AND DESNOYERS, P. Write endurance in flash drives: Measurements and analysis. In *Proc. of FAST* (2010).

[3] CYPRESS. *CY14B256LA nvSRAM*. 2012.

[4] INTEL. *Iometer User's Guide*. 2003.

[5] INTEL. NVM express revision 1.0.

[6] INTEL, AND SEAGATE. Serial ATA NCQ.

[7] JOSEPHSON, Y., ET AL. DFS: A file system for virtualized flash storage. In *Proc. of FAST* (2010).

[8] JUNG, M., ET AL. Cooperative memory management. In *US Patent 2008189485* (2008).

[9] JUNG, M., ET AL. Physically addressed queueing (PAQ): Improving parallelism in solid state disks. In *ISCA* (2012).

[10] JUNG, M., AND KANDEMIR, M. Middleware - firmware cooperation for high-speed solid state drives. In *Proc. of Middleware D&P* (201).

[11] JUNG, M., AND KANDEMIR, M. An evaluation of different page allocation strategies on high-speed SSDs. In *HotStorage* (2012).

[12] JUNG, M., AND KANDEMIR, M. Revisiting widely held ssd expectations and rethinking system-level implications. In *Proc. of SIGMETRICS* (2013).

[13] KOLLER, R., ET AL. Write policies for host-side flash caches. In *Proc. of FAST* (2013).

[14] T10. SCSI storage interfaces.

[15] ZHANG, Y., ET AL. De-indirection for flash-based ssds with namelesswrites. In *Proc. of FAST* (2012).