Enhanced Hypervisor-based SSD Cache with Dynamic Cache Scanning and Allocation for Virtualized Cloud System

Hee Jung Park
College of Information and Communication Engineering, Sungkyunkwan University, 2066 Seoburo, Suwon, 440-746, *Republic of Korea*
E-mail: hjhjpark@skku.edu

Kyung Tae Kim
College of Information and Communication Engineering, Sungkyunkwan University, 2066 Seoburo, Suwon, 440-746, *Republic of Korea*
E-mail: kyungtaekim76@gmail.com

Man Yun Kim
College of Information and Communication Engineering, Sungkyunkwan University, 2066 Seoburo, Suwon, 440-746, *Republic of Korea*
E-mail: benimaru82@skku.edu

Hee Yong Youn
College of Information and Communication Engineering, Sungkyunkwan University, 2066 Seoburo, Suwon, 440-746, *Republic of Korea*
E-mail: youn7147@skku.edu

Abstract

Recently, SSD caching is widely studied for VM-based systems. In this paper we propose a novel hypervisor-based SSD caching scheme, employing a new metric to accurately determine the demand on SSD cache space of each VM. Computer simulation confirms that it substantially improves the accuracy of cache space allocation compared to the existing schemes. It also allows comparable hit ratio as the existing schemes with less amount of SSD cache for the VMs.

Keywords: Cache allocation, SSD Caching, Virtual Machine, Hit Ratio, Hypervisor

1. Introduction

In recent years virtualization technology enables multiple virtual machines (VMs) to be run on a physical machine, where each VM is run independently on its own operating system. Virtualization technology has been adopted in various IT fields because of its ability to improve the utilization of hardware resource, achieving low-power consumption, simplifying server management, and reducing maintenance cost. In a typical VM environment, multiple VMs are simultaneously run on the same host.

In VM environment, high-performance storage systems are in high demand especially for data-intensive computation. Most storage systems, even those specifically designed for high-speed data processing, are still built on conventional hard disk drive (HDD) of several long-lasting limitations including low random access performance and high power consumption. Unfortunately, these problems essentially stem from the mechanical nature of HDD, and thus they are difficult to be addressed with the same type of drive. Flash memory-based Solid State Drive (SSD) is an emerging storage technology which plays a critical role in revolutionizing the design of storage system. Different from HDD, SSD are completely built on semiconductor chips without any moving parts. Such fundamental difference makes SSD capable of providing one order of magnitude higher performance than HDD, and allows it to be an ideal storage medium for building high-performance shared storage system. Due to the advantages, SSD caching has been widely studied in conventional systems [1], [2]. However, it is impractical to directly apply the existing cache management solutions to the VM systems since the SSD caching scheme needs to maximize the utilization of shared SSD cache while ensuring the isolation of the operations of the VMs.

The previous works on cache partitioning for VMs focus on the identification of the demand on cache space of each application. For example, some researches proposed to monitor the number of hits/misses of each cache unit, and then use the data as a basis for computing the space demand [3]. Some studies proposed to use the change of hit ratio at main memory level during a time window as a metric used to predict the space demand [4]. However, the hit ratio-related techniques cannot be effective for shared cache due to the filtering effect of higher-level caches. For this reason, a cache space reallocation scheme based on random sampling was proposed in [9]. Here the space allocation for a VM is determined by the memory utilization and maximum/minimum memory quota predefined by the system administrator. The VM of lower memory utilization has a smaller share reclaimed, and thus it is more likely to get the requested memory. Here the memory requirement of a VM can be predicted in some limited condition. In addition, the prediction is inaccurate because of the mechanism of random scanning of the blocks.

In this paper a hypervisor-based SSD caching scheme is proposed, which effectively manages the cache in the VM environment by collecting and exploiting the runtime information from both the VMs and storage devices. Due to the unique position of hypervisor between the VMs and hardware devices, it does not require any modification of guest OS, user applications, or the underlying storage systems. The proposed scheme uses a new metric called "*HLP* (Hit ratio for Logically Partitioned blocks)" to identify the cache space demand of each VM at runtime. In essence, *HLP* is the ratio of the cache being used by a VM to the total cache space allocated to it. It is a critical reference for cache space allocation. Computer simulation reveals that the proposed scheme improves the accuracy of the estimation of the demand on cache space, and more effectively uses the cache space compared with the existing schemes.

The rest of the paper is organized as follows. In Section 2 the existing schemes related to hypervisor-based SSD caching are explained. The proposed scheme is presented in Section 3, and Section 4 compares its performance with the existing methods using computer simulation. Finally, Section 5 gives the conclusion and future work.

2. Related Work

2.1 Hypervisor-based SSD Caching

Recently, various hypervisors or VM monitors are run on top of physical machines which schedule the execution of VMs. Here multiple instances of operating systems may share the virtualized hardware resources. Among them, Xen hypervisor is a popular abstraction layer existing between the guest domains and physical hardware, and it is responsible for resource allocation and isolation. Here the LRU (Least Recently Used) policy is usually employed to remove data from the cache. Some studies track the count of block accesses to identify frequently used blocks, and then cache them in SSD [4], [5].

