The Page Cache Deduplication Mechanism in Virtualized Systems

Seho Lee, Inhyeok Kim, Dongwoo Lee and Young Ik Eom

College of Information and Communication Engineering, Sungkyunkwan University
Gyeonggi-do 440-746, Korea
{loadic, kkojiband, lightof, yieom}@skku.edu

Abstract

The memory deduplication mechanism has been used in virtualized environment. The Linux kernel, however, was implemented as anonymous pages with the same content across inter-virtual machines. To resolve this problem, previous work was considered in the page cache inside the guest. So, we propose a KVM-based deduplication mechanism which is used on the host page cache. It does not require a modification of the guest kernel. Our implementation detects more sharing opportunities than the KSM (by factor of 2x), and its CPU overhead is less than 4%. Also, it reduces the page swap in/out by 40% compared to default VMs in the worst case. Thus, the page cache deduplication is required in virtual environments.

Keywords: *Cloud computing, Virtualization, Page cache, Deduplication*

1. Introduction

The virtualization technique is used for cloud computing, and many IT techniques. It shares one physical computing resource for multiple logical machines. Especially, physical memory has become a scarce resource. Its management is a significant issue in virtual environments. Previous work has tried to resolve this problem using the memory deduplication mechanism. It is researched as the memory sharing [3, 6, 13] and the page deduplication mechanism [1, 12, 14, 16]. The memory sharing in para-virtualization [2, 17] required a modified guest OS. It is possible that the host kernel know semantic information [5, 6, 12] between the hypervisor and virtual machine. The host kernel is able to detect short-lived sharing opportunities [13] more efficiently using semantic information. Our approach, on the other hand, does not modify guest OS on the KVM (Kernel-based Virtual Machine) [10]. However, it is not possible that the host kernel knows the semantic gap, and lead to the double-caching problem [14].

In this paper, we propose an efficient page cache deduplication mechanism of the host page cache-based. The pages that contain the same contents across VMs are mostly in their Virtual Disk Image (VDI) [11]. Our approach detects sharing opportunities in the host page cache from the VDI used by virtual machines. It is able to reduce the double-caching that waste physical memory. Our sharing process has a similar concept with Kernel Shared Memory (KSM) included since the Linux kernel 2.8.32. Our goals are to detect the more sharing opportunities and show that the page cache is potentially a shareable page.

The remainder of this paper is structured as follows: We provide the background of the page cache life cycle and KSM in Section 2. We describe an overview of architecture and the page sharing process in Section 3. In Section 4, we evaluate and

analyze our mechanism. Finally, in Section 5 we summarize our work and depict the future research directions.

2. Backgrounds and Related Work

The candidates of shareable page are selected in the page list that is managed by *address_space* of *inode* from the VDI. In this section, we describe the cycle of page cache that is used in our approach, and the process of the KSM that motivated our content-based page sharing mechanism.

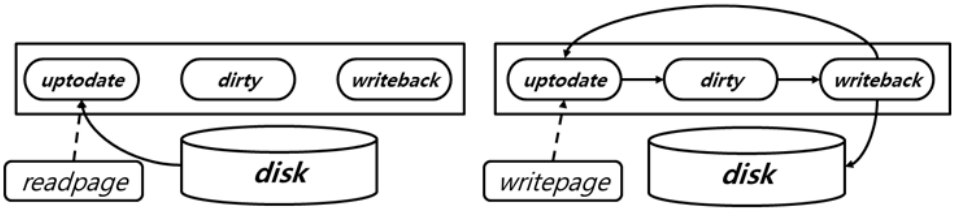


Figure 1. read / write Page Cache Life Cycle

2. 1. Page Cache Life Cycle

The Linux page cache [15] stores frequently accessed disk blocks from the disk to memory. Caching to reduce the disk I/O is a good way to improve system performance. As shown in Figure 1, the page cache has three states of *uptodate*, dirty and *writeback* through the read/write operations. The page cache has the *uptodate* state when the kernel calls *readpage()*. The page cache goes from the dirty to the *writeback* state when the kernel calls *writepage*(). After that *writeback* function is occurred, the page cache writes back into the physical disk block and is set as the uptodate state. We check three states of *uptodate*, *dirty* and *writeback* in order to select the shareable pages, and process to share pages in accordance with states of the page cache.

