Effective Flash-based SSD Caching for High Performance Home Cloud Server

Dongwoo Lee, Changwoo Min and Young Ik Eom

**Abstract — ***In the home cloud environment, the storage performance of home cloud servers, which govern connected devices and provide resources with virtualization features, is critical to improve the end-user experience. To improve the storage performance of virtualized home cloud servers in a cost-effective manner, caching schemes using flash-based solid state drives (SSD) have been widely studied. Although previous studies successfully narrow the speed gap between memory and hard disk drives, they only focused on how to manage the cache space, but were less interested in how to use the cache space efficiently taking into account the characteristics of flash-based SSD. Moreover, SSD caching is used as a read-only cache due to two well-known limitations of SSD: slow write and limited lifespan. Since storage access in virtual machines is performed in a more complex and costly manner, the limitations of SSD affect more significantly the storage performance. This paper proposes a novel SSD caching scheme and virtual disk image format, named sequential virtual disk (SVD), for achieving high-performance home cloud environments. The proposed techniques are based on the workload characteristics, in which synchronous random writes dominate, while taking into consideration the characteristics of flash memory and storage stack of the virtualized systems. Unlike previous studies, SSD is used as a read-write cache in the proposed caching scheme to effectively mitigate the performance degradation of synchronous random writes. The prototype was evaluated with some realistic workloads, through which the developed scheme was shown to allow improvement of the storage access performance by 21% to 112%, with reduction in the number of erasures on SSD by about 56% on average.*[1]

Index Terms — Home cloud, Storage performance, NAND flash-based storage, SSD caching

[1] This work was supported by Ministry of Science, ICT & Future Planning (MSIP), Korea, under the Information Technology Research Center (ITRC) support program (NIPA-2014(H0301-14-1020)) supervised by the National IT Industry Promotion Agency (NIPA), and Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the MSIP (2010-0020730)

Dongwoo Lee is with the College of Information and Communication Engineering, Sungkyunkwan University, 2066, Seobu-ro, Jangan-gu, Suwon 440-746, South Korea. (e-mail: lightof@skku.edu)

Changwoo Min is with the College of Computing, Georgia Tech, 266 Ferst Dr NW, Atlanta, GA 30332-0765, USA. (e-mail:changwoo@gatech.edu)

Young Ik Eom is with the College of Information and Communication Engineering, Sungkyunkwan University, 2066, Seobu-ro, Jangan-gu, Suwon 440-746, South Korea. (e-mail: yieom@skku.edu)

Contributed Paper
Manuscript received 04/01/15
Current version published 06/29/15
Electronic version published 06/29/15.

I. Introduction

Virtualization is restrictively employed for personal applications, but has been widely used in many enterprise fields in the past few years. However, with the advent of the IoT (Internet-of-Things) era, recent trends show that some appliances such as smart TVs and home gateways are expected to evolve into home cloud servers [1], [2]. Cloud computing is a type of service that provides computing resources to customers on demand through virtualization technology. By accessing a cloud computing service, customers can use high-performance computing resources without buying a new computer. Applying the same idea to the home environment, home cloud servers govern all connected devices and provide them with computing resources. Therefore, users and devices can run many applications using those resources.

Despite valuable functionalities of virtualization such as multiplexing and fair management of physical resources, one of its major drawbacks is performance degradation due to virtualization latency. Constant research efforts have brought the performance of processors and memories in virtualized systems to nearly the same level as in bare-metal systems, but, unfortunately, storage systems affect the performance of applications in unanticipated ways. Kim et al. [3] also figured out that storage performance is important for end-user experience since many applications have several functionalities that depend on storage. For example, the synchronous interface in databases, which is generally used for ease of development for web caching and data indexing, incurs massive random write traffics with plenty of force-unit-access (FUA) operations. These synchronous operations significantly degrade the performance of applications running on the home cloud server since virtual storage is accessed in a more costly and complex manner than physical access. Therefore, improving the performance of storage system in the virtualized environment can expedite applications of home cloud servers.

Flash-based solid-state drives (SSD) appeared in the last several years, outperforming magnetic hard disk drives (HDD) in several aspects. They have smaller thermal footprints that allow lower power consumption, with several orders of magnitude lower latency and higher throughput, which have broadened the market share. Though flash-based SSD is roughly 20 times faster than HDD, it is not an affordable option for the replacement of storage in all computer systems, since SSD is currently about 10 times more expensive than HDD. Instead of using SSD as persistent storage, many conventional systems [4]-[6] have adopted it as a cache to fill the gap between DRAM and HDD. Since SSD caches can be applied selectively for performance-critical systems and the presence of the cache is completely transparent to the applications, this provides a cost-effective way to take advantage of SSD.