The caches come in two varieties. Firstly, dedicated deployment such as memcached where each node is allocated a fixed amount of memory along with opportunistic caches such as Linux buffer and page cache that can expand to consume underutilized memory. With opportunistic cache a process of a VM may greedily consume the memory pages if there is no other process inside the VM using the memory. However, there might be another VM of higher priority on the same host which could make better use of the memory. Then efficient use of virtual resources becomes difficult, and a new approach needs to be adopted. Meanwhile, dedicating a fixed memory region to a memcached node is convenient for offering a predictable level of quality of service. Also, UniCache allows flexible allocation of storage resource to the operating system and applications by offering a unified caching service at the hypervisor level as shown in Fig. 1.

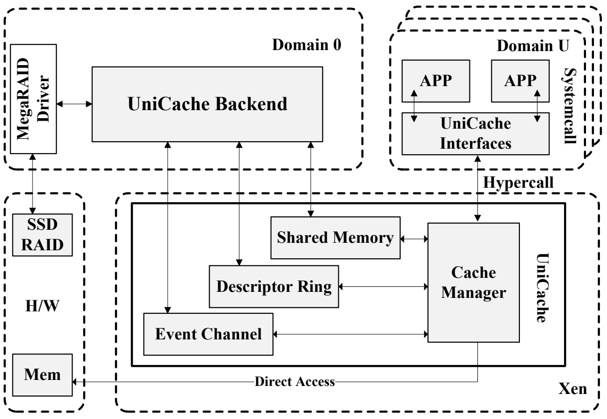


Fig. 1. The structure of UniCache store with hypervisor and SSD RAID.

Here data are split between hypervisor controlled main memory and flash memory to provide varying levels of performance based on the application type and priority of the VM. Expanding the cache to include both memory and SSD allows a much larger amount of data to be stored, which is very important in virtualized environment where competing VMs need to make efficient use of limited memory resources [6], [7], [8]. There exist three different approaches for using SSD cache in VM environment. Firstly, SSD is directly managed by the hypervisor, where the management center is located between the VMs and hardware resources. The VM-based SSD caching has significant disadvantages such that the guest OS or user application needs to be modified to manage the cache. This incurs extra burdens on the users, and is hardly available for legacy systems. The storage-based SSD caching shows a drawback in its isolated design approach. The block interface between the storage system and the virtualization software stack is primitive without the ability to deliver rich semantic information. Local optimization in the storage subsystem may not enhance the performance of overall VM system.

This paper employs hypervisor-based SSD caching to avoid the limitations in VM-based and storage-based SSD caching, while retaining their advantages. The hypervisor can manage SSD cache for the VMs in a transparent way, addressing the problem of modifying guest OS or application. Here the VM activities, particularly I/O requests are managed by the hypervisor which collects critical information required for effective management of SSD cache. With the full access privilege to hardware resources, the hypervisor can directly enforce the space allocation decisions to maximize resource utilization in an efficient way [9].

2.2 Cache Partitioning

At present, a number of researchers have proposed flash-based buffer management algorithm for SSD such as CFLRU [22] and improved CRLRU [23]. Taking advantage of the asymmetry in flash read-write performance, CFLRU is a kind of buffer replacement strategy which first replaces read-only pages and assumes that write cost of flash is far greater than read cost. Its key idea is dividing LRU linked list into two parts: working area and replacement area. Once cache is full and some data need to be replaced to outside, CFLRU chooses read-only pages for replacement according to LRU supposing that there exists read-only data in the replacement area. When there are only dirty pages in the replacement area, the ones at the tail of the linked list are replaced. Other researchers improved the traditional LRU and LFU policy to accommodate diverse requirements of the applications [17].

Note that there exists a high potential for SSD to be widely employed in large-scale cluster storage system. SSD is more expensive than traditional HDD, but performs better for random reads and writes. S-CAVE [9] is a flash cache partitioning scheme which tries to maximize cache utilization for multiple VMs on a single VMware host by running a hypervisor module. Based on the identification of runtime working set, S-CAVE monitors the changes in locality, especially transient bursts in data reuse. Here the performance of caching is measured by the number of cache hits an application encounters. If proper data blocks are cached, the number of cache hits will increase, and accordingly the effectiveness of caching. Therefore, an efficient algorithm needs to be employed to maximize the number of cache hits.

Fig. 2 illustrates the structure of S-CAVE. For each VM, S-CAVE launches a module, called Cache Monitor, to manage the allocated cache space and keep the cache transparent to the VM. In order to effectively allocate the shared SSD cache space among multiple

VMs, S-CAVE also uses a central control, called Cache Space Allocator, to analyze the information on cache usage collected from each cache monitor and make the decision on cache allocation accordingly.