2.2. Kernel Shared Memory (KSM)

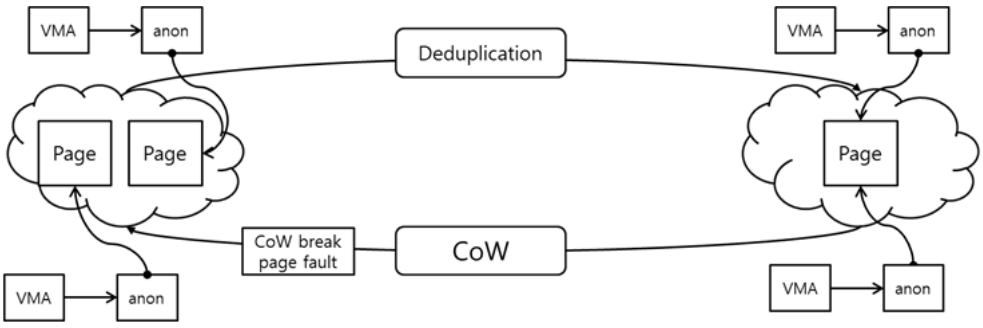


Figure 2. The Pages Merge Process on the KSM

The KSM [1] exists as a daemon in the Linux kernel since version 2.6.32 that periodically performs the page scan to identify anonymous pages with equal content across different virtual machines. As shown in Figure 2, the duplicated page is marked read-only state, but if the page is changed for any reason, the kernel will generate Copy-On-Write (CoW) [7, 16]. Pages in the KSM are managed by two of Red-Black

binary-search trees (the KSM call the *unstable tree* and *stable tree*). The *unstable tree* is used to store the page of sharing candidate that is not yet merged. The *stable tree* stores those pages that are stabilized and shared by the KSM.

The KSM is a scanning-based sharing mechanism because the KVM is a full-virtualization that has the semantic gap between the host and guest. To scan anonymous memory doesn't detect more sharing opportunities without raising the scan rate, so prior studies have mentioned that mergeable pages are the page cache [12, 14]. Also, the current KSM are not well studied to find shot-lived sharing opportunities [6].

Our approach is based on the host-page-cache from the VDI used by virtual machines. So, we propose the sharing mechanism and show the result of sharing opportunities more than the KSM.

3. The Host Page Cache based Memory Deduplication

In this section, we describe a difference between our approach and related work: to select source pages. It has a different memory area for targeting source pages. Also, we introduce implementation methods: the page cache merging and scanning process. The page cache requires a sophisticated management.