To increase the productivity of virtualized systems and improve the performance of each virtual machine (VM) in the system, SSD caching for virtualized environments has been widely studied [7], [8]. Although previous research resulted in virtualized systems that can successfully manage cache space along with multiple VMs, such studies lacked efforts to ensure efficient utilization of the cache space. The unique characteristics of the flash memory should be accounted for, in order to make SSD caching a more fascinating scheme for I/O acceleration. Kim et al. [9] pointed out the differences between flash-based and DRAM-based caching, and suggested a flash-conscious cache population scheme. Moreover, due to the negative characteristics of writes in flash memory, typical SSD caching [7], [10] is unsuitable for write-most workloads. However, since many realistic workloads involve random write patterns, mitigating the effect is important to implement effective SSD caching system.

This paper demonstrates how to effectively leverage SSD caching in virtualized environments. The goal of this work was to ensure that workloads incurring synchronous random writes, as well as read-most workloads, can benefit from SSD caching by addressing the limitations of flash memory. In order to cache those workloads in SSD, reducing the number of random writes is important since it is about ten-fold slower than the sequential write and can cause massive internal fragmentation. The fragmentation increases garbage collection costs inside the SSD by incurring more block erases and thus drastically reduces the lifetime of the SSD. The problem of random writes can be addressed by transforming them to the sequential writes similar to the log-structured file system [11]. This paper applies the similar concept to the virtual disk layer (VDL). In a virtualized environment, the VDL is a crucial component that determines the write patterns of each VM since it provides mapping between the virtual disk in the VM and image file in the host. To this end, this paper proposes a novel virtual disk image format, sequential virtual disk (SVD), which fully utilizes the SSD sequentially. Previous version of this work [12] only focused on how to utilize SSD space efficiently, but this paper includes schemes for management of the cache space. SVD with SSD caching mitigates the effect of storage on the application performance in virtualized systems, thereby improving the overall performance of the home cloud server.

The rest of the paper is organized as follows. Section II examines the characteristics of flash memory and I/O path in virtualized systems. Section III presents the design and implementation of SSD caching in detail. The evaluation results are presented in Section IV. Finally, Section V concludes this paper.

II. Background and Related Work

To help understand the design of SSD caching for improving storage performance in virtualized home cloud servers, this section will provide an overview of storage I/O path in virtualized systems and the characteristics of flash-based solid state drives.

A. Storage Access in the Virtualized Environment

In native environment, the OS kernel simply accesses the storage to process requests from applications as shown in Fig. 1(a). On the contrary, storage in virtualized environment must be accessed through more complex and costly paths than native machines as shown as Fig. 1(b). The OS kernel of virtual machines cannot access the physical hardware directly. Instead, the hypervisor captures the I/O request and processes it by software emulation. Emulation is the most traditional and straightforward method by which the hypervisor mimics common storage interfaces such as E-IDE or SCSI. Although there is no need to modify the guest OS or to install any special device drivers on the guest machine, emulation cost is very high because the hypervisor must process every hardware command by software emulation, and moreover, each command is delivered by costly mode switches between the guest and host.

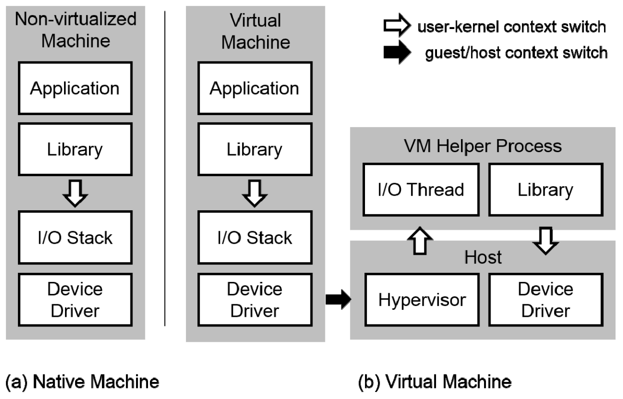

Fig. 1. Comparison of storage I/O paths in virtual machine and native machine

Recently, hardware features [13], [14] supporting direct assignment have been introduced to process the I/O request from VMs without the intervention of hypervisor. Since direct assignment allows the VM to directly access the physical device, it can improve storage performance to nearly bare-metal levels. Despite the performance advantages, direct assignment has some problems: legacy storage devices must be replaced with new ones that provide hardware-assisted virtualization features, making the cost relatively high; the hypervisor cannot provide any abstraction or management for the underlying hardware, and therefore, VM can only use entire disk partitions instead of disk images that can provide snapshots, differencing, and encryption; absence of hardware abstraction also makes VM migration difficult and the destination machine should be equipped and configured identically to the source machine. For these reasons, many hypervisors still use software-based approaches such as the para-virtualized device drivers [15]-[17]. Though these drivers can reduce the number of interventions of the hypervisor by coalescing several I/O commands, storage access using para-virtualized device drivers is still slower than native environments because the context switch remains a major source of virtualization overhead. Since many applications running on the home cloud server depend on the performance of its underlying storage, it is important to mitigate the virtualization overhead.