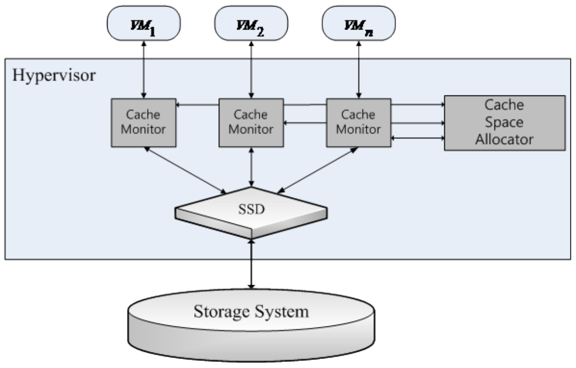


Fig. 2. The structure of the S-CAVE scheme.

The key idea of S-CAVE is to effectively allocate an appropriate amount of cache space to each VM. For this, S-CAVE identifies the demand of cache space of each VM, and each cache monitor is required to provide accurate information on the demand of the SSD cache space of the VM it monitors. Then cache space allocation is made considering the demands of all VMs. While satisfying the demand on the cache of each VM represented by the ratio of used cache space, the cache space allocation of all VMs needs to be properly balanced. The proposed scheme efficiently resolves this issue as shown next.

2.3 Accuracy of Cloud monitoring

A distributed monitoring system is required for the cloud system to operate properly. Above all things, the accuracy is important for cloud monitoring system since it can seriously influence the operations that make use of the monitored information. For example, when the monitoring system is used for measuring the performance, inaccuracy in the measure may lead to incorrect identification of the bottleneck. The monitoring system used for controlling the operations of the cloud system, accurate monitoring is necessary to effectively and efficiently identify the status of the components.

The analysis of the literature reveals two main issues related to the accuracy of monitoring system in cloud environment. The first one is related to the workload used to perform the measurements to monitor the cloud system, especially when using active monitoring approach. Here it is necessary to apply a suitable stress. The second issue is related to the virtualization technique used in the cloud where the measurement error is imputable to the virtualization system that adds additional layers between applications and physical resources. For example, time-related measurements are impaired by the sharing of physical resources such as CPU, cache, and memory. As for the workload, the research efforts in this area comprise the characterization of real workloads, the reproduction of the workload in the cloud which provides the clue on the parameters to measure.

The metrics considered in the previous works include disk throughput, VM startup time, jitter and loss of the network, memory throughput, server throughput and money cost, etc. The earlier work identified a number of limitations of the benchmarks [26], [27], [28], [29]. In particular, they suggest that various aspects such as scalability, peak load, and fault tolerance are not adequately considered by the current state-of-the-art benchmarks such as TPC-H for On-line Transaction Processing (OLTP), TPC-C for On-line Analytical Processing (OLAP), or TPC-W for e-commerce applications [30]. They also propose a number of other tests and parameters required to evaluate important aspects of modern clouds. As for the impact of virtualization on measurement accuracy, the works in the literature analyzed the accuracy of Round Trip Time (RTT), jitter, capacity and available bandwidth, topology, and also the performance of auto-tuning applications. Regarding these metrics, the issue is with accurate time stamping at the measuring nodes. Implementing VMs at the end nodes requires a timely

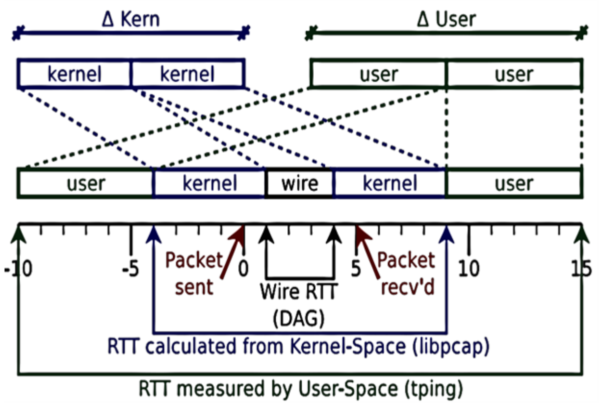


Fig. 3. The measurement of RTT [33].

scheduling and switching mechanism between different VMs. As a consequence, the packets belonging to a specific VM may be queued until the physical system switches back to that VM, which leads to inaccurate time stamping. Some works reported that accurate RTT measurements are possible only under low network and computing loads, and that most delay is introduced while sending packets (as opposed to receiving packets) [31], [32], [33], [34]. They conclude that kernel-space timestamps are not accurate enough under heavy network load, and access to timestamps as seen by physical network interfaces would be necessary to overcome this issue. Regarding the measurement of topology, [31] proved that network virtualization generates several virtual topologies on top of a single physical topology, and common active measurement tools like trace route are unable to discover the real physical topology. Moreover, their accuracy is affected by the migration of nodes, which dynamically modifies the placement of virtual nodes and the distance among them. Finally, regarding the performance of auto-tuning applications, [34] showed that the combination of ATLAS auto-tuning and Xen para-virtualization delivers native execution performance and nearly identical memory hierarchy performance profiles. Moreover, they showed that para-virtualization has no significant impact on the performance of memory-intensive applications, even when memory becomes a scarce resource [35].