3. 1. Kernel Architecture

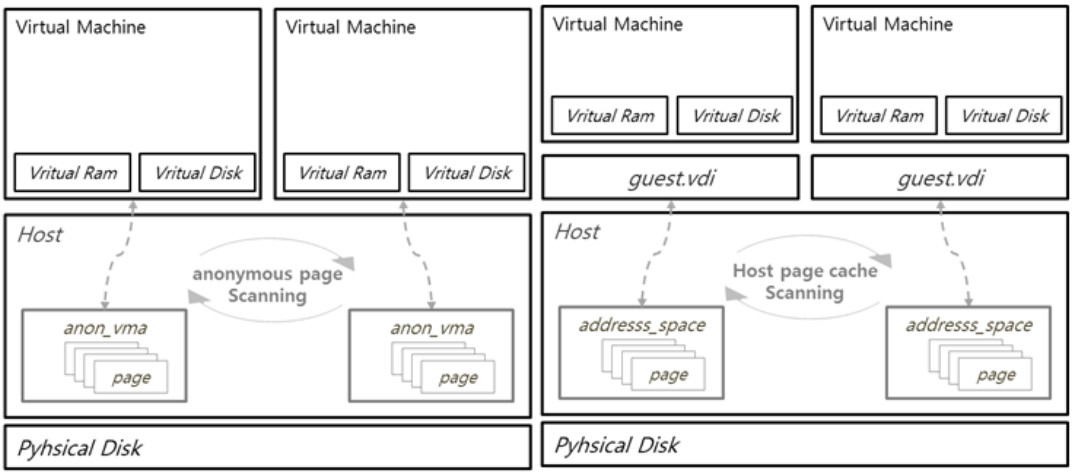


Figure 3. The KSM and the Page Cache Deduplication Mechanism Architecture

As shown in Figure 3, the KSM and host page cache deduplication mechanism is processed through scanning thread in the host kernel. Our architecture looks different from the KSM in the source pages. The target page of the KSM is the host anonymous page for virtual machines. On the other hand, our approach uses the host page cache that is managed by the *address_spcae* from the *inode* of each of the VDI for virtual machines. The pages are managed in the list of *address_space*. We implement the memory deduplication method that is based on the host-page-cache.

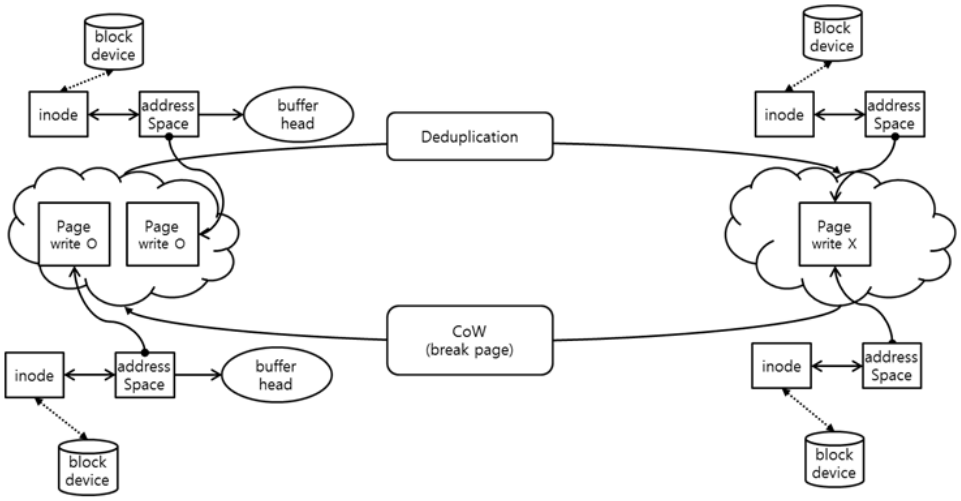


Figure 4. The Page Cache Merging Process

3. 3. The Page Cache Merging Process

In this section, we present the page merging and breaking process that differentiate our approach from prior work on merged page because our approach targets the page cache inside the host. As shown in Figure 4, our approach has the page cache merging and breaking process. The host page cache that has backing device information for a write-back is generated from the VDI. The block device information is saved in the *buffer_head* structure. It is able to quickly locate the disk address of each individual block in the page cache. But, it has problems that are management for the *buffer_head* when the page cache is merged and broken: The merged page cache has two of the *buffer_head* for write-back. It is need to very complex mechanism that manages the page cache when a CoW-break and deduplication is occurred.

We remove the *buffer_head* by clearing a PG_mappedtodisk flag and set a non-writeable page in the page table entry when the page cache is merged. This approach becomes possible because the page cache state of *uptodate* that is called in the *readpage()* hardly ever is changed. The new page cache is allocated when the merged page is broken. If the merged page cache is being written, then *do_wp_page()* is called as the page cache is a CoW page. It is allocated as the new page cache that has the *buffer_head* for the write-back.

3. 3. The Page Cache Scanning Process

We can merge the duplicated pages that have the same content of the page cache. We have two kinds of data structures: red-black tree and hash table. Shared pages are managed in a red-black tree that includes flags, kpfn(kernel page frame number), address_sapce, and pgoff_t(page offset). In this case, both average and worst-case insert, delete, and search operation time is O(log n). The page is registered in a hash table when the page that has the same content does not exist in the red-black tree and hash table. In this case, hash table that guarantee worst case running time of O(n) for search and delete.

