SSD Bandwidth Distributing I/O Scheduler Considering Garbage Collection

Jung Kyu Park[1], Member and **Jaeho Kim**[2]

ABSTRACT

There were scheduler studies for QoS(Quality of Service) or SLA(Service Level Agreement) of hard disks. The use of SSDs as storage has been increasing dramatically in recent systems due to their fast performance and low power usage. However, the studies to guarantee the SLA are based on the hard disk and do not consider SSD which is a flash storage device. In the SSD, GC(Garbae Collection) process copies data to an empty block and the corresponding block is removed by the GC. This causes SSD performance to degrade in a virtualized environment with many I/Os. We considered the Linux scheduler to take SSD characteristics into consideration and to improve I/O performance. In this paper, we propose a MTS-CFQ I/O scheduler that is implemented by modifying the existing Linux CFQ I/O scheduler. Our proposed method controls the time slice based on the I/O bandwidth for the current storage device. Real workload-driven simulation based experimental results have shown that MTS-CFQ can improve performance by up to 20% with an average of 5%, compared with the traditional CFQ I/O for the four workload considered.

Keywords: I/O scheduler, CQF, SSD, SLA

1. INTRODUCTION

Solid State Drive(SSD), which is a storage device based on flash memory, has been rapidly increasing in usage as a storage device of computer system due to its fast performance and low power consumption. As the degree of integration of SSD increases and the price drops, the use of SSD is also emerging in virtualization systems that are widely used in servers. However, most recent operating system and system software have been developed assuming that the hard disk is used as a storage device for a long period of time [1] [2].

SSDs are used extensively in general systems, but they can perform better in server environments. If you plug in the slowest hardware on the server computer system can pick the hard disk storage device. Therefore, if you replace the hard disk with the SSD, you can expect to improve the performance of the server. Since the price of SSD is higher than that of hard disk, it is possible to improve the server performance by adding a little extra cost if SSD is used as a cache in the upper layer of the hard disk instead of replacing all the hard disks.

Among the most recent server systems, cloud-based server systems and virtual environment server systems are the most important issues. Since multiple virtual machines can be operated on a single physical system, system resources can be efficiently used by minimizing the amount of resources remaining while sharing hardware resources. If SSD is used in the virtualization server environment part, it is necessary to share SSDs of various virtual machines because each virtual machine must efficiently use resources.

In a server system in a virtualized environment, it is easy to allocate CPU and memory resources compared to other resources. However, resource allocation of storage devices is one of the relatively difficult research areas [3]. This is particularly evident in terms of service level agreements (SLAs). Unlike a hard disk that reads and writes data on the basis of the physical characteristics of the hard disk (rotation delay time, seek time, and data transfer time), the SSD can read and write data electronically without any physical work. The biggest difference between a hard disk and an SSD is the garbage collection resulting from the physical characteristics of the SSD [4] [5] [6].

The hard disk can physically erase data because existing data can be overwritten with new data at the location. However, in the case of SSD, the semiconductor chip is composed of blocks, blocks are composed of pages, data can be read and written in units of pages, and block units can be deleted. Therefore, when deleting some data, it is often indicated that the data is not valid through the Flash Translation Layer (FTL), and the data is not actually deleted. As the writing and erasing are repeated, an invalid page is generated in the flash. If there is a lot of invalid pages and the free space in the flash of the SSD becomes insufficient, delete the invalid pages in the block and copy the valid pages to the other blocks and delete the existing blocks to secure the pages to be written. This process is called garbage collection.

Manuscript received on August 14, 2017 ; revised on December 15, 2017.
Final manuscript received on January 29, 2018.

[1] The author is with Department of Computer Software Engineering, Changshin University, Korea., E-mail: smartjkpark@gmail.com
[2] The author is with Department of Electrical and Computer Engineering at Virginia Polytechnic Institute and State University, USA., E-mail: kjhnet@gmail.com