B. Flash Memory and SSD Caching

NAND flash is most commonly used as a memory component in SSD. It is a purely electronic device, unlike magnetic disks with mechanical moving parts to seek sectors. Though flash memory can provide uniform random access speed, it has two well-known limitations: erase-before-write and relatively small write endurance. In addition, the read and write speed of flash memory are asymmetric; it takes more time to write (or inject charge into) a flash cell than to read the status from a cell. Moreover, random write is typically about 7 times slower than sequential write, which also exacerbates the limitations of flash memory since it can cause excessive internal fragmentation. Increase in fragmentation results in frequent garbage collection inside the SSD, which incurs more block erases, and thus drastically reduces the lifespan of the SSD. To extend their capacity, modern SSDs have been equipped with multi-level cell (MLC) or even tri-level cell (TLC) flash memory, which have a tenth and a hundredth of the endurance compared to single-level cell (SLC), respectively. Consequently, they makes fragmentation more harmful to the life span of the SSD.

Many studies on the software stack from file systems [11] to flash translation layers (FTL) [18], [19] have been performed to mitigate the limitations of flash memory by transforming the access pattern, by which data written randomly is stored sequentially in the SSD. However, since the process of storage access in virtualized system is more complex than in native system, the entire access path from the VM to the storage must be examined again. Most VMs generally run unmodified OS or applications for ease of use and to ensure compatibility. Therefore, it is difficult to deal with the storage access patterns in the VM and thus they should be managed on the host-side.

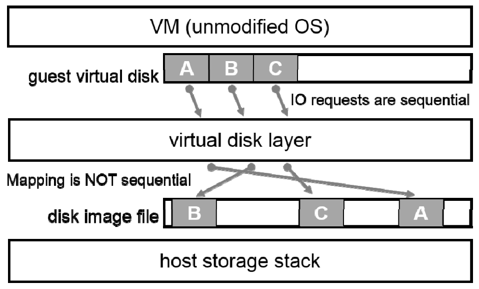

Fig. 2. Effect of virtual disk layer to the storage access patterns

Though there are many layers on which the SSD cache can be located on the host-side, S-CAVE [7] pointed out that the hypervisor is the most appropriate layer because it is the place where all I/O requests are sent from VM with information about the request including data size and offset. Specifically, virtual disk layer in the hypervisor transforms the requested sector of the virtual disk to the offset on the image file along with the corresponding virtual disk image format. For this reason, VDL has the greatest effect on the storage access patterns in the host as shown in Fig. 2. Although the random access pattern can be transformed to the sequential one in the storage stack of the host, the throughput and latency of the transformed access might be worse than those of the original sequential access due to the semantic gap between each layer. Gecko [20] also showed that poor management of the VDL significantly affects the storage access patterns, even if the patterns of the VM are sequential, because they may be mixed up by the VDL. Therefore, the VDL is chosen for placing SSD cache in order to prevent that the randomized storage access patterns degrade the effect of SSD caching.

III. Design and Implementation

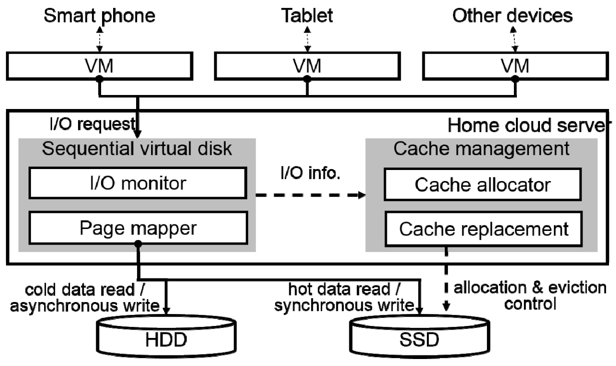

Fig. 3. The overall architecture and main components of the SSD caching system

Fig. 3 shows the overall architecture of the proposed caching system in the home cloud server. It mainly consists of two components: sequential virtual disk and cache management. Each component is designed to take account of the structure of the virtualized systems and characteristics of flash-based SSD, respectively. The design aims to provide the VM with high-performance storage in a cost-effective way, using only a cheap and small-sized SSD. To this end, this paper mainly addresses the following technical challenges:

- **Fully sequentially utilizing the SSD.** The system must be able to cache data into the SSD, even if it is randomly written by the VM, without harming the performance or lifetime of SSD.
- **Efficient management of cache space.** Since the unit of operation on flash memory varies (i.e., read/write by pages and erase by blocks), this must be considered by the system to allow efficient management of cache space.
- **Low system overhead.** SSD caching can obviously benefit storage I/O performance, but it can also harm the overall home cloud server performance, which can annoy end-users if the operational cost is too high.