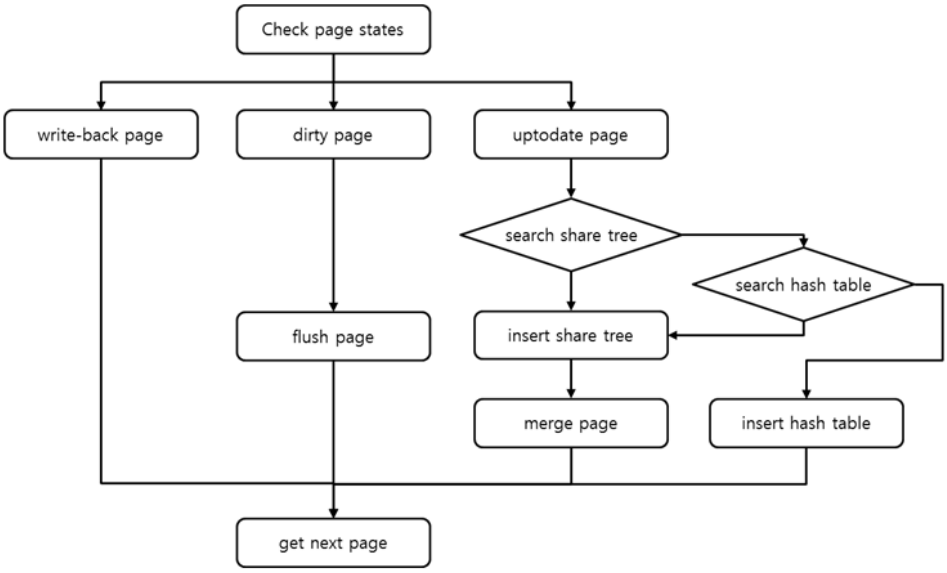


Figure 5. The Scan Process Flow

Figure 5 shows the page cache scanning process. We check the flag when an unshared page is selected. The writeback status does not touch the page because it does try to the write back to the physical disk. Also, the page of the dirty status is not a shareable page candidate, but its page comes back to the uptodate status after the page in the dirty list is written back to the disk. Thus, we try to flush the dirty states of the page cache by force for to get the shareable pages.

Our implementation goes through the scanning process of the memory deduplication by the uptodate page. First of all, the page cache is searched in a red-black tree (which our implementation called a share tree). If the searched page is a shareable page, the page cache is merged as write-protected and inserted into the share tree. Once again the page cache is searched in a hash table for the sharing opportunity when the page does not exist in the share tree. If the hash table does not have the page cache of the same contents, it is inserted into the hash table by not sharing. Otherwise, the page cache of the same content is merged and inserted into the share tree. Finally, we get the next page cache for page sharing. The scan thread runs like the above process during the fixed interval time with the number of pages.

We have the significant difference between the previous work and our implement that is to select the source pages. We can merge the host-page-cache that is cached to the VDI. Thus, our approach is able to share page more than the KSM.