3. The Proposed scheme

The environment of VM is highly dynamic, where multiple VMs of different and changing cache demands are run on a host. To rapidly reflect the runtime dynamics and guarantee effective and fair sharing of cache space, a dynamic control mechanism is proposed to periodically cross-compare the cache demand of each VM and adjust the space allocation accordingly. Through this, the VMs of increasing demand is granted more cache space, while some portion of already allocated cache space of those of decreasing demand is deprived. In order to achieve fine adjustments, the previous decisions are also taken into account as a feedback when a new decision is made.

The proposed scheme consists of two steps. The first step is to estimate the value of *HLP* identifying the demand of each VM on the cache space. Cache space reallocation is made in the next step, considering the demands of all the VMs.

3.1 Assessment of Cache Utilization

This paper proposes a new metric called *HLP* which is the ratio of the size of cache space being used to the allocated cache space during a time window. For a specific VM for which *n* blocks have been allocated, if *m* unique cache blocks have been accessed within a time window, then *HLP* is obtained by Eq. (1).

$$HLP = \frac{m}{n} \times 100\% \tag{1}$$

To accurately and efficiently estimate the *HLP* value at runtime, two counters, C_i and C_i^d, are manipulated for each VM, where C_i is the total number of cache blocks allocated to VM_i and C_i^d is the number of unique cache blocks used by VM_i. At the beginning of a time window, both counters are set to 0, and the metadata of the blocks residing in the global pool is scanned to update the counters. Since the global pool contains all the blocks allocable to different VMs, only the counter corresponding to the accessed block for the VM is incremented by one. Whenever, C_i^d is incremented, C_i is also incremented by one. Fig. 4 shows how the proposed counters are manipulated.

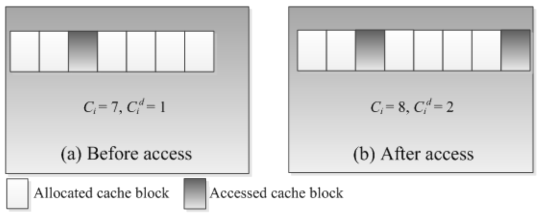


Fig. 4. An example of counter manipulation for VM_i.

In order to obtain accurate *HLP* value, all reference counters need to be scanned for the given time window. A small time window enables quick adjustment of cache allocation, but it is impossible to complete the scan of entire counters during the window. A large time window allows to finish the scan, but cache allocation becomes less responsive to the runtime dynamics. As a result, selecting an appropriate window size is an important issue in achieving the best performance.

In order to efficiently estimate the *HLP* value while rendering a high accuracy, a new sampling mechanism

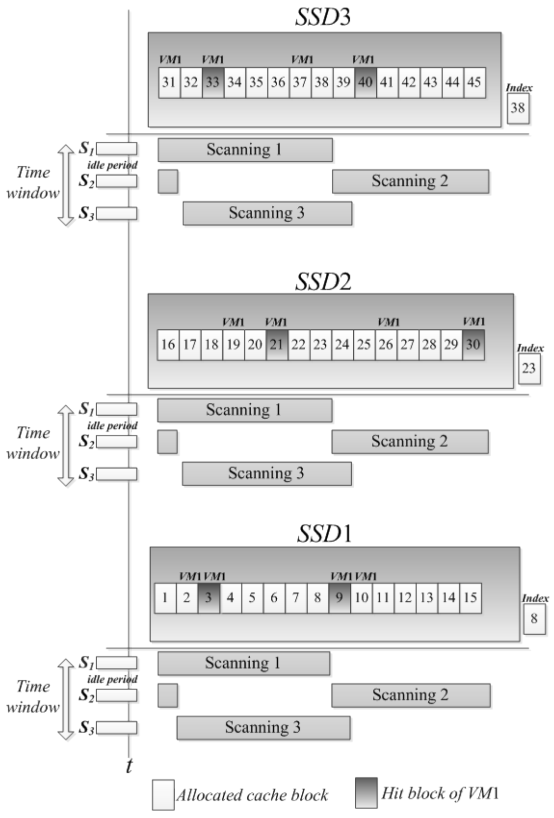


Fig. 5. An example of scanning for the estimation of *HLP*.

is adopted in this paper. Here a time window is split into multiple small sampling periods, and an idle period is inserted between two consecutive sampling periods. This approach is to reduce the computation overhead while increasing the effectiveness of the scan. Note that the change in the access during the idle period can be counted in the subsequent sampling period, which increases the accuracy of *HLP*. Within each sampling period, the scanning begins from the block where the previous scan ends. The scanning operation stops when the sampling period expires. Fig. 5 shows an example of the operation of the proposed scheme obtaining the *HLP* values. Note here that the physically tied SSD is logically partitioned, and the demand on cache space of each partition is monitored during each time window.

The proposed scheme is different from the existing ones in the management of the scan which allows accurate estimation of the demand on the cache for each VM and cache space allocation based on it.