In the rest of this section, the rationale behind the design decisions and key techniques of the proposed scheme will be described in detail.

A. Sequential Virtual Disk

Sequential virtual disk is a novel virtual disk image format for supporting the proposed SSD caching system. It maps the virtual disk of VM to the image file by using a mapping method similar to QCOW, except that hot data or synchronous write data can be stored into the SSD cache. QCOW maintains two-level mapping tables to translate a sector number and an offset of the virtual disk into an offset of the image file. In SVD, exploiting two-level translation, the first-level table represents the chunk index and the second-level table represents the page index. Moreover, SVD supplements a flag to the second-level table, which represent whether data is in the image file or SSD cache. It normally passes the request to the HDD, and gathers information about the request. If the request is identified as a read for hot data or synchronous write, the data is cached on the SSD. More details about the caching scheme will be discussed in Section III-C.

When a new cache space is allocated, the information in the request, including sector number and offset of the virtual disk, is translated to the chunk index and page index of the SSD cache. The current design stores the mapping table in DRAM, where it may be vulnerable to sudden power failure of the VM server, though power failure rarely happens in real situations. In future systems, this problem can be easily addressed by adopting a small amount of NVRAM for storing the table.

B. Caching on Virtual Disk Layer

In order to efficiently cache write data as well as read data, it is important to mitigate the effect of writes, especially random writes, which are more harmful to SSDs. Since populating read data is also associated with the write on the SSD cache, the limitations of SSD for writes must be addressed. Many studies [11], [18], [19] already showed that random writes can be transformed into sequential ones. This idea is applied to the VDL to use the SSD cache in a fully sequential manner. The VDL is the most suitable location for the SSD cache in virtualized systems. In a virtualized environment, a major benefit from caching the write data is improving the performance of the system and workloads running on it by exploiting the advantages of SSD, such as high throughput and low latency. VDL is the first place where an I/O request of the VM is processed, so if the host can complete the I/O request as quickly as possible, the VM can use more system resources. For example, when an application on a VM calls fsync() to flush the data written to the disk, this becomes a performance bottleneck of the application since the application must wait until the request is completely processed. Instead of passing the request to the image file in slow HDD storage through the host block I/O stack, the VM can resume the application right after caching the data into the SSD.

C. Cache Space Management

In typical virtualized systems, the cache space is shared among the VMs, and the cache manager distributes the entire cache space to each VM. To maximize the utilization of the cache space, existing caching systems [7], [8] monitor the I/O requests from VMs and make an effort to precisely predict the cache demand of each VM. In the proposed scheme, the cache space is allocated sequentially without pre-partitioning the read and write space to allow efficient management of the entire cache space. Instead of distributing the entire cache space to VMs, it is partitioned into chunks, whose size is the same as a block of the flash memory, and its space allocation is performed in the unit of chunks for each VM's request. Since the size of chunk is equal to the unit of the erase operation, the cache space can be efficiently managed with the flash-conscious eviction scheme, which will be minutely discussed in Section III-D. After the chunk is allocated to a VM, it is divided by the page size of the flash memory, and each page is consumed sequentially for each update on the cached page, preventing fragmentation on the SSD, which may be incurred by in-place updates. With this method, although massive amounts of small writes may waste space in the SSD, such situation would rarely occur because multiple writes are usually coalesced at the page cache layer in the VM.

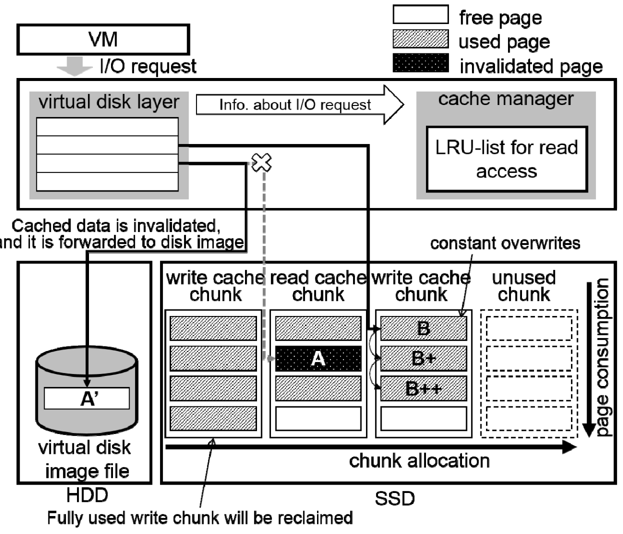

Fig. 4. An example of the caching and management process

To further understand the management process of the cache space to achieve sequential allocation, Fig. 4 represents an example of the caching and management process of the proposed scheme. When a VM requests to modify the page A cached in the SSD, the request of modification is performed directly in the disk image instead of updating the cached page. Simultaneously, the page A in the cache is invalidated and mapping for the page is redirected to the disk image. In cases where successive updates for the page B in the write cache are requested, the new pages B+ and B++ with modified data are allocated for each update.