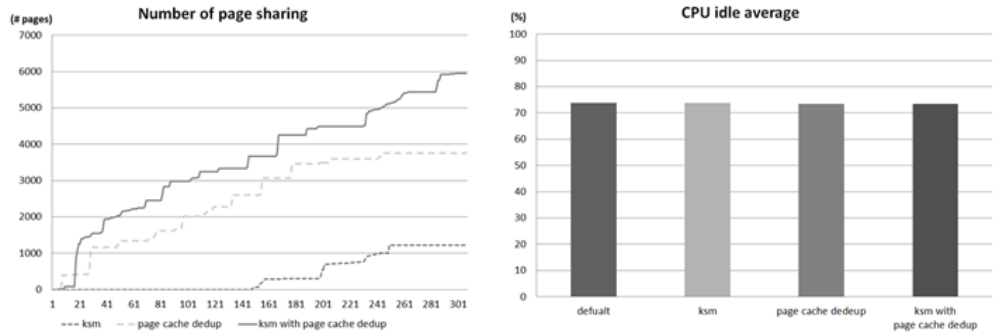
4. Evaluations

In this section, we show why additional page cache management is required. We have conducted the kernel benchmark that was used to prior work [12, 13, 14, 15] and the MD5SUM (eg. access one gigabyte file in ten second) that is used to read a file and calculate the checksum because previous work [1, 14] has shown the I/O performance in the worst-case scenario. Through this, we can identify merged performance, overhead and I/O performance.

4.1. Setup

Our benchmarks have been conducted on a PC with an Intel core i7 2.8GHz and 4 GB of RAM. The host runs the Ubuntu Linux 11.10 (kernel 3.2.0) and the guest virtual machines run kvmtools-based [9] on the Ubuntu server 11.10 (kernel 3.2.0). We configured 100 of pages to scan on wake-up and 20 ms of sleep time between spurts for both KSM and our system.

4.2. Page Sharing Opportunities



(a) Detect of page sharing opportunities (b) CPU idle average

Figure 6. Performance Analysis of the Kernel Benchmark on 2 VMs

The major difference between our approach and the KSM is to detect pages in different part of memory. The KSM source is a guest page cache and guest anonymous page into the host anonymous page. Source of our implementation is the host page cache from the VDI for virtual machines. Figure 6(a) and Figure 7(a) show the number of shared pages as time progresses in the kernel benchmark with two and tree virtual machines. In this case, the number of merged pages with our approach is almost 4x - 11x KSM. Our approach detects more mergeable pages when the number of virtual machines is increased. Also, we have short-lived sharing opportunities without aggressive scan rates when it comes to identifying shareable pages because of the kernel benchmark characteristics. Thus, the host page cache is able to become the good sharing candidate without raising the scan rate.

Our approach with two virtual machines has about 1% CPU overhead that is comparable to the default VM in Figure 6(b), because we have the enough host memory space. On the other hands, we have about 10% CPU overhead when the number of virtual machines is increased, as show in Figure 7(b). It is because of the host memory shortage that is occurred by increasing virtual machines. We have trade-off between the scan rates and CPU utilization. It is enough to merge shot-lived and long-lived sharing opportunities with little overhead in case of kernel benchmark.

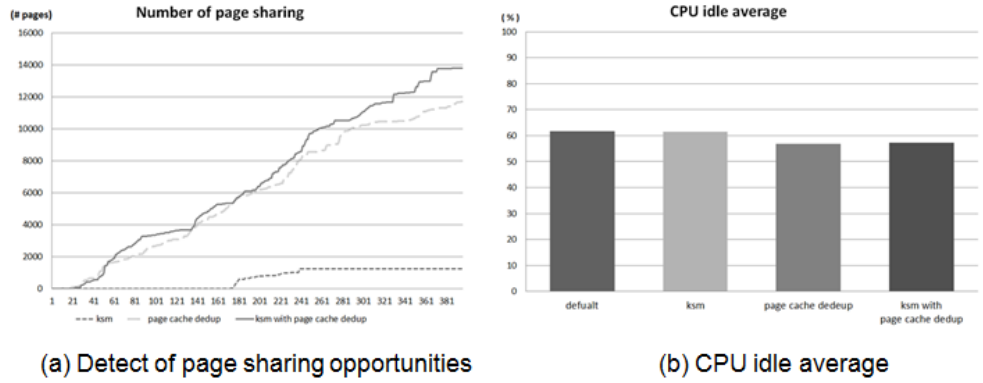


Figure 7. Performance Analysis of the Kernel Benchmark on 3 VMs

4.3. Deduplication Effectiveness

The MD5SUM is a sequential IO workload that is used to read a file and calculate the 128-bit MD5 hashes. Figure 8(a) and (b) shows the swap and block I/O size in MD5SUM with one-gigabyte file on four virtual machines the worst-case scenario: the host physical memory size is 4 GB and the each virtual machine memory size is 1GB. It leads to the host memory shortage. Thus, this occur the swapping and performance loss. In this case, the KSM reduces *swapping* by about 9%. On the other hand, our approach reduces *swapping* by about 40% compared to the default virtual machine. Also, our approach is able to improve the performance by reducing disk IO about 12%, as shown in Figure 6(b).

