JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 28, 999-1018 (2012)

ReSSD: A Software Layer for Improving the Small Random Write Performance of SSDs*

YOUNGJAE LEE[1], JIN-SOO KIM[2] AND SEUNGRYOUL MAENG[1]
[1]*Computer Science Department*
Korea Advanced Institute of Science and Technology
Daejeon, 305-701 Korea
[2]*School of Information and Communication Engineering*
Sungkyunkwan University
Suwon, 440-760 Korea

Recently, NAND flash-based solid state drives (SSDs) have emerged as revolutionary storage media. Numerous studies have been carried out to employ SSDs in database systems and storage systems, motivated by SSD's attractive features such as decreased drive weight, increased shock resistance, low power consumption, and no seek latency. However, low-end SSDs targeting desktop and notebook environments show problematic random write performance which is only comparable to or lower than that of HDDs

This paper proposes a novel software layer called ReSSD whose purpose is to improve the small random write performance of low-end SSDs with low memory usage. ReSSD works as a virtual block device on top of SSD which requires no modification neither in the operating systems nor in the applications. By inspecting all incoming requests, ReSSD identifies small random writes which have potential to degrade SSD's performance significantly and transforms them into sequential and ordered-sequential writes which are more favorable to SSDs. Our evaluation results through Postmark and OLTP benchmarks show that the proposed approach accomplishes noticeable performance improvement on low-end SSDs under all workloads.

Keywords: solid state drives (SSDs), NAND flash memory, small random write, virtual block device, operating system

1. INTRODUCTION

Currently, NAND flash-based solid state drives (SSDs) have drawn considerable attention in both industry and academia. Many studies have been carried out to replace hard disk drives (HDDs) with SSDs in database systems or storage systems [1-3]. Several hybrid approaches are also under investigation which insert SSDs as a cache component into existing HDD-only storage systems [4-7].

Most of the previous studies have been motivated by attractive properties of SSDs. Since there is no mechanical part in SSDs, they are lighter, more robust, and more energy-efficient than HDDs. In particular, since there is no seek time which has been a key bottleneck in HDD's performance, SSDs can provide fairly good random access performance as well as high sequential access performance. SSDs targeted at enterprise server environments (called *high-end SSDs*) present more than 30 times higher band-

Received May 31, 2011; accepted March 31, 2012.
Communicated by Junyoung Heo and Tei-Wei Kuo.
* This work was supported by Next-Generation Information Computing Development Program (No. 2011-0020520) and by Mid-career Researcher Program (No. 2011-0027613) through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology. This work was also partly supported by the IT R&D program of MKE/KEIT (KI10041244, SmartTV 2.0 Software Platform).

widths in random reads and writes compared to HDDs [2, 8].

However, the random write performance of *low-end SSDs* aimed at desktop environments (PCs, notebooks, and netbooks) is somewhat disappointing, particularly in small size, due to stringent cost constraint. In fact, the bandwidth of small random writes is merely around 1 MB/s in low-end SSDs. This is only comparable to or sometimes lower than that of HDDs, although the random read bandwidth is still much higher as in high-end SSDs [2, 8]. For example, the 4KB random write bandwidths of SSDs from Samsung and SuperTalent are 0.42 MB/s and 0.018 MB/s, respectively, while their 4KB random read bandwidths are more than 8 MB/s.

This basically originates from the nature of NAND flash memory. In NAND flash memory, the previous data should be erased first to write another data in the same location and the erase unit is larger than the read and write unit. These unique characteristics of NAND flash memory are usually handled transparently by the SSD's firmware called *Flash Translation Layer* (*FTL*). Yet, small random writes increase the FTL overhead in managing NAND flash memory, which potentially leads to significantly lower bandwidth. In case of high-end SSDs, the FTL overhead is mitigated through various hardware enhancements to SSD's internal architectures. However, since the hardware enhancements inevitably increase the manufacturing cost, low-end SSDs are not usually equipped with such enhancements. As a result, low-end SSDs show very poor small random write performance which offsets many of their benefits.

A software company called EasyCo LLC has recently proposed a performance enhancement solution called Managed Flash Technology (MFT) to improve the small random write performance of SSDs [9]. Although the detailed architecture of MFT is still not known exactly, it appears to work as a virtual block device on SSDs and handle all write requests in a sequential fashion as in LFS [10], so that the underlying SSD never encounters random writes. Under a workload including many small random writes, the use of MFT may show the write bandwidth comparable to the sequential write bandwidth. However, MFT has a serious drawback that the amount of mapping information needed to keep track of physical locations becomes very huge. It is known that MFT on a 128 GB SSD typically requires about 150 MB of kernel memory. Considering the current trend of ever increasing SSD storage capacity, the MFT approach is not practical especially for systems with a modest amount of memory such as notebooks or netbooks.

In this paper, we propose a novel software approach called ReSSD† which aims at resuscitating low-end SSDs from their poor small random write performance with low memory usage. ReSSD divides the underlying SSD into two areas, the normal area and the reserved area, and shows only the normal area to the upper layer (such as page cache or file systems) as a single virtual block device. Unlike MFT which manages the entire SSD space in a log-structured manner, ReSSD identifies small random write requests and records those requests sequentially in the reserved area. If the amount of free space in the reserved area becomes lower than a certain threshold, data written to the reserved area are eventually moved to its original locations in the normal area in an ordered-sequential write fashion. Since the small random writes are transformed into sequential and ordered-sequential writes which are favorable to low-end SSDs, the overall performance of low-end SSDs is improved. Note that the ordered-sequential write denotes the write pattern where the accessed logical blocks are arranged in increasing order of LBAs (Logical Block Addresses) [12].

† This paper is the extended version of the poster paper [11] presented in the 25th ACM Symposium on Applied Computing (ACM SAC 2010).

In ReSSD, the reserved area size is smaller than the normal area size by an order of magnitude. Thus, the amount of in-memory mapping information is also very small. According to our evaluation results using Postmark [13], ReSSD improves the overall performance by 43% with using only 5.2% of the workload size as the reserved area. In this configuration, the mapping information occupies only 1.7 MB of kernel memory. In addition, ReSSD is completely transparent to applications and other components of the operating system kernel. We have implemented ReSSD as a kernel module which can be plugged into the kernel without any kernel modifications. Since ReSSD supports the standard interface of the general block device, page cache or I/O scheduler considers it as a usual block device.

The rest of this paper is organized as follows. Section 2 describes the target SSDs of this paper and summarizes related work. In section 3, we describe the design and implementation details of ReSSD. Experimental evaluation results are shown in section 4. Finally, section 5 concludes this paper.

2. BACKGROUND

2.1 High-end vs. Low-end SSDs

A typical SSD is composed of a number of NAND flash memory chips connected by multiple channels, a DRAM buffer, a host interface, a controller, and other logic including ECC (error correction codes) hardware. Each NAND flash memory chip has a number of blocks and each block, in turn, consists of 32-128 pages. A page is the unit of read and written operations, while a block is the unit of erase operations.