The value of *HLP* for j^{th} partition of i^{th} time window, HLP_i^j, having k sampling periods is the average value of k samples, $Sample_i^j$ (i=1,…,k) which can be formulated by Eq. (2).

$$HLP_i^j = \frac{1}{k}\sum_{i=1}^{k} Sample_i^j \tag{2}$$

Assume that current time window is i. HLP_i denotes the final *HLP* of i^{th} time window covering all the partitions. Then,

$$HLP_i = \frac{1}{p}\sum_{j}^{p} HLP_i^j \tag{3}$$

where p is the number of partitions. In the example of Fig. 5 the SSD is partitioned into three parts, SSD1, SSD2, SSD3, and the time window consists of three scanning periods. Each partition contains 15 blocks, and VM_1 is allocated 12 blocks. The '*index*' in each SSD partition points the block number scanned last in the previous sampling period. In Fig. 5, it is assumed that S_1 has just been finished. To further improve the accuracy, the current *HLP* value is averaged with the two recent *HLP* values using a weight parameter a, enabling small time window to be more responsive to the change in the space demand. The final HLP_i is obtained by Eq. (4). In this paper a is assumed to be 0.8. Observe from the figure that six blocks were accessed out of 12 blocks for VM_1, and thus its *HLP* in this time window is 1 second.

$$HLP_i = HLP_i^j \cdot a + \frac{(HLP_{i-2} + HLP_{i-1})}{2} \cdot (1-a) \tag{4}$$

Algorithm 1 shows the procedure for the estimation of *HLP*.

Algorithm 1 Estimation of *HLP*

```
 1: //Initially setting
 2: Index block = 0
 3: Sequential start block = Index block
 4: Totally running $VM$s = $N$
 5: Allocated blocks of $VM_i$ = $VMC_i$
 6: Hit blocks of $VM_i$ = $VMC_i^d$
 7: Number of partitions = $p$
 8:
 9: for each $VM_i$ ($1 \le i < N$) do
10:    $VMC_i = 0$
11:    $VMC_i^d = 0$
12: end for
13:
14: //Calculate $HLP$ of a time window
15: for each $p$ do
16:   for $S_l \le S_k < k$ do
```

```
17:     if Scanned block = Allocated block
18:         $VMC_i$ ++
19:     else if Scanned block = Hit block
20:         $VMC_i^d$ ++
21:     end if
22:    $HLP = VMC_i^d / VMC_i + VMC_i^d * 100$
23:    $HLP_i^j += HLP / k$
24:    Index block = Last seek block
25:  end for
26:  $HLP_i = HLP / p$
27: end for
28:
29: $HLP_i = HLP_i^j * a + (\mathrm{HLP}_{j-2} + \mathrm{HLP}_{j-1}) / 2 * (1-a)$
```

3.2 Allocation of Cache

In cloud computing the available cache space is allocated to the cloud applications. The cache space is provided on demand in a fine-grained, multiplexed manner. Here the cache space allocation is based on the infrastructure as a service (IaaS).

Resource fragmentations occur as the resources are continuously allocated and deallocated. Even though the entire amount of available resource is enough to satisfy the need, it cannot be allocated to the requesting application due to fragmentation. The proposed cache allocation scheme considers the issue of scarcity of resources because the resources are usually limited while the demand is high in the hypervisor-based cloud environment. A dynamic allocation scheme is thus adopted to solve the problem [10].

Each time the cache space of a VM is adjusted, the amount of change is determined by the parameter *AlloCache*, which is the average number of blocks the VM can access within a time window. It is obtained by averaging the number of accesses in the previous time windows. VM_{min} and VM_{max} denote the VM of the smallest *HLP* and largest *HLP*, respectively. In other words, *AlloCache* of VM_{min} is the rate of missed accesses for a VM in the recent time window. The purpose of using the parameter is to ensure that the new data accessed by VM_{max} in the subsequent time window can be accommodated using the new available cache space released by VM_{min}. The proposed scheme finds VM_{max} to increase its cache space, and the amount of increase is determined by *AlloCache* of VM_{min}. Algorithm 2 below explains how to allocate the cache space. In case the total free space is smaller than 5%, the cache space of VM_{min} is swapped with that of VM_{max} for maximizing the utilization of the total space of SSD [11]. Additionally, the proposed scheme applies the CLOCK-based cache replacement approach to be more adaptive. The CLOCK algorithm captures the information on cache access and exploits the frequency of cache access via the reference bit unlike LRU [12].

In Algorithm 2, the dynamic cache allocation scheme based on *HLP* is presented. With the *HLP* identifying the demand on the cache of a VM, the proposed scheme effectively balances the allocation between the VMs considering the availability of hardware resources. For this, the configuration information including the amount of allocated cache of the VMs is utilized. Here *Cache*(VM_i) denotes the amount of cache allocated to VM_i.