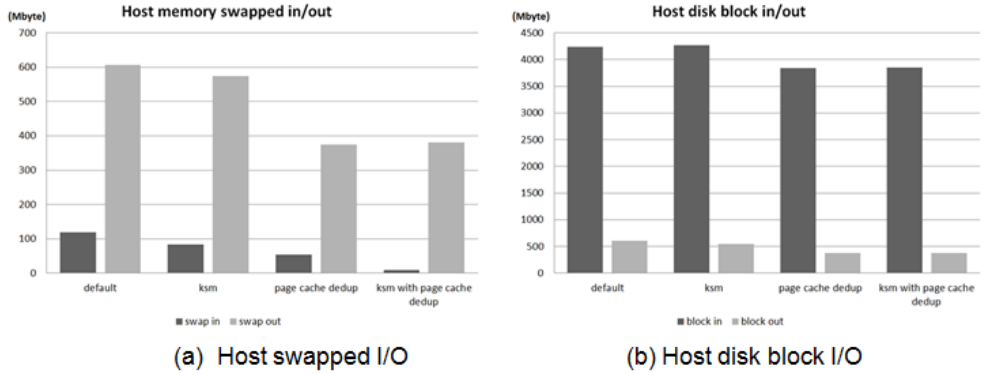


Figure 8. I/O Performance on 4 VMs

5. Conclusions

In this paper, we find out that most of the shareable pages are generated from the VDIs. Thus, we have proposed the memory deduplication mechanism more efficient using the page cache inside the host in virtual environments. Traditional memory deduplication scheme target the anonymous memory area that contain the guest page cache and anonymous page. Our approach is able to share more pages than KSM. Also, it is possible that we detect more sharing opportunities with little CPU overhead, and

reduce the number of calls to the Linux kernel swap daemon (kswapd) in the worst case. Thus, the page cache deduplication is additionally required in virtual environments.

In the future, we will consider a larger variety of workloads to research the performance specific of our mechanism in the different scenarios. Also, our approach is able to apply to the virtual machines for guest physical memory. In addition, we will study efficient solution that is the problem of double-caching in full-virtualization. This is occurred by the lack of semantic information between the host and guest. We use the consistent workload feature in virtual systems. This will make it possible to the efficient memory management in the host physical memory.

Acknowledgements

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the of Education, Science and Technology (2012-0006423)

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(2010-0022570)