SSDs have some unique characteristics which make them distinct from HDDs. These characteristics are essentially rooted in the nature of NAND flash memory. The most remarkable one is the absence of mechanical parts which brings many technical advantages compared to HDDs such as decreased drive weight, increased shock resistance, low power consumption, and no seek latency. These advantages have stimulated considerable research which aims to integrate SSDs into the storage hierarchy effectively.

Another notable difference of SSDs is the use of sophisticated firmware called FTL [14-17] inside the SSD controller. The most important role of FTL is to provide the traditional block device interface over NAND flash memory by hiding the presence of erase operations. Since the overwritten is not allowed in NAND flash memory, SSDs are usually over-provisioned with a certain amount of extra blocks. FTL redirects the incoming write requests to empty pages of extra blocks and maintains the mapping information between their logical addresses and the physical addresses of the NAND pages. Those pages containing the previous version of data are invalidated by FTL and reclaimed later by the procedure known as garbage collection. When a block is reclaimed, valid pages in the block should be migrated to other empty pages before the block is erased.

Small random writes on SSDs have potential to increase the FTL's overhead significantly in reclaiming invalidated pages. Since small random writes update contents across a wide range of the logical address space, invalidated pages will be scattered over numerous blocks. In order to reclaim a certain amount of invalidated pages, FTL should

erase a large number of blocks which involves significant overhead in migrating valid pages.

To address this problem, high-end SSDs developed for enterprise server environments are equipped with hardware enhancements such as rich DRAM buffer and fat provisioning [2, 8, 18]. Rich DRAM buffer absorbs bursts of small random writes effectively, reducing the number of physical writes on flash memory. Fat provisioning, which keeps much more extra clean blocks, is useful for minimizing the average latency of small random writes through avoiding extra write and erase operations. The high-end SSDs with such enhancements show very good performance in small random writes. Chen *et al*. presented that 4 KB random write bandwidth of a high-end SSD is about 50 MB/s, more than 30 times higher than that of HDDs [8]. Also, Lee *et al.* showed that an enterprise-class SSD outperforms HDDs by more than 5 times in the random write throughput of 8 KB-sized data [2].

On the other hand, low-end SSDs targeting desktop and notebook environments are equipped with relatively small DRAM buffer and thin provisioning to save manufacturing cost. According to Chen *et al*. [8], the price of low-end SSDs is only one fifth of that of high-end SSDs. Consequently, low-end SSDs show significantly low small random write bandwidth around 1 MB/s, which is just comparable to or worse than that of HDDs [2, 8, 19]. Yet, low-end SSDs present very good performance in random reads and sequential reads/writes as high-end SSDs do.

In this paper, we focus on improving the small random write performance of such low-end SSDs. ReSSD is a relatively cheap solution which requires a little amount of main memory and CPU power of the host. Since our solution is a software-only approach, it can be applied to existing SSDs unlike the costly hardware enhancement requiring modifications of SSD's internals.

2.2 Related Work

Rajimwale *et al*. listed several "*unwritten contracts*" in the software stack of storage systems and explained how each of them fails when applied to SSDs, resulting in poor performance and lifetime [19]. In particular, they noted some of the tested SSDs have random-write performance that is worse than HDDs due to write amplification, *i.e.*, the overhead of FTL.

Chen *et al*. conducted intensive experiments and measurements on different types of state-of-the-art SSDs, from low-end to high-end SSDs, to gain insight into the unique performance characteristics of SSDs [8]. They measured latencies of read and write operations whose access patterns are sequential, random, and strided, and analyzed the results based on the internal knowledge of each type of SSDs. They found that a low-end SSD has a much lower random write bandwidth of 1.14 MB/sec, which is only comparable to the HDD (1.49 MB/sec), even though its sequential write bandwidth is 88 MB/sec.

Several previous studies have attempted to address the issue of poor small random write performance in low-end SSDs. Dumitru proposed MFT which works as a virtual block device on SSDs [9]. While ReSSD tries to identify only the small random writes which are expected to degrade the performance of SSDs significantly, MFT stores all the write requests in a sequential fashion as in LFS [10]. In MFT, the underlying SSD receives sequential writes only, never suffering from random writes. Thus, under the work-

load including many small random writes, MFT can show the write bandwidth comparable to the sequential write bandwidth. However, MFT needs to keep /track of the logical-to-physical mapping information for every 4 KB logical block, whose size can be substantial for large-capacity SSDs. About 150 MB of kernel memory is known to be required to apply MFT on a 128 GB SSD. As the capacity of SSD is getting larger, MFT's high memory footprint can be a limiting factor. In addition, the detailed architecture of MFT has not been available to the public, including how the garbage collection is performed when there is no more free space in SSD to handle the incoming data.

Unlike MFT, the memory overhead of ReSSD is fairly moderate. In ReSSD, only the requests identified as small random writes are handled in a sequential manner, and they are eventually moved to their original locations due to the limited size of the reserved area. Therefore, the amount of mapping information is much smaller than MFT and bounded by the reserved area size, as it is sufficient for ReSSD to keep track of mapping information only for those data currently stored in the reserved area. ReSSD is able to achieve significant performance improvement with dedicating only a few percent of the workload size as the reserved area.

Kim *et al.* presented *FlashLite* which is a user-level library to support the standard file system interface [20]. Its goal is to eliminate random writes generated when downloading a huge file through P2P protocols. When downloading a file, it transforms all writes to the file into sequential writes and maintains mapping information for their original locations in memory, like MFT [9] and LFS [10]. After downloading the file completely, its contents are rewritten to their original locations so that they can be accessed without the special FlashLite interface. According to their experiment results, random writes are almost eliminated and the authors insist that the durability of SSD is improved significantly. Unlike ReSSD, they focused on the durability of SSD and did not present evaluations about the performance improvement. Also, since FlashLite is a user-level library, applications have to be modified and linked with the FlashLite library, while ReSSD doesn't require modification neither in applications nor in other parts of the operating system.

3. RESSD

In this section, we present the overall architecture of ReSSD and how ReSSD reshapes the write pattern to improve the small random write performance for low-end SSDs.

3.1 Overall Architecture

The primary design goal of ReSSD is convenience and ease of deployment. In compliance with this goal, we have designed ReSSD as a virtual block device which works on an SSD. The virtual block device supports the standard interface of the general block device as illustrated in Fig. 1.

The upper layer components, such as page cache and virtual file system (VFS), view it as a single block device. They issue ordinary block I/O requests or build file systems on top of ReSSD. The I/O scheduler at the lower layer also considers it as a single device. ReSSD delivers common disk I/O requests to the scheduler as other block devices do.

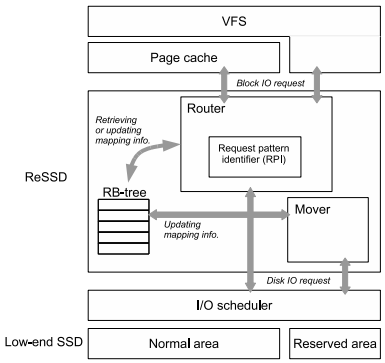


Fig. 1. The architecture of ReSSD.