Algorithm 2 Allocation of Cache Space

```
 1: Totally running VMs = $N$
 2: if $N = 1$ then
 3:     Cache ($VM_0$)= $\infty$
 4:     exit
 5: end if
 6:
 7: if Total free space > 5% then
 8:     for $VM_i$ ($1 \le i \le N$) do VM Cache ($VM_i$) = upper limit
 9:     end for
10: end if
11:
12: Calculate HLP
13: Sort { $HLP(VM_i) \mid 1 < i < N$ }
14:
15: if Total free space < 5% then
16:     Swap(Cache ($VM_{max}$), Cache ($VM_{min}$ ))
17: end if
18:
19: Cache ($VM_{max}$) + = AlloCache ($VM_{max}$ )
20: Cache ($VM_{min}$ ) – = AlloCache ($VM_{min}$ )
```

4. Performace Evaluation

In this section the performance of the proposed scheme is evaluated which uses *HLP* to correctly identify the demand on the cache of the VMs and thereby reallocate the cache space when managing multiple VMs. The simulation was conducted with a PC of Intel 3.5Ghz i3 CPU, 16GB RAM, 64bit Window 7, and the simulation code was written in C++ language with Visual studio 2010. The size of SSD cache was gradually increased from 1,000 to 10,000 blocks for the evaluation. The effectiveness of the proposed scheme is compared with that of S-CAVE in terms of cache space utilization and hit ratio. In the simulation the length of time window, sampling period, and idle period are set to be 1 sec, 0.2sec, 0.1 sec, respectively.

4.1 Accuracy

First, one VM is used to run a single workload each time. The VM starts with a small size cache which will then be dynamically adjusted during runtime. Fig. 6 compares the accuracy of the proposed scheme with S-CAVE. The accuracy is the ratio of the number of blocks hit during a time window to the number of blocks allocated to the VM. Here the workload is proj_4 from SNIA IOTTA Repository [24]. Observe from the figure that the proposed scheme reflecting the cache usage of entire window allows consistently better accuracy than S-CAVE. Notice that the average accuracy with the proposed scheme is consistently higher and more stable than that with S-CAVE.

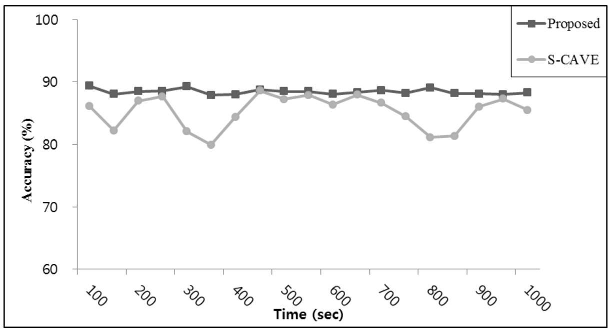


Fig. 6. The comparison of accuracies of the schemes.

4.2 Cache Allocation

The evaluation on cache utilization with multiple VMs running with shared SSD cache is presented here. Fig. 7 shows that the proposed scheme considerably reduces the amount of SSD cache space allocated to the VMs. It can be deemed that the proposed scheme uses SSD cache space more efficiently than the other scheme. Fig. 8 compares hit ratio of the schemes. Both the proposed scheme and S-CAVE display similar hit ratio.

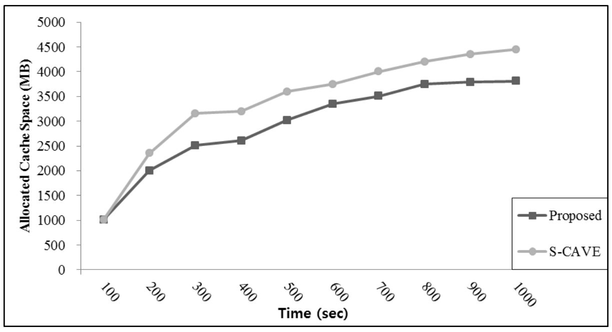


Fig. 7. The comparison of cache space allocated to each VM.

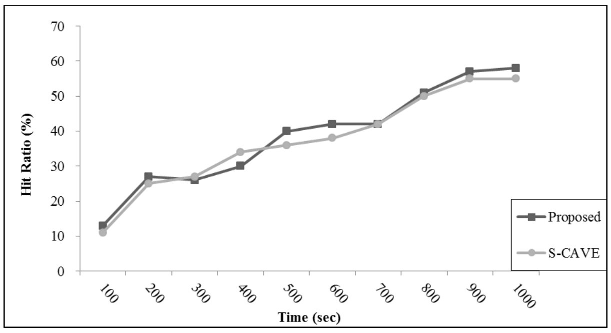
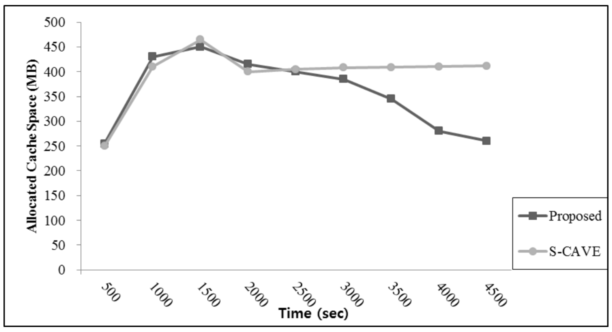
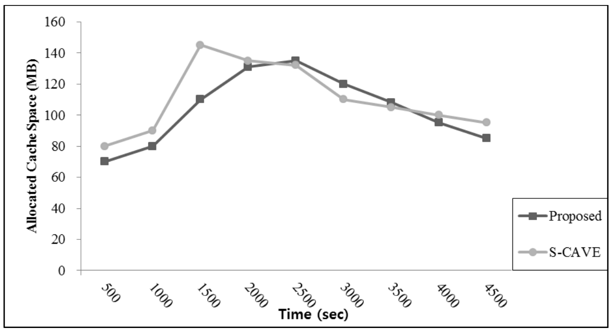


Fig. 8. The comparison of hit ratio of different schemes.