References

[1] A. Arcangeli, I. Eidus and C. Wright, "Increasing memory density by using KSM", In Proceedings of the Linux Symposium, **(2009)** July 13-17; Montreal, Canada.
[2] P. Barham, B. Dragovic, K. Fraster, S. Hand, T. Harris, A. Ho, R. Neugebauser, I. Pratt and A. Warfield, "Xen and the art of virtualization", In Proceedings of the nineteenth ACM Symposium on Operating Systems Principles, SOSP '03, **(2003)** October 19-22; The Sagamore, Bolton Landing (Lake George), New York, United States.
[3] E. Bugnion, S. Devine, K. Govil and M. Rosenblum, "Disco: Running Commodity Operating Systems on Scalable Multiprocessors", ACM Transactions on Computer Systems, vol. 15, **(1997)**.
[4] C. R. Chang, J. J. Wu and P. Liu, "An Empirical Study on Memory Sharing of Virtual Machines for Server Consolidation", In proceedings of the 2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications, ISPA '11, **(2011)** May 26-28; Busan, Korea.
[5] P. M. Chen and B. D. Noble, "When Virtual is better than real. In Proceedings of the Eighth Workshop on Hot Topics in Operating Systems", **(2001)** May 20-22; Washington, DC, United States.
[6] D. Gupta, S. Lee, M. Vrable, S. Savage, A. Snoeren, G. Varghese, G. M. Voelker and A. Vahdat, "Difference Engine: Harnessing Memory Redundancy in Virtual Machines", In Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation, **(2008)** December 8-10; San Diego, CA, United States.
[7] M. T. Jones, "Anatomy of Linux Kernel Shared Memory (Memory de-duplication in the Linux kernel)", *http://www.ibm.com/developerworks/linux/library/l-kernel-shared-memory/index.html.* IBM developerWorks **(2010)**.
[8] S. T. Jones, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau, "Geiger: Monitoring the Buffer Cache in a Virtual Machine Environment", In Proceedings of the 12th international conference on Architectural Support for Programming Languages and Operating Systems, **(2006)** October 21-25; Jan Jose, CA, United States.
[9] kvmtools, *http://www.linux-kvm.org/page/kvmtools*.
[10] A Kivity, Y. Kamay, D. Laor, U. Lublin and A. Liguori, "KVM: The Linux Virtual Machine Monitor", In Proceedings of the Linux Symposium, **(2007)** July 17-30; Ottawa, Canada.
[11] F. Kloster, J. Kristensen and A. Mejlholm, "Determining the Use of Interdomain Shareable Pages using Kernel Introspection", Technical report, Aalborg University, **(2007)**.
[12] K. Miller, F. Franz, T. Groeninger, M. Rittinghaus, M. Hillenbrand and F. Bellosa, "KSM++: Using I/O-based Hints to Make Memory-Deduplication Scanners more Efficient", Proceedings of the ASPLOS Workshop on Runtime Environments, Systems, Layering and Virtualized Environments, **(2012)** March 3; London, United Kingdom.
[13] G. Miłos, D. G. Murray, S. Hand and M. A. Fetterman, "Satori: Enlightened Page Sharing", Proceedings of the 2009 USENIX Annual Technical conference, **(2009)** June 14-19; San Diego, United States.

[14] P. Sharma and P. Kulkarni, "Singleton: System-wide Page Deduplication in Virtual Environments", Proceeding High-Performance Parallel and Distributed Computing, **(2012)** June 18-22; Delft, Netherlands.
[15] B. Singh, "Page/Slab Cache Control in a Virtualized Environment", In Proceedings of the Linux Symposium, **(2009)** July 13-16; Ottawa, Canada.
[16] C. A. Waldspurger, "Memory Resource Management in VMware ESX Server", In Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation, **(2002)** December 9-11; New York, United States.
[17] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker and S. Savage, "Scalability, Fidelity, and Containment in the Potemkin Vitual Honeyfarm", In Proceedings of the twentieth ACM symposium on Operation systems Prinsiples, **(2005)** October 23-26; Brighton, United Kingdom.

Authors

Seho Lee received the B.S. degree in Department of Electronics and Communications Engineering from Kwangwoon University, S. Korea, in 2012. He is now a member of Distributed Computing Lab at Sungkyunkwan University as a M.S. student. His research interests include embedded software, Linux kernel, and virtualization.

Inhyeok Kim received the B.S. degree and M.S. degree at the College of Information and Communication Engineering in Sungkyunkwan University, S. Korea in 2006 and 2010 respectively. He is now a member of Distributed Computing Lab at Sungkyunkwan University as a Ph.D. candidate. His research interests include Operating System, Linux kernel, and virtualization.

Dongwoo Lee received the B.S. degree and M.S. degree at the College of Information and Communication Engineering in Sungkyunkwan University, S. Korea in 2010 and 2012 respectively. He is now a member of Distributed Computing Lab at Sungkyunkwan University as a Ph.D. candidate. His research interests include mobile virtualization and cloud computing.

Young Ik Eom received a B.S., M.S. and Ph.D. degrees from the Department of Computer Science and Statistics of Seoul National University, Korea, in 1983, 1985 and 1991, respectively. He was 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 is a professor at Sungkyunkwan University in Korea. His research interests include distributed computing, system software and virtualization.