In addition, ReSSD is implemented as a kernel module which can be inserted by a simple command without any kernel modifications. Thus, other parts of the operating system need not be changed in order to use ReSSD.

3.2 Basic Components

As Fig. 1 depicts, ReSSD consists of three major components: *the router*, *the mover*, *and the RB-tree* (*red-black tree*). The underlying SSD is divided into two regions, *the normal area* and *the reserved area*. The normal area is visible to the upper layer as a single block device. On the other hand, the upper layer is not aware of the reserved area. All I/O requests towards the reserved area are issued only by ReSSD.

The router handles all the incoming requests from the upper layer. The upper layer issues I/O requests as if ReSSD were a normal block device. All of those requests are first transferred to the router. In case of write requests, *the request pattern identifier* (*RPI*) inside the router checks whether the request is a small random write or not. If the request is identified as a small random write, its data is sequentially written into the reserved area. Otherwise, the request is forwarded to the normal area. When the router receives read requests, it first looks up the RB-tree which maintains the mapping information (described below) to see whether the data is in the reserved area or in the normal area. If there is no corresponding mapping information in the RB-tree, it means that the requested data is resident in the normal area and the request is served directly from the normal area. Otherwise, the request is redirected to the reserved area with its sector addresses being modified to the ones in the reserved area as designated by the RB-tree.

The main role of the mover is to reclaim free space in the reserved area by moving some of data to the normal area. The mover usually remains inactive until the router wakes it up as the amount of free space falls below a certain threshold. The default value of the threshold is half the reserved area size. The operations of the router and the mover will be described in more detail in section 3.3.

The RB-tree contains the mapping information needed to track the original locations of data, which is sequentially written to the reserved area. It keeps the current locations in the reserved area and original locations in the normal area where they were supposed to be written. The size of the mapping unit is set to 4 KB, which is the same as the I/O

unit of page cache. All nodes of the RB-tree are always kept in memory and their memory usage is about 1 MB per 100 MB of the reserved area.

3.3 Improving Small Random Writes

This subsection provides details about how the router and the mover transform small random writes into sequential and ordered-sequential writes.

The logical sector number and the size of every incoming I/O request are inspected by the RPI module. A write request is considered as a small random write if its logical sector numbers (LSNs) do not immediately follow the previous write request and the size of the request is not larger than 8 KB, as this type of write request has potential to degrade SSD's performance significantly. The router redirects such identified small random writes to the reserved area and their data are written sequentially. In case the reserved area already has some data for the requested sector numbers, the previous versions of data are invalidated by the router. Note that the router issues an additional write request to the reserved area to update the mapping information on the underlying SSD. In this way, ReSSD's in-memory mapping information is protected against potential loss that may occur from sudden power failure.

As soon as the mover becomes active by the router, it moves a certain amount of data from the reserved area to the normal area. The amount of data moved at a time is configurable and is currently set to 512 KB by default. The mover reads valid data from the reserved area in the ascending order of its original LSNs and writes it into the normal area in an ordered-sequential write fashion. Most of the requests handled by the reserved area are 4 KB or 8 KB in size. While they are staying in the reserved area, they have high chance of being merged with other write requests directed to the nearby locations. As a result of this, many of the ordered-sequential writes issued by the mover show the request size larger than or equal to 12 KB.

By filtering out small random writes, the underlying SSD does not suffer from random writes whose size is less than or equal to 8 KB. Instead, the underlying SSD only encounters sequential writes given by the router and ordered-sequential writes issued by the mover. Fig. 2 compares the bandwidth of random, ordered-sequential, and sequential

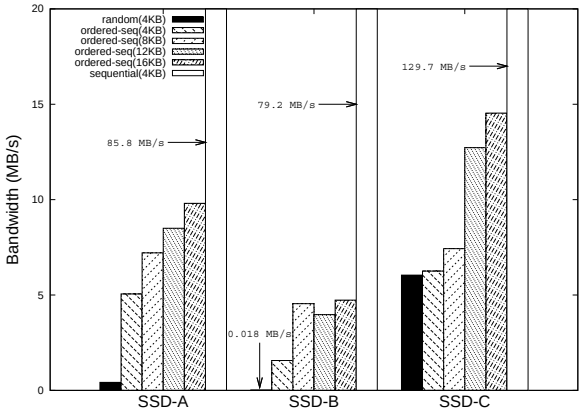


Fig. 2. Bandwidth of the random, sequential, and ordered-sequential write.

write for three commercial SSDs used in the evaluation of this paper. SSD-A and SSD-B are typical low-end SSDs available in the market whose 4 KB random write bandwidth is lower than 1 MB/s. SSD-C's random write performance is higher than typical low-end SSDs. The detailed information of each SSD used in this paper is summarized in Table 2. In Fig. 2, the number in parentheses for each write pattern represents the request size used. The increment value [12] of ordered-sequential writes, which denotes the distance between two successive ordered-sequential write requests, is set to 128 KB.

First of all, we can see that the sequential write outperforms the random write in all SSDs. In particular, the sequential write bandwidth of SSD-B is about 4,300 times higher than the random write bandwidth. The ordered-sequential write also outperforms the random write by at least a factor of 12 in SSD-A and by at least a factor of 86 in SSD-B. The bandwidth of ordered-sequential write grows as the request size increases. In SSD-C, the ordered-sequential write only doubles the bandwidth compared to the random write. We believe that this is because SSD-C is equipped with some hardware enhancements for effective handling of small random writes, which low-end SSDs do not have.

Since sequential and ordered-sequential writes show higher performance than small random writes in low-end SSDs, ReSSD can improve the overall performance significantly. Moreover, when the data in the reserved area is invalidated by the subsequent write requests, no additional ordered-sequential write is necessary to move them to the normal area. In this case, small random writes are completely replaced with sequential writes, which enables ReSSD to attain maximum improvement in write performance. Therefore, ReSSD is especially beneficial for those workloads that exhibit high temporal locality of reference among small random writes, as most of them are converted to sequential writes without generating additional ordered-sequential writes.

3.4 Implementation

We have implemented ReSSD using *the device-mapper* [21], one of Linux kernel components for logical volume management. The device-mapper provides a framework to group a number of logical block devices into a single virtual block device and to redirect incoming block I/O requests to one of the devices. Through this framework, various storage subsystems such as RAID and an encrypted block device can be implemented without any hardware supports [22].

ReSSD is implemented as a kernel module linked to the device-mapper framework and this module does not require any kernel modifications. The router is based on a *map* function of the framework which hooks all the block I/O requests towards the virtual block device created by the framework. The RB-tree uses a red-black-tree library of the Linux kernel and the mover is implemented as a kernel thread. After inserting the ReSSD module into the kernel, users can create a virtual block device by a user-space tool called *dmsetup* and build any file systems on top of it.