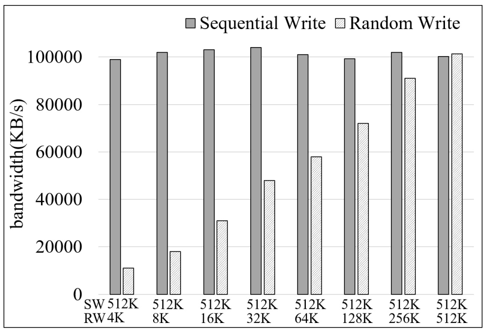


Fig.1: *Write performance test without garbage collection*

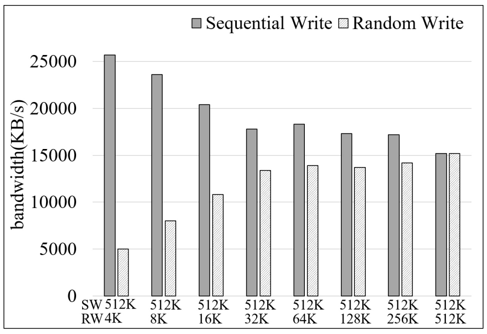


Fig.2: *Write performance test with garbage collection*

If there is garbage collection in a single SSD by multiple processes due to garbage collection, the write bandwidth of each process may differ. Fig.1 and Fig.2 show the experimental results of how two virtual machines run in a KVM hypervisor virtualization environment and how the write performance of each virtual machine changes due to garbage collection. In the first virtual machine, the block size is fixed at 512KB and 1GB of data is sequentially written. In the second virtual machine, the block size is gradually increased from 4KB to 512KB, and the bandwidth is investigated while performing 1GB random write. Fig.1 shows an SSD with no garbage collection, and Fig.2 shows an SSD with garbage collection.

In Fig. 1, we can see that 512 KB of sequential writes in the write bandwidth are not affected as the block size of the random write increases from 4 KB to 512 KB. That is, when garbage collection does not occur, it does not affect the writing of each virtual machine.

However, in the case of garbage collection in Fig. 2, the bandwidth of sequential writes decreases as the block size of random write increases. In other words, it can be seen that when two virtual machines write at the same time when garbage collection occurs, the bandwidth of the sequential write can be reduced due to the garbage collection caused by the random write.

This garbage collection can adversely affect the virtual environment server using SSD. When garbage collection occurs when multiple virtual servers access a single SSD at the same time, the ratio of write bandwidth when they do not occur varies so that proper SLA based on the ratio can not be guaranteed. For this reason, considering the characteristics of garbage collection of SSD, there is a need for a solution that guarantees SLA when garbage collection occurs and when garbage collection does not happen.

In this paper, we utilize Manager of Time Slice-CFQ (MTS-CFQ), which is an I/O scheduler which is a modification of CFQ scheduler which is a Linux basic scheduler considering the characteristics of SSD [7]. The MTS-CFQ determines whether the bandwidth is distributed as intended by the current manager referring to the I/O bandwidth, and then the bandwidth is distributed by the ratio set by the time slice adjustment.

2. STATE OF THE ART

There have been studies on QoS(Quality of Service) or SLA(Service Level Agreement) for hard disks or SSDs and related schedulers. However, studies to guarantee SLA were based on hard disk without considering flash-based SSD. The Studies considering the characteristics of SSDs that perform I/O operations without mechanical characteristics have studied scheduling techniques to guarantee fairness of I/O, but have not considered SLA aspects. Also, existing studies have considered QoS and SLA, but often do not consider garbage collection, which is an important characteristic of SSD.

PARDA algorithm and VM-PSQ algorithm have been proposed a new scheduling scheme for distributing I/O resources in virtualized environments or distributed systems [6] [8]. The PARDA improves I/O bandwidth distribution performance and fairness by measuring I/O latency in a distributed storage environment and adjusting the length of the I/O queue based on this information. VM-PSQ improves I/O bandwidth allocation performance over existing schedulers by considering I/O time slices and scheduling tokens simultaneously in a virtualized environment. However, since the storage medium is limited to the existing hard disk, the characteristics of the SSD are not properly reflected.

FIOS algorithm and FlashFQ algorithm have been proposed a new scheduler for fair I/O resource allocation considering the characteristics of flash-based SSD storage [10] [11]. FIOS improves efficiency and I/O performance for fair I/O distribution by implementing a scheduler that understands the nature of SSD's read faster than write and reflects latency for reads and writes. FlashFQ improves the I/O time slice distribution method when several threads perform I / O considering the characteristics of SSD. This technique reduces I/O response time and improves I/O fairness

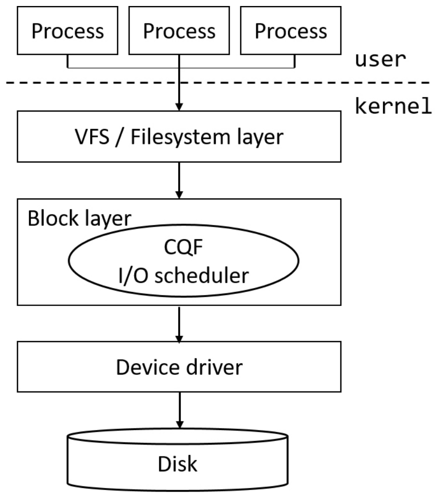


Fig.3: *CFQ I/O scheduler in Linux*

compared to existing schedulers. However, the algorithms did not consider the SLAs and the garbage collection characteristics of SSDs.