D. Data Placement and Eviction

There are two cases in which data is cached on the SSD: hot data to the read cache and data for synchronous writes to the write cache. In order to determine the hotness of data or identify the type of writes, I/O requests of the VMs should be monitored. Information about the cache demand can also be used for eviction. There are a few options for monitoring storage access at different levels of system hierarchies. By exploiting the virtualization hierarchy, SVD can easily capture the access requests without costly monitoring overhead since all access requests in a VM are processed through the VDL.

Populating all read accesses to the cache may increase the number of pages that are never accessed until they are evicted.

Also, frequent population and eviction of the read cache may harm the lifespan of the SSD, and cause the caching system to suffer from management overhead. In order to mitigate these problems, the cache manager lazily populates data into the read cache. A variation of the LRU-like list is maintained as in the scheme by Kim et al. [9], and only the data classified as hot is promoted into the read cache. Though previous version of this work [12] adopts CLOCK algorithm [21] due to its low operation cost, other LRU-like lists [22], [23] which consider the characteristics for flash-based SSD can also be applied to the cache manager for more optimization such as reducing the number of write operations to flash memory and preventing seriously degradation of the hit ratio. Meanwhile, the write cache operates in the write-through mode. Along with resuming the execution of the VM right after the I/O request is cached, the I/O request is performed continuously in the background. This helps maintain the consistency of the image file, and thus the VM can easily be migrated without validating the image.

If there is no available chunk in the cache, the cache manager looks up the VM with the least demand and evicts all write cache chunks that are fully consumed. If there is no fully used write cache chunk, the cache manager firstly selects victim chunks among the write cache chunks, so as to maintain the read cache for a long time. If the ratio of the read cache exceeds the threshold (75% in the current prototype), chunks in the read cache can also be selected for victims. Eviction can be performed easily since allocating the cache space is performed by unit of chunks, each of which has the same size as that of the block. Evicted chunks are simply marked by the TRIM command, after which they are completely wiped internally for later use. It is also possible to use the cache utilization statistics to eagerly reclaim cache space for reduction of the on-time eviction overhead.

IV. Evaluation

A. Experimental Setup

1) Software and Hardware Configurations

To validate the proposed design for improving storage performance in the home cloud server, the caching system was implemented with QEMU [24] (version 1.5.0) and Linux KVM [25] (version 3.12.10). Though the complete caching system should include a scheme for managing and distributing the cache space, the current prototype mainly focuses on identifying the effects of sequential allocation on the SSD as a write cache. All experiments were performed on a tiny PC equipped with a low-power processor (2.1 GHz dual-core) supporting the hardware-assisted virtualization features and 8GB of DRAM. Each VM was configured to run a Linux-based mobile platform for smartphones, tablets, or other home devices in order to make the environment similar to real-world home cloud servers.

2) Target SSDs

Currently, the spectrum of SSDs available on the market is very wide in terms of price and performance. In order to cover all ranges of SSD, two very different SSDs were selected as shown in TABLE 1. The two SSDs are both based on MLC, but one is a high-end SSD (SSD-H) connected with a PCI-E bus while the other is a low-end SSD (SSD-L) connected with a typical S-ATA3 bus. With the SSD-H, the peak storage performance of caching system could be determined. On the other hand, the average gain of the storage performance was estimated for cost-effective SSD caching with the SSD-L.

TABLE I
Specification Data of the SSD Used in Experiments

	SSD-H	SSD-L
Capacity	256GB	32GB
Interface	PCI-E	SATA3
Flash Memory	MLC	MLC
Sequential Reads (MB/s)	515.3	349.0
Random 4KB Reads (MB/s)	34.5	20.3
Sequential Writes (MB/s)	450.7	134.3
Random 4KB Writes (MB/s)	64.1	14.1

3) Workloads

To study the impact of the SSD caching system on synchronous random write workloads, a synthetic workload with uniform random write patterns and real-world workload of the light-weight database was employed. For the synthetic workload, the file system level trace was collected while running RL Benchmark, and the trace for the real-world workload is collected while running Web Benchmark. Since the main area of interest was the maximum write performance, write requests in the workloads were replayed as fast as possible in a single thread, and the throughput was measured at the application level. All benchmarks were run with sufficient memory, which was configured to enable all default caching and buffering options to place the workload in the real-world situation.