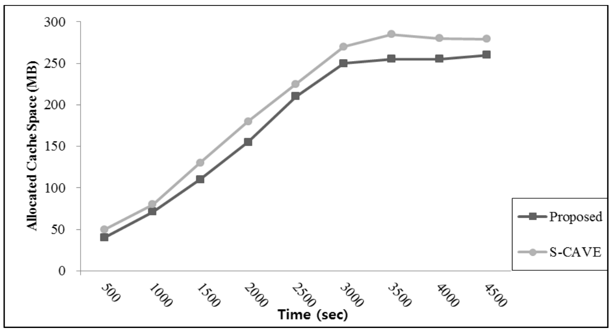
Two additional real-world traces and one benchmark are employed to evaluate the proposed scheme, which are MSR Cambridge trace from SNIA IOTTA Repository [24], UMass Trace Repository [25] from a search engine, and TCP-H benchmark [30]. The traces represent a variety of workloads, hm_1 (hardware monitoring) and Websearch. Also, tpch_q9 is used to collect the traces forming long running query. They generate I/O accesses at the storage disk tier and account for SSD cache as well as application caching effect. Fig. 9 shows that the proposed scheme reduces the amount of SSD cache space compared to S-CAVE.



(a) hm_1



(b) Websearch



(c) tpch

Fig. 9. The comparison of allocated cache with different schemes.

5. CONCLUSION

In this paper a hypervisor-based SSD caching scheme has been presented, which effectively manages the SSD storage cache in the VM environment by properly collecting and exploiting the runtime status of the VMs. A new metric was employed to accurately identify the demand on the cache space of each VM. The ratio of available cache space has also been accounted for dynamic adjustment of cache allocation among the VMs. Computer simulation validates the effectiveness the proposed scheme in achieving higher accuracy in the estimation of cache space for the VMs compared to the existing scheme. This allows the proposed scheme to display comparable hit ratio with less amount of SSD cache.

As future study, the proposed scheme will be enhanced for effective allocation with clustered SSD cache. Thereby, it will be able to provide flexible allocation of cache space and hardware resource in large-scale cloud environment. In addition to SSD cache, other shared resources such as CPU, memory, disk, network will be investigated for efficient resource management in the VM environment.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012R1A1A2040257 and 2013R1A1A2060398), the second Brain Korea 21 PLUS project, ICT R&D program of MSIP/IITP (1391105003), and Samsung Electronics (S-2014-0700-000). Corresponding author: Hee Yong Youn.

References

1. M. Canim, G. A. Mihaila, B. Bhattacharjee, K. A. Ross and C. A. Lang, SSD Bufferpool Extensions for Database Systems, *Proceedings of the VLDB Endowment*, vol. 3, no. 2, pp. 1435–1446, September 2010.
2. T. Luo, R. Lee, M. P. Mesnier, F. Chen, and X. Zhang, hStorage-DB: Heterogeneity-aware Data Management to Exploit the Full Capability of Hybrid Storage Systems, *Proceedings of the VLDB Endowment*, vol. 5, no. 10, pp. 1076–1087, June 2012.
3. A. J. Smith, Disk Cache-Miss Ratio Analysis and Design Considerations, *ACM Transactions on Computer System (TOCS)*, vol. 3, no. 3, pp. 161–203, August 1985.
4. Kgil, Taeho, David Roberts, and Trevor Mudge, Improving NAND flash based disk caches, *Computer Architecture ISCA 2008 35th International Symposium*, pp. 327-338, June 2008.
5. Pritchett, Timothy and Mithuna Thottethodi, SieveStore: a highly-selective, ensemble-level disk cache for cost-performance, *ACM SIGARCH Computer Architecture News*, Vol. 38. No. 3. ACM, pp. 163-174, June 2010.
6. Stewart, Christopher, Aniket Chakrabarti and Rean Griffith, Zoolander: Efficiently Meeting Very Strict, Low-Latency SLOs, *International Conference on Autonomic Computing*, pp. 256-277, June 2013.
7. Zhu, Timothy, et al., Saving cash by using less cache, *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, June 2012.
8. Jinho Hwang, Wei Zhang, et al., UniCache: Hypervisor Managed Data Storage in RAM and Flash, *Cloud Computing (CLOUD) 2014 IEEE 7th International Conference*, pp. 216-223, July 2014.
9. Luo, Tian, et al., S-CAVE: effective SSD caching to improve virtual machine storage performance, *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*, pp. 103-112, October 2013.
10. Krishnaveni, N., G. Sivakumar, Survey on Dynamic Resource Allocation Strategy in Cloud Computing Environment, *International Journal of Computer Applications Technology and Research (IJCATR)*, Volume 2, Issue 6, pp. 731-737, November 2013.
11. Ahn, Jeongseob, et al., Dynamic virtual machine scheduling in clouds for architectural shared resources, *Proceedings of 4th USENIX Workshop on Hot Topics in Cloud Computing*, June 2012.
12. Janapsatya, Andhi, et al., Dueling clock: adaptive cache replacement policy based on the clock algorithm, *Design, Automation & Test in Europe Conference & Exhibition (DATE) on IEEE*, pp. 920-925, March 2010.
13. Qureshi, Moinuddin K., and Yale N. Patt., Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches, *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society*, pp. 423-432, December 2006.