3.5 Analytical Model

In this subsection, through analytical modeling of the time taken to handle small random writes, we derive a condition that needs to be satisfied to realize the performance improvement when the proposed technique is used. Table 1 defines symbols used in this analysis.

Table 1. Symbols and descriptions.

Symbol	Description
SW	sequential write bandwidth
RW_{nKB}	nKB random write bandwidth
RR_{nKB}	nKB random read bandwidth
OW_{nKB}	nKB ordered-sequential write bandwidth
B_{nKB}	the number of bytes moved by nKB ordered-sequential writes
α	the ration of invalidated data in the reserved area

Let us assume that a workload writes the total N bytes using a series of 4 KB random writes. Eq. (1) presents the time needed to perform such random writes on an SSD.

$$T_{SSD} = \frac{N}{RW_{4KB}} \tag{1}$$

The time taken to handle the 4KB random writes on ReSSD can be given by Eq. (2).

$$T_{ReSSD} = \frac{N}{SW} + \frac{(1-\alpha)N}{RR_{4KB}} + \sum_{i=4,8,\ldots} \frac{B_{iKB}}{OW_{iKB}} \text{ where} \sum_{i=4,8,\ldots} B_{iKB} = (1-\alpha)N \tag{2}$$

T_{SSD} consists of the following three terms. The first term denotes the time taken to record all the 4 KB random writes sequentially in the reserved area. The second one indicates the time used by the mover to read all the valid data from the reserved area. The amount of the valid data is $(1 - \alpha)N$ bytes, where α represents the average ratio of data invalidated by subsequent writes in the reserved area. The mover reads the valid data in increasing order of their original LSNs via 4 KB random reads. The last term denotes the time for the mover to write the valid data into the original location in the normal area using ordered-sequential writes larger than or equal to 4 KB in size.

If $T_{ReSSD} < T_{SSD}$, the use of ReSSD can improve the performance of 4 KB random writes in SSDs. However, α and B_{nKB} are dependent on the temporal locality of the workload and the reserved area size. We eliminate these variables as follows in order to derive a condition independent of the workload and the reserved area size.

Since $OW_{4KB} < OW_{iKB}$ for $\forall i > 4$,

$$T_{ReSSD} < \frac{N}{SW} + \frac{(1-\alpha)N}{RR_{4KB}} + \frac{\sum_{i=4,8,\ldots} B_{iKB}}{OW_{4KB}} = \frac{N}{SW} + \frac{(1-\alpha)N}{RR_{4KB}} + \frac{(1-\alpha)N}{OW_{4KB}}. \tag{3}$$

Because α is between 0 and 1,

$$T_{ReSSD} < T'_{ReSSD} = \frac{N}{SW} + \frac{N}{RR_{4KB}} + \frac{N}{OW_{4KB}}. \tag{4}$$

$T'_{ReSSD} < T_{SSD}$ obviously means $T_{ReSSD} < T_{SSD}$. Therefore, Eq. (5) is a sufficient condition for *ReSSD* to improve the 4 KB random wrote performance of *SSD*s. Note that we can easily derive a sufficient condition to improve the performance of 8 KB random

writes in a similar way.

$$\frac{1}{SW} + \frac{1}{RR_{4KB}} + \frac{1}{OW_{4KB}} < \frac{1}{RW_{4KB}} \tag{5}$$

In SSDs, sequential writes and 4 KB random reads are usually much faster than 4 KB random writes. Therefore, Eq. (5) is satisfied if the 4 KB ordered sequential write performance (OW_{4KB}) is better than the 4 KB random write performance (RR_{4KB}), *i.e.*, if the 4 KB ordered sequential write is fast enough to compensate the overhead of extra write operations needed to move the valid data in the reserved area to the normal area. In general, the ordered-sequential write bandwidth of low-end SSDs is higher than the small random write bandwidth by several times. In Fig. 2, the ordered-sequential write performance of SSD-A and SSD-B is more than ten times higher than the small random write performance. Therefore, we can see that typical low-end SSDs such as SSD-A and SSD-B satisfy the condition shown in Eq. (5). In case of SSD-C whose ordered-sequential write performance shows only marginal improvement over the small random write performance, we expect that ReSSD can improve the overall performance only when α is large due to the high temporal locality in the workload and/or the large reserved area size.

4. EVALUATION

In this section, we show that ReSSD improves the overall performance of low-end SSDs effectively for the workloads with many small random writes.

4.1 Environments

We have evaluated ReSSD on a machine equipped with 2.20 GHz Intel Core2Duo CPU and 4 GB of main memory, running the Debian Linux kernel 2.6.29.4. The memory size available to the kernel is limited to 384 MB in order to minimize the effect of page cache. We have used the ext4 file system with default configurations and the default CFQ scheduler.

Table 2. Characteristics of SSDs used in the evaluation.

	SSD-A	SSD-B	SSD-C
Manufacturer	Samsung	Super Talent	OCZ
Model	MCCOE64G5MPP	FTM60GK25H	OCZSSD2-1VTX60G
Capacity	64 GB	60 GB	60 GB
Sequential Read BW	110 MB/s	117 MB/s	220 MB/s
Sequential Write BW	85 MB/s	79 MB/s	129 MB/s
Random Read BW	16.3 MB/s	8.51 MB/s	20.2 MB/s
Random Write BW	0.421 MB/s	0.018 MB/s	6.54 MB/s

Each bandwidth is measured by IOmeter using an access specification of its 4KB request.

The characteristics of three SSDs used in our evaluation are described in Table 2. SSD-A and SSD-B are typical low-end SSDs which satisfy Eq. (5). As discussed in sec-

tion 3.4, SSD-C does not meet the sufficient condition in Eq. (5). All SSDs have two partitions, one is 51.2 GB in size and the other 8 GB. In all experiments, ReSSD employs the first partition as the normal area and part of the second one as the reserved area. When the performance is measured on SSD alone (without using ReSSD), the second partition has not been used.

4.2 IOmeter

First, we have evaluated the sequential read/write and random read/write performance of each SSD using IOmeter. IOmeter is a widely-used benchmark tool for I/O subsystem measurement and characterization [23]. After creating a single 4 GB file, IOmeter generates one set of I/O requests to the file for five minutes and reports the average bandwidth and latency. The size of each request is either 4 KB or 8 KB. In ReSSD, the size of the reserved area is configured to 512 MB, which corresponds to 12.5% of the file size. In order to reflect the overhead associated with the mapping information lookup, we have generated enough small random writes to fill the RB-tree with about 60,000 nodes before the actual execution of IOmeter. During this warm-up period, almost half the reserved area is occupied with data, consuming 3,334 KB of kernel memory for maintaining the mapping information.

Figs. 3 and 4 illustrate the bandwidth and latency of each SSD for the request size of 4 KB and 8 KB, respectively. The results of ReSSD are normalized to those obtained with SSD alone. Observing that SSD and ReSSD show almost the same performance for sequential/random reads and sequential writes we can see that the overhead of ReSSD is very small.