B. Experimental results

First, this section explores how much SSD caching can improve workload throughput. The experiments were performed with two configurations: using all disks as a persistent storage for storing the image file (Persistent storage); using SSD as a cache managed by the proposed scheme (Cache). The performance of the HDD was measured only as a persistent storage to provide a baseline. The experimental results for the synthetic workload and the real-world workload are presented in Fig. 5 and Fig. 6, respectively.

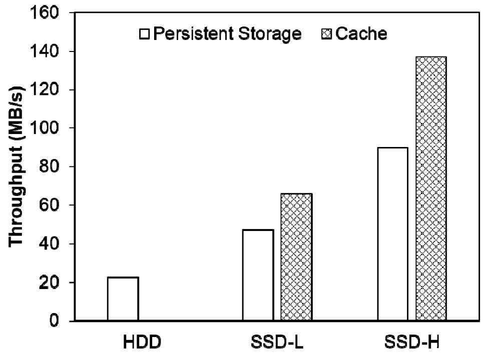

Fig. 5. Comparison of the throughput of synthetic workload

The results clearly showed that the SSDs were two to four times faster than the HDD when configured only as a persistent storage. However, the throughput of the SSD-H as a persistent storage was only 26% better than that of the SSD-L for the real-world workload, even though SSD-H has much higher throughput in its specification. This result is due to the virtualization overhead which harms the storage access patterns and limits the throughput by the costly I/O path. Although the difference in the throughput of two SSDs are 90% for the synthetic workload, the performance of the SSD-H is still restricted by virtualization.

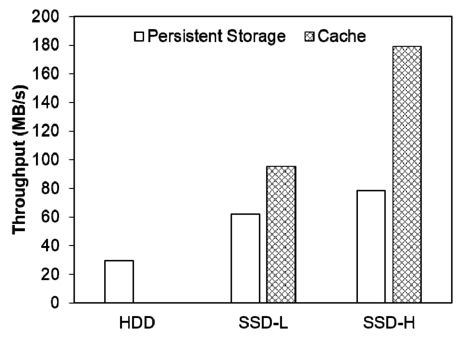

Fig. 6. Comparison of the throughput of real-world workload

On the contrary, using of the SSD as a cache results in better performance than using it as a persistent storage: 21% (SSD-L) and 33% (SSD-H) better in the synthetic workload, and 24% (SSD-L) and 112% (SSD-H) better in the real-world workload. This improvement was achieved by finishing the I/O request in the SSD cache layer and immediately returning control to the VM. In addition, the caching system fully utilized the SSDs sequentially, thereby minimizing the garbage collection overhead in the SSD. In both cases, the results show that the real-world workload benefits are greater than the case of synthetic workload. This difference was due to severe access to the storage during the operational process of Web Benchmark for caching web pages in the local storage whereas RL Benchmark only accesses the local storage through the database. Web Benchmark accesses database to index the cached data while writing significant amount of cached web pages to the local storage. Since the database frequently triggers synchronization operations to retain consistency and the pattern is random, transformation to the sequential pattern and earlier return to the VM with SSD caching is a major source of improvement.

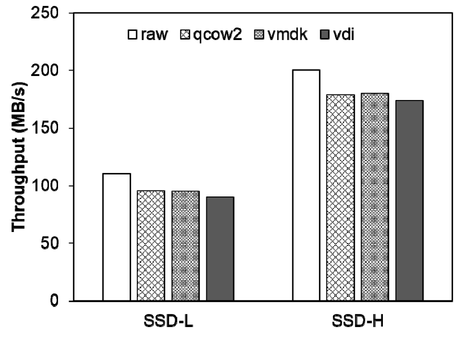

Fig. 7. Comparison of throughput with SSD caching on various disk images

To determine whether the various virtual disk image formats, including raw, qcow2, vmdk [17], and vdi [26], would result in performance differences, the same experiments were performed with the real-world workload for the various image formats using SSD as a cache. As shown in Fig. 7, since data caching is performed before converting the VM's request to the request on the disk image file, the results were similar among the image formats tested. Exceptionally, the VM with raw image format performed better due to its potentially higher throughput, which was achieved by eliminating management functionalities such as migration, snapshot, and encryption.

C. *Improvements in the Lifetime of SSD*

The next experiment compared erase counts, which are representative of the garbage collection overhead and lifespan of the flash memory inside the SSD. The I/O trace issued by the file system was collected using blktrace while running the real-world workload. The trace was run on an FTL simulator, implemented with two FTL schemes – (a) FAST [19] as a representative hybrid FTL scheme, and (b) page-level FTL [27]. In both schemes, 32 GB NAND flash memory was configured with 4 KB pages, and 512 KB blocks.