14. Narayanan, Dushyanth, et al., Migrating server storage to SSDs: analysis of tradeoffs, *Proceedings of the 4th ACM European conference on Computer systems*, pp. 145-158, April 2009.
15. Anchev, Nenad, et al., Optimal cache replacement policy for matrix multiplication, *ICT Innovations 2012 Springer Berlin Heidelberg*, pp. 71-80, September 2012.
16. Jiang, Song, Feng Chen, and Xiaodong Zhang, CLOCK-Pro: An Effective Improvement of the CLOCK Replacement, *USENIX Annual Technical Conference, General Track*, pp. 323-336, February 2005.
17. Liu, Jinjiang, et al., An Efficient Schema for Cloud Systems Based on SSD Cache Technology, *Mathematical Problems in Engineering 2013*, Volume 2013, Article ID 109781, September 2013.
18. Jennings, Brendan, and Rolf Stadler, Resource management in clouds: Survey and research challenges, *Journal of Network and Systems Management*, pp. 1-53, March 2014.
19. Gulati, Ajay, et al., Demand Based Hierarchical QoS Using Storage Resource Pools, *USENIX Annual Technical Conference*, pp. 1-13, June 2012.
20. Liao, Xiaofei, et al., A Performance Optimization Mechanism for SSD in Virtualized Environment, *The Computer Journal*, April 2013.
21. Liu, Jinjiang, et al., An Efficient Schema for Cloud Systems Based on SSD Cache Technology, *Mathematical Problems in Engineering 2013*, Article ID 109781, 2013.
22. S.-Y. Park, D. Jung, J.-U. Kang, J.-S. Kim, and J. Lee, CFLRU: a replacement algorithm for flash memory, *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems 2006*, pp. 234–241, ACM Press, October 2006.
23. H. Shim, B. K. Seo, J. S. Kim, and S. Maeng, An adaptive partitioning scheme for DRAM-based cache in solid state drives, *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies 2010*, May 2010.
24. "SNIA IOTTA Repository," http://iotta.snia.org/
25. "UMass Trace Repository," http://traces.cs.umass.edu/index.php/
26. Jiang Dejun, Guillaume Pierre, Chi-Hung Chi, EC2 performance analysis for resource provisioning of service-oriented applications, *Proceedings of the 2009 International Conference on Service-Oriented, Computing (ICSOC/ServiceWave'09)*, Volume 6275, pp. 197-207, 2009.
27. J. Schad, J. Dittrich, J.A.Q. Ruiz, Runtime measurements in the cloud: observing, analyzing, and reducing variance, *Proceedings of the VLDB Endowment*, vol. 3(1–2), pp. 460–471, 2010.
28. Donald Kossmann, Tim Kraska, Simon Loesing, An evaluation of alternative architectures for transaction processing in the cloud, *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, pp. 579–590, 2010.
29. Carsten Binnig, Donald Kossmann, Tim Kraska, Simon Loesing, How is the weather tomorrow? Towards a benchmark for the cloud, *Proceedings of the Second International Workshop on Testing Database Systems. ACM*, 2009.
30. TPC Benchmark. <http://www.tpc.org/tpch/>
31. Ahmed Abujoda, Network measurements in virtualized networks and its challenges, *6th GI/ITG KuVS Workshop on Future Internet*, November 2010.
32. I.M. Rafika, N. Sadeque, JA Andersson, A. Johnsson, Time-stamping accuracy in virtualized environments, *Advanced Communication Technology (ICACT), 2011 13th International Conference on*. IEEE, pp. 475–480, 2011.
33. J. Whiteaker, F. Schneider, R. Teixeira, Explaining packet delays under virtualization, *ACM SIGCOMM Computer Communication Review*, Volume 41, Issue 1, pp. 38-44, January 2011.
34. L. Youseff, K. Seymour, H. You, J. Dongarra, R. Wolski, The impact of paravirtualized memory hierarchy on linear algebra computational kernels and software, *Proceedings of the 17th international symposium on High performance distributed computing. ACM*, pp. 141–152, 2008.
35. Aceto G, Botta A, de DonatoW, PescapèA. Cloud monitoring: a survey, *Computer Networks*, Volume 57, Issue 9, pp. 2093–2115, June 2013.