On SSD-A and SSD-B, the random write performance of ReSSD is higher than the case without ReSSD. The bandwidth of the 4 KB random write is doubled at SSD-A and more than quintupled at SSD-B due to the use of ReSSD. Accordingly, the latency of the random write has been improved on both SSDs. However, the bandwidth of random writes is not improved on SSD-C since, unlike other SSDs, its ordered-sequential write performance is not superior enough to the random write performance as can be seen in Fig. 2.

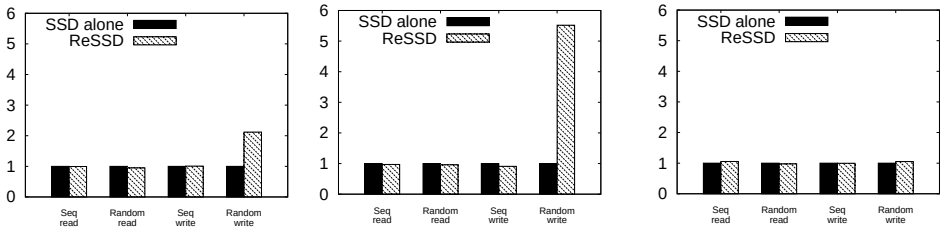
Note that during the random write test shown in Figs. 3 and 4, all write requests received by ReSSD are treated as small random writes as the request size is either 4 KB or 8 KB. Therefore, they are all written to the reserved area. From the performance point of view, this can be considered the worst case scenario to ReSSD as the mover is always active and busy in moving valid data from the reserved area to the normal area. Only on SSDs which satisfy the sufficient condition shown in Eq. (5), ReSSD can expect some performance improvement. On SSD-A and SSD-B whose random write performance is much lower than the ordered-sequential write, ReSSD can be effective in improving small random writes even with the small reserved area.

4.3 Postmark

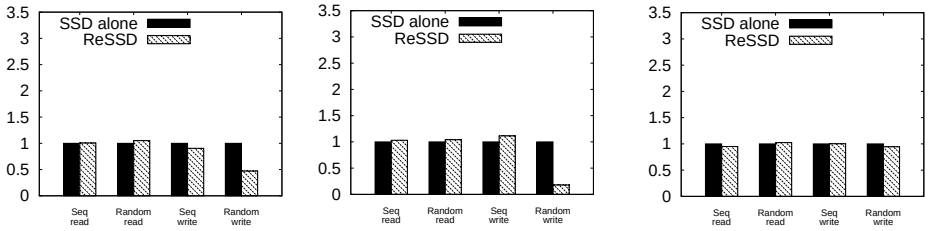
Postmark [13] is a representative file system benchmark which aims at measuring the performance of handling many small files by simulating email server workloads. Once executed, Postmark first creates a number of directories and a set of small files inside each directory. The file size is chosen from the uniform distribution between 500 B and 20 KB. Then, Postmark conducts a number of transactions consisting of file crea-

tion, deletion, read, and append operations. After the transactions are completed, all files and directories are deleted and Postmark reports the elapsed time.

We have measured the elapsed time of Postmark for three different workload configurations with and without the use of ReSSD. The number of directories, initial files, and transactions of each workload configuration is summarized in Table 3. In ReSSD, we vary the reserved area size from 0.125 GB to 0.75 GB. Since the initial file set size of each Postmark configuration ranges from 0.8 GB to 2.4 GB, the reserved area size corresponds to 5.2%-93.75% of the amount of small files created by Postmark before initiating transactions.

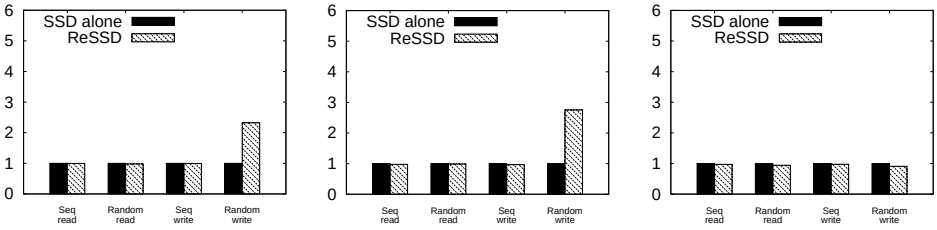


(a) Bandwidth (SSD-A). (b) Bandwidth (SSD-B). (c) Bandwidth (SSD-C).

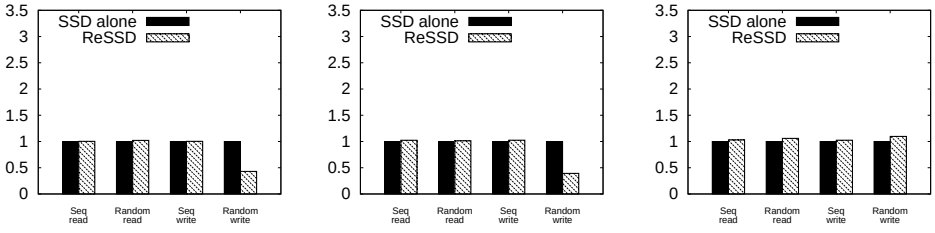


(d) Latency (SSD-A). (e) Latency (SSD-B). (f) Latency (SSD-C).

Fig. 3. Normalized bandwidth and latency of the 4KB request.



(a) Bandwidth (SSD-A). (b) Bandwidth (SSD-B). (c) Bandwidth (SSD-C).



(d) Latency (SSD-A). (e) Latency (SSD-B). (f) Latency (SSD-C).

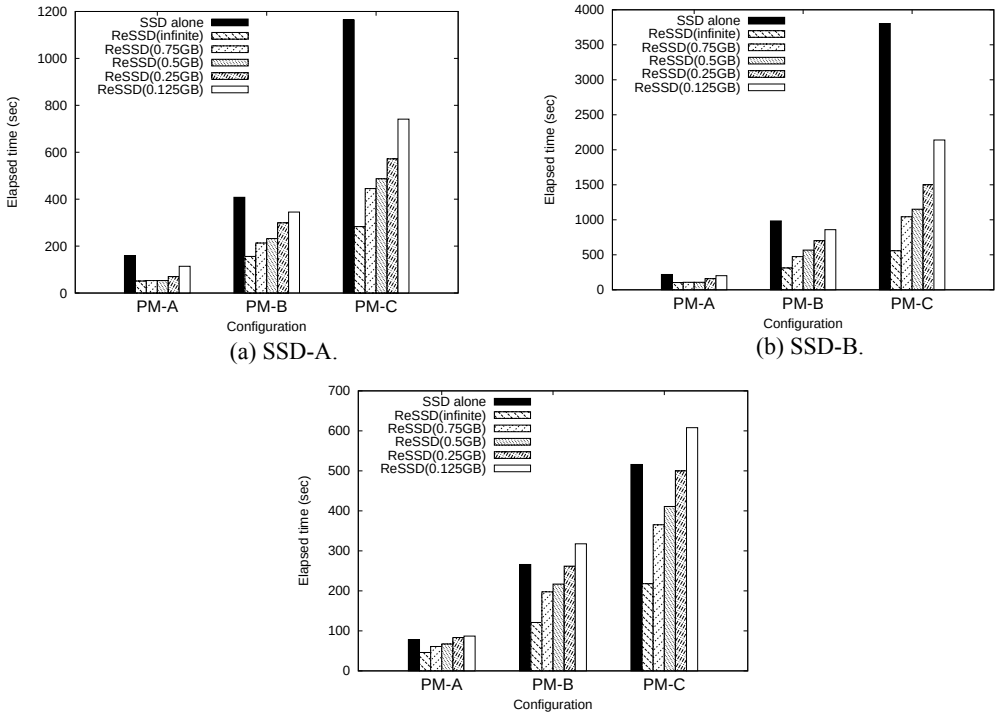
Fig. 4. Normalized bandwidth and latency of the 8KB request.