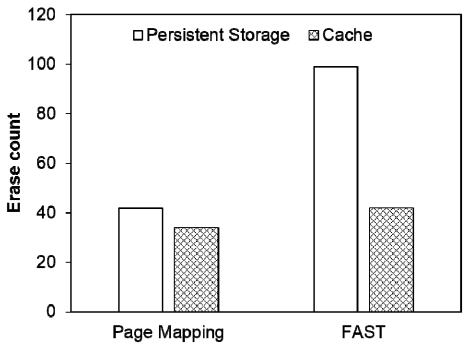

Fig. 8. Comparison of erase count with the different FTL algorithms

Fig. 8 shows the erase counts in FAST and the page mapping, while the real-world workloads were processed by SVD with SSD caching. As can be expected from the improvement of the throughput, a significantly lower number of block erases occur with caching scheme. The erase counts are significantly higher for persistent storage, which manages the image file with overwrite file systems, than in the SSD caching. In total, using SSD as a persistent storage incurred 2.2 times more block erases in FAST and 2.3 times more block erases in the page-level FTL.

V. Conclusion

This paper proposed a novel SSD caching system for improving storage performance while also improving the performance of applications in the home cloud server. Proposed SSD caching system effectively caches not only read operations but also write operations, which are generally unlikely to be cached in SSDs due to the characteristics of the flash storage. This performance improvements are achieved by fully utilizing the SSD sequentially. The experimental results demonstrated that our caching system can significantly improve the performance of random writes by transforming them to sequential ones, consequently prolonging the lifespan of the SSD through reduction in the number of erasures.

References

[1] M. R. Cabrer, R. P. D. Redondo, A. F. Vilas, J. J. Pazos Arias, and J. G. Duque, "Controlling the smart home from TV," *IEEE Trans. Consumer Electron.*, vol. 52, no. 2, pp. 421-429, Jul. 2006.

[2] H. Park, I. Lee, T. Hwang, and N. Kim, "Architecture of home gateway for device collaboration in extended home space," *IEEE Trans. Consumer Electron.*, vol. 54, no. 4, pp. 1692-1697, Nov. 2008.

[3] H. Kim, N. Agrawal, and C. Ungureanu, "Revisiting storage for smartphones," *ACM Trans. Storage*, vol. 8, no. 4, article 14, Nov. 2012.

[4] M. Canin, G. A. Mihaila, B. Bhattacharjee, K. A. Ross, and C. A. Lang, "SSD bufferpool extensions for database systems," *Proceedings of the Very Large Data Bases Endowment*, vol. 3, no. 1-2, pp. 1435-1446, Sep. 2010.

[5] T. Luo, R. Lee, M. Mesnier, F. Chen, and X. Zhang, "hStorage-DB: Heterogeneity-aware data management to exploit the full capability of hybrid storage systems," *Proceedings of the Very Large Data Bases Endowment*, vol. 5, no. 10, pp. 1076-1087, Jun. 2012

[6] M. Mesnier, F. Chen, T. Luo, and J. B. Akers, "Differentiated storage services," *in Proc. ACM Symposium on Operating Systems Principles*, Cascais, Portugal, pp. 57-70, Oct. 2011.

[7] T. Luo, S. Ma, R. Lee, X. Zhang, D. Liu, and L. Zhou, "S-CAVE: Effective SSD caching to improve virtual machine storage performance," *in Proc. International Conference on Parallel Architecture and Compilation Techniques*, Edinburgh, Scotland, pp. 103-112, Sep. 2013.

[8] F. Meng, L. Zhou, X. Ma, S. Uttamchandani, and D. Liu, "vCacheShare: Automated server flash cache space management in a Virtualization Environment," *in Proc. USENIX Annual Technical Conference*, Philadelphia, USA, pp. 133-144, Jun. 2014.

[9] H. Kim, I. Koltsidas, S. Seshadri, P. Muench, C. Dickey, and L. Chiu, "Flash-conscious cache population for enterprise database workloads," *In Proc. International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures*, Hangzhou, China, pp. 45-56, Sep. 2014.

[10] H. Jo, Y. Kwon, H. Kim, E. Seo, J. Lee, and S. Maeng, "SSD-HDD-hybrid virtual disk in consolidated environments," *in Proc. International Euro-Par Conference*, Ischia, Italy, pp. 375-384, Aug. 2010.

[11] C. Min, K. Kim, H. Cho, S. W. Lee, and Y. I. Eom, "SFS: Random write considered harmful in solid state drives," *in Proc. USENIX Conference on File and Storage Technologies*, San Jose, USA, pp. 139-154, Feb. 2012.

[12] D. Lee, C. Min, and Y. I. Eom, "Effective SSD caching for high-performance home cloud server," *in Proc. IEEE International Conference on Consumer Electronics*, Las Vegas, USA, pp. 163-164, Jan. 2015.

[13] E. Zhai, G. D. Cummings, and Y. Dong, "Live migration with passthrough device for Linux VM," *in Proc. Ottawa Linux Symposium*, Ottawa, Canada, pp. 261-268, Jul. 2008.

[14] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High performance network virtualization with SR-IOV," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1471–1480, Nov. 2012.

[15] R. Russell, "virtio: Towards a de-facto standard for virtual I/O devices," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 95–103, Jul. 2008.

[16] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, Dec. 2003.

[17] J. Sugerman, G. Venkitachalam, and B. H. Lim, "Virtualizing I/O devices on VMware Workstation's hosted virtual machine monitor," *in Proc. USENIX Annual Technical Conference*, Boston, USA, pp. 1–14, Jun. 2001.

[18] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," *in Proc. International Conference on Architectural Support for Programming Languages and Operating Systems*, Washington, USA, pp. 229-240, Mar. 2009.

[19] S.-W. Lee, D.-J. Park, T.-S., Chung, D.-H, Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Transactions on Embedded Computing Systems*, vol. 6, no. 3, article 18, Jul. 2007

[20] J.-Y. Shin, M. Balakrishnan, T. Marian, and H. Weatherspoon, "Gecko: Contention-oblivious disk arrays for cloud storage," *in Proc. USENIX Conference on File and Storage Technologies*, San Jose, USA, pp. 285–298, Feb. 2013.

[21] S. Jiang, F. Chen, and X. Zhang, "CLOCK-Pro: An effective improvement of the CLOCK replacement," *In Proc. USENIX Annual Technical Conference*, Anaheim, USA, pp. 323-336, Apr. 2005.

[22] S. H. Park, D. Jung, J. U. Kang, J. S. Kim, and J. Lee, "CFLRU: A replacement algorithm for flash memory," *in Proc. International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, Seoul, Korea, pp. 234-241, Oct. 2006.

[23] J. Cui, W. Wu, Y. Wang, and Z. Duan, "PT-LRU: A probabilistic page replacement algorithm for NAND flash-based consumer electronics," *IEEE Trans. Consumer Electron.*, vol. 60, no. 4, pp. 614-622, Nov. 2014.

[24] F. Bellard, "QEMU, A fast and portable dynamic translator," *in Proc. USENIX Annual Technical Conference*, Anaheim, USA, pp. 41-46, Apr. 2005.

[25] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: The Linux virtual machine monitor," *in Proc. Ottawa Linux Symposium*, Ottawa, Canada, pp. 225-230, Jul. 2007.

[26] J. Watson, "VirtualBox: Bits and bytes masquerading as machines," *Linux Journal*, vol. 2008, no. 166, article 1, Feb. 2008.

[27] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for compact flash systems," *IEEE Trans. Consumer Electronics*, vol. 48, no. 2, pp. 366–375, May. 2002.

BIOGRAPHIES

Dongwoo Lee received his B.S. degree in Computer Engineering from Sungkyunkwan University, Korea in 2010 and his M.S. degree in mobile Systems Engineering from Sungkyunkwan University in 2012. He is currently a Ph.D. candidate in the Department of IT Convergence at Sungkyunkwan University. His research interests include virtualization, cloud computing, storage systems, embedded systems, mobile platform and operating systems.

Changwoo Min received his B.S. and M.S. degrees in Computer Science from Soongsil University, Korea in 1996 and 1998, respectively, and his Ph.D. degree in Mobile Systems Engineering from Sungkyunkwan University, Korea in 2014. From 1998 to 2005, he was a research engineer at the Ubiquitous Computing Laboratory (UCL) of IBM, Korea. From 2005 to 2014, he was a principal software engineer at Samsung Electronics. He is currently a postdoctoral researcher at Georgia Tech., USA. His research interests include embedded systems, storage systems, and operating systems.

Young Ik Eom received his B.S., M.S., and Ph.D. degrees in Computer Science and Statistics from Seoul National University, Korea in 1983, 1985, and 1991, respectively. From 1986 to 1993, he was an associate professor at Dankook University in Korea. He was also a visiting scholar in the Department of Information and Computer Science at the University of California, Irvine, from Sep. 2000 to Aug. 2001. Since 1993, he has been a professor at Sungkyunkwan University in Korea. His research interests include virtualization, operating systems, cloud systems, and system securities.