Table 3. Postmark configurations.

	Initial file set size	Initial files	Directories	Transactions
PM-A	0.8 GB	80000	10000	30000
PM-B	1.6 GB	160000	20000	60000
PM-C	2.4 GB	240000	30000	90000

Table 4. The ratio of write requests identified as small random writes of each Postmark configuration.

	PM-A	PM-B	PM-C
Ratio	11.24 %	21.61 %	25.75 %



(a) SSD-A. (b) SSD-B.

(c) SSD-C.

Fig. 5. Postmark elapsed time of each configuration on SSD alone and ReSSD.

Fig. 5 presents the elapsed time to execute each Postmark configuration. The number in parentheses for ReSSD indicates the reserved area size. The infinite size represents the case where the reserved area is large enough to make the mover remain inactive during the entire Postmark run. The performance of ReSSD with the infinite reserved area demonstrates the upper bound ReSSD can attain, as all the identified small random writes are stored in the reserved area sequentially, being never moved to the normal area. Table 4 shows the ratio of write requests which are identified as small random writes for each Postmark workload configuration. We can see that as the initial file set size be-

comes larger, the ratio of small random writes also grows.

On SSD-A and SSD-B, ReSSD outperforms the case with SSD alone in all workload configurations. With the larger reserved area size, ReSSD accomplishes more performance improvement. In case of PM-C on SSD-B, ReSSD improves the elapsed time by 43% (with 0.125 GB)-71% (with 0.75 GB) depending on the reserved area size. This improvement is close to the upper bound of 85% when the reserved area size is infinite.

As we move the workload configuration from PM-A to PM-C, ReSSD results in more significant improvement in the elapsed time. That is because the performance of SSD degrades more rapidly than that of ReSSD as the amount of small random writes increases. When the reserved area size is 0.25 GB, ReSSD shows the performance improvement over the SSD alone case by 27.8% in SSD-A and by 7.7% in SSD-B under the PM-A configuration. However, with the same reserved area size, the improvement is increased to 36.3% in SSD-A and 43.7% in SSD-B under the PM-C configuration, even though the relative ratio of the reserved area size to the initial file set size is dropped from 31% (for PM-A) to 10% (for PM-C).

On SSD-C, ReSSD does not improve the elapsed time with the reserved area size smaller than or equal to 0.25 GB. As mentioned in the previous subsection, this is due to that the ordered-sequential write performance is not fast enough to offset the overhead of additional writes caused by the mover. However, when the reserved area size is larger than 0.25 GB, there is more than 20% of performance improvement as in other SSDs. Since there is still much room for improvement even in SSD-C, we will pursue a further enhancement of ReSSD in future work.

4.4 Sysbench-OLTP Benchmark

OLTP (OnLine Transaction Processing) refers to a class of systems that facilitate and manage transaction-oriented applications, typically for data entry and retrieval transaction processing. A workload given to the storage system of such OLTP systems is known to contain lots of small random accesses. OLTP benchmarks are widely used not only for evaluating database performances, but also for estimating small random access performance of storage systems. Usually, the performance metric of this benchmark is TPS (Transactions Per Second), which refers to the average number of transactions that can be handled within one second. In this subsection, we evaluate how much the proposed approach can improve the performance of OLTP systems.

In order to measure the performance of the OLTP workload, we have used Mysql 5.0 DBMS and the OLTP test mode of Sysbench [24] for benchmarking real database performance. The OLTP test mode first creates a single table in the specified DBMS and fills it with a certain number of rows. Then, multiple threads are invoked and they conduct a specified number of transactions consisting of conventional SQL statements such as SELECT, UPDATE, DELETE, and INSERT queries. After all threads finish the transactions, Sysbench reports a TPS value and deletes the table.

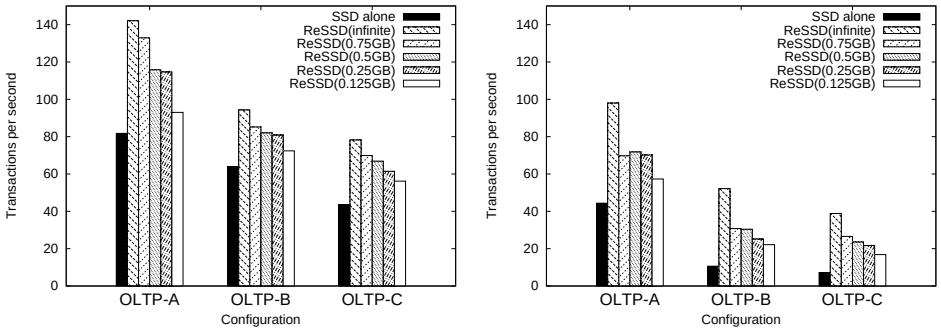
We have measured TPS values for three different configurations. The number of rows of the table and its data size are summarized in Table 5. In all configurations, the number of threads is set to 16 and they conduct evenly 70,000 transactions. As the previous evaluation using Postmark, we vary the reserved area size from 0.125 GB to 0.75 GB in ReSSD, which corresponds to 0.73%-12.93% of the table size.

Table 5. The number of rows in the table of each configuration.

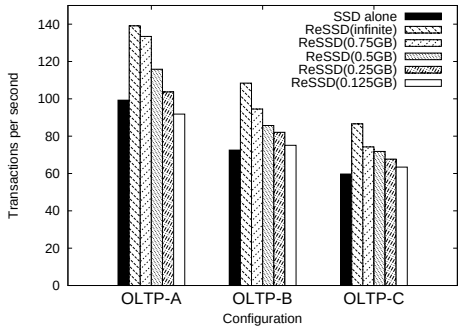
	OLTP-A	OLTP-B	OLTP-C
Rows	20,000,000	40,000,000	60,000,000
Initial table size	5.8 GB	11 GB	17 GB

Table 6. The ratio of write requests identified as small random writes of each OLTP workload configuration.

	OLTP-A	OLTP-B	OLTP-C
Ratio	85.72%	91.1%	94.12%



(a) SSD-A. (b) SSD-B.



(c) SSD-C.

Fig. 6. Transaction per second of each configuration on SSD alone and ReSSD.

Table 6 shows the ratio of write requests which are identified as small random writes for each workload configuration. As mentioned above, more than 85% of write requests are identified as small random writes and the ratio increases as the table size grows.

Fig. 6 illustrates the TPS value of each workload configuration. Overall trends of evaluation results are similar to those of Postmark described in section 5.3. On SSD-A and SSD-B, ReSSD outperforms the case with SSD alone in all configurations. Also, with the larger reserved area size, ReSSD attains more performance improvement. In case of OLTP-C on SSDA, TPS values are increased by about 29% (with 0.125 GB)-80% (with 0.75 GB) depending on the reserved area size. In particular, ReSSD on SSD-B

in the same workload configuration makes TPS about 5 times higher when the reserved area size is 0.75 GB which is only 4.41% of the table size.

Like Postmark, as the table size becomes larger, ReSSD results in more significant improvement. This tendency is especially noticeable on SSD-B whose performance is dramatically decreased as the table size grows. When the reserved area size is 0.25 GB, the TPS value of ReSSD on SSD-B is about 1.5 times higher than that of the SSD alone case in the OLTP-A configuration. However, with the same reserved area size, the TPS value is trebled with ReSSD on SSD-B in the OLTP-C configuration.

On SSD-C, only when the reserved area size is larger than 0.125 GB, ReSSD shows performance improvements over the SSD alone case. As mentioned in the previous subsection, this is because the ordered-sequential write performance of SSD-C is not fast enough compared to its random write performance. However, with the reserved area size of 0.75 GB, which is only 4.41% of the table size in case of OLTP-C, ReSSD improves TPS by more than 44% as in other SSDs.

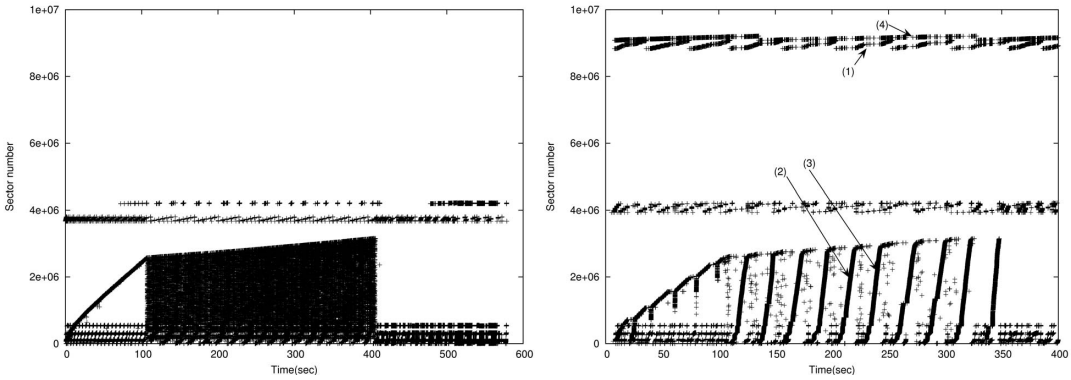
4.5 Write Request Patterns

In this subsection, we investigate the difference in write request patterns arrived at the underlying SSD using block I/O traces. Block I/O traces are obtained by blktrace, which is a block layer I/O tracing tool included in the Linux kernel.

Figs. 7 and 8 depict the write request traffic received by the underlying SSD when we run Postmark and Sysbench OLTP mode, respectively, (a) on SSD alone and (b) on ReSSD. The Postmark workload configuration of PM-B is used for this experiment. The normal area size is 4 GB and the reserved area size is 0.125 GB, which corresponds to about 7.8% of the initial file set size. The OLTP workload configuration for this experiment is OLTP-C. The normal area size is 51.2 GB and the reserved area size is 0.125 GB, which is about 0.73% of the table size. Each point (x, y) in Figs. 7 and 8 represents the LSN (y) of a single write request arrived at time x.

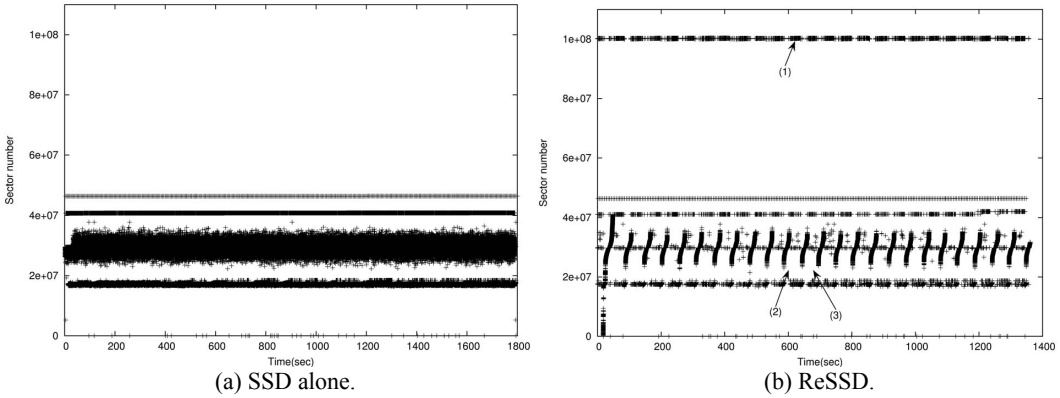
In Fig. 7 (a), we can see that write requests are randomly scattered across the wide range of LSNs when Postmark is performing various file system transactions. In Fig. 7 (b), however, we can confirm visually that ReSSD eliminates most of such small random writes. The number of scattered random writes has been significantly reduced, while many of them are converted to sequential and ordered-sequential writes. Similarly, Fig. 8 (b) shows that most of small random writes are diminished by ReSSD while, in Fig. 8 (a), there are many write requests randomly scattered across the wide range of LSNs when Sysbench is generating lots of transactions.

The points labeled with (1) in Figs. 7 (b) and 8 (b) illustrate sequential write requests generated as some of requests are identified as small random writes and they are written sequentially to the reserved area. At about 200 seconds in Postmark and at about 650 seconds in Sysbench OLTP, half of the reserved area is filled and the mover begins to move valid data from the reserved area to the normal area via ordered-sequential writes labeled with (2). The points marked with (3) represent ordered-sequential writes issued by the mover for another half of the reserved area. Note that sequential writes labeled with (4) in Fig. 7 (b) are generated when in-memory mapping information is written sequentially to the end of the reserved area to prevent potential data loss due to sudden power failures. In Fig. 8 (b), since the y-axis unit is too large, those write requests are not invisible.



(a) SSD alone. (b) ReSSD.

Fig. 7. Write request traffic of postmark executed on (a) SSD alone and (b) ReSSD.



(a) SSD alone. (b) ReSSD.

Fig. 8. Write request traffic of sysbench OLTP mode executed on (a) SSD alone and (b) ReSSD.

As mentioned in section 3.3, the request sizes of ordered-sequential writes can be larger than 8 KB since several small random writes can be merged into one ordered-sequential write. In Fig. 7, about 13% of the ordered-sequential writes are larger than or equal to 12 KB in size. They are responsible for 39% of valid data moved from the reserved area to the normal area. The rest of them are moved by 4 KB-sized or 8 KB-sized ordered-sequential writes. In Fig. 8, the ratio of the ordered-sequential writes whose size is larger than or equal to 12 KB is about 6% and they are in charge of moving 21% of valid data from the reserved area to the normal area.

As the reserved area size becomes larger, small random writes stay longer in the reserved area and the chance of getting larger ordered-sequential writes also increases. When we execute the same Postmark configuration with the 0.25 GB of the reserved area, the percentage of ordered-sequential writes whose sizes are larger than 8 KB is increased to 15.6% and they occupy 44% of valid data moved to the normal area. In case of Sysbench OLTP with the same reserved area size, about 8% of the ordered-sequential writes are larger than or equal to 12 KB in size and 27% of valid data is moved from the reserved area to the normal area by these writes.

5. CONCLUSION

In this paper, we propose ReSSD to improve small random write performance of low-end SSDs with low memory usage. ReSSD works as a virtual block device on top of SSD, which does not require any modifications of the operating system kernel and applications. Through monitoring the logical sector number and the size of incoming write requests, ReSSD identifies small random writes and converts them into sequential and ordered-sequential writes, which are more favorable to SSDs. Since the underlying SSD does not suffer from small random writes, the overall performance of SSD is improved with low memory usage under the workload including many small random writes. Through analytical modeling, we also induce a sufficient condition which would be needed to achieve performance improvement by making use of ReSSD. Our evaluation results present that the proposed approach accomplishes noticeable performance improvement on low-end SSDs under all workloads.

As future work, we plan to enhance ReSSD further to achieve more improvements, especially on SSDs with relatively good random write performance. In addition, to extend the current simple policy of identifying small random writes, it is necessary to develop more accurate analytical models which describe the relation between the transformation of small random writes into sequential writes and the amount of resulting performance improvement. Finally, as SSDs are gaining momentum in the enterprise system, we will study how to apply our approach to larger storage systems based on the RAID technology.

REFERENCES

1. S. W. Lee and B. Moon, "Design of flash-based DBMS: an in-page logging approach," in *Proceedings of ACM International Conference on Management of Data*, 2007, pp. 55-66.
2. S. W. Lee, B. Moon, and C. Park, "Advanced in flash memory SSD technology for enterprise database applications," in *Proceedings of ACM International Conference on Management of Data*, 2009, pp. 863-870.
3. S. W. Lee, B. Moon, C. Park, J. M. Kim, and S. W. Kim, "A case for flash memory SSD in enterprise database applications," in *Proceedings of ACM International Conference on Management of Data*, 2008, pp. 1075-1086.
4. Explore the features: Windows ReadyBoost, http://www.microsoft.com/windows/windows-vista/features/readyboost.aspx
5. Y. Kim, A. Gupta, and B. Urgaonkar, "MixedStore: An enterprise-scale storage system combining solid-state and hard disk drives," Technical Report CSE-08-017, *The Pennsylvania State University*, 2008.
6. I. Koltsidas and S. D. Viglas, "Flashing up the storage layer," in *Proceedings of VLDB Endow*, Vol. 1, 2008, pp. 514-525.
7. D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating server storage to SSDs: analysis of tradeoffs," in *Proceedings of the 4th ACM European Conference on Computer Systems*, 2009, pp. 145-158.

8. F. Chen, D. A. Koufaty, and X. Zhang, "Understanding intrinsic characteristics and system implications of flash memory based solid state drives," in *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems*, 2009, pp. 181-192.
9. D. Dumitru, "Optimizing flash storage with linearization software," in *Flash Memory Summit*, 2009.
10. M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Transactions on Computer Systems*, Vol. 10, 1992, pp. 26-52.
11. Y. Lee, J. Kim, and S. Maeng, "ReSSD: A software layer for resuscitating SSDs from poor small random write performance," in *Proceedings of the 25th ACM Symposium on Applied Computing*, 2010, pp. 242-243.
12. L. Bouganim, B. P. Jonsson, and P. Bonnet, "uFLIP: Understanding flash IO patterns," in *Proceedings of the 4th Biennial Conference on Innovative Data Systems Research*, 2009, http://dblp.uni-trier.de/db/conf/cidr/cidr2009.html.
13. J. Katcher, "Postmark: A new file system benchmark," NetApp Technical Report, No. TR-3022.
14. A. Ban, "Flash file system," United States Patent, No. 5,404,485 1993.
15. A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings," in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2009, pp. 229-240.
16. Intel, "Understanding the flash translation layer (FTL)," Application Note, AP-684.
17. Y. G. Lee, D. Jung, D. Kang, and J. S. Kim, "uFTL: a memory-efficient flash translation layer supporting multiple mapping granularities," in *Proceedings of the 8th ACM International Conference on Embedded Software*, 2008, pp. 21-30.
18. N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proceedings of the USENIX Annual Technical Conference*, 2008, pp. 27-70.
19. A. Rajimwale, V. Prabhakaran, and J. Davis, "Block management in solid-state devices," in *Proceedings of the USENIX Annual Technical Conference*, 2009, pp. 279-284.
20. H. Kim and R. Umakishore, "FlashLite: A user-level library to enhance durability of SSD for P2P file sharing," in *Proceedings of International Conference on Distributed Computing Systems*, 2009, pp. 534-541.
21. "Device-mapper resource page," http://sourceware.org/dm/.
22. "dm-crypt: a device-mapper crypto target," http://www.saout.de/misc/dm-crypt/.
23. "Iometer project," http://www.iometer.org.
24. "System performance benchmark," http://sysbench.sourceforge.net.

Youngjae Lee received the B.S. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 2005. He is currently working toward the Ph.D. degree in the same school. He is interested in storage systems and operating systems.

Jin-Soo Kim received the B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, Korea, in 1991, 1993, and 1999, respectively. He is currently an associate professor in Sungkyunkwan University (SKKU). Before joining SKKU, he was an associate professor at Korea Advanced Institute of Science and Technology (KAIST) from 2002 to 2008. He was also with the Electronics and Telecommunications Research Institute (ETRI) from 1999 to 2002 as a senior member of research staff, and with the IBM T. J. Watson Research Center as an academic visitor from 1998 to 1999. His research interests include embedded systems, storage systems, and operating systems.

Seungryoul Maeng received the B.S. degree in electronics engineering from Seoul National University (SNU), Korea, in 1977, and the M.S. and Ph.D. degrees in computer science in 1979 and 1984, respectively, from Korea Advanced Institute of Science and Technology (KAIST), where he has been a faculty member in the Department of Computer Science since 1984. From 1988 to 1989, he was with the University of Pennsylvania as a visiting scholar. His research interests include microarchitecture, parallel computer architecture, cluster computing, and embedded systems.