The CFQ (Completely Fair Queuing) I/O scheduler has been included in the Linux kernel since 2003. CFQ is basically based on time slice rather than based on I/O requests. Fig. 3 shows the Linux CFQ I/O scheduler. CFQ is an I/O scheduler that is useful when processes have many I/O requests because they have I/O wait queues per process. CFQ is designed to maintain fair I/O fairness by fairly adjusting the time slice of each I/O request when multiple I/O requests occur simultaneously on a disk device by multiple processes. It is also possible for CFQ to assign fair I/O distribution and priority for I/O to processes. And I/O bandwidth ratio can be adjusted by using time slice. Using the KVM method, each virtual machine is treated as a separate process and is recognized by the I/O scheduler, so allocating a weight for I/O to this process is equivalent to distributing I/O bandwidth per virtual machine.

3. MTS-CFQ ALGORITHM

The MTS-CFQ proposed in this paper is implemented by adding MoTS (Manger of Time Slice) to manage the time slice part in the CFQ I/O scheduler of existing Linux. Fig. 4 shows the MoTS operating inside the CQF.

When a user weights a virtual machine for bandwidth allocation, CFQ creates and activates an MoTS thread. MoTS operates in a device-independent manner. MoTS is device independent. If you have one hard disk and one SSD shared by multiple virtual machines, CFQ creates two threads, one for the MoDT for the hard disk and one for the SSD. That is, it is designed to control the time slice by creating a thread for each storage device. Therefore, even if the virtual machine assigns different weights to each storage device, it can control the time slice independently of the

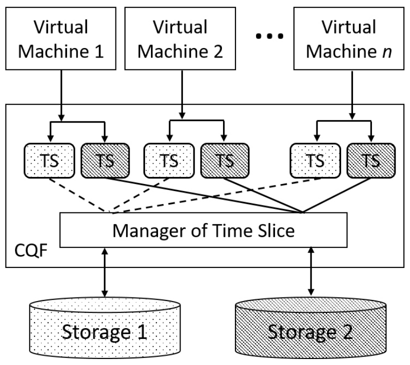


Fig.4: *Operation method of Manager of Time Slice*

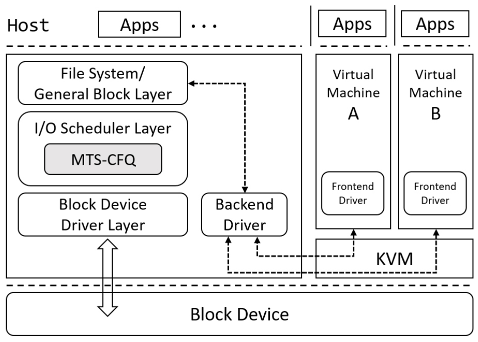


Fig.5: *Operating Environment of MTS-CFQ*

device.

MTS-CFQ adjusts the time slice based on the I/O bandwidth for the current device. So when multiple virtual machines on a disk are performing I/O at the same time, they must be able to collect bandwidth for the respective storage device of the virtual machine.

In this paper, we utilize the iostat which is linux command for system monitoring to measure the detailed I/O bandwidth information of storage devices so that MoTS can collect bandwidth [12]. Fig. 5 shows the MTS-CFQ operating environment.

3.1 Implementation

Fig. 6 shows the algorithm behavior of MTS-CQF. When the weight of I/O bandwidth is given to the CFQ by the virtual machine in the storage device of the virtual machine, the MoTS thread is operated. The MoTS thread collects bandwidth every 1 second for each virtual machine through iostat to check that the bandwidth is distributed as much as the user-specified weight. Based on this, the remaining virtual machines except *TVM* (Tiny VM) Remaining VM I/O time slice. When distributing the bandwidth to two or more virtual machines, the time slice of *RVM* is adjusted based on *TVM* without adjusting the I/O time slice of *TVM* having the least weight.

MTS-CQF refers to the amount of time that the MoTS thread has been running or the bandwidth window (BW) seconds, and adjusts the I/O time slice

```
TVM : Tiny VM that has least weight of VM
RVM : Remaining VMs
BW : bandwidth windows
BWT : Mean bandwidth of TVM
BWR : Mean bandwidth of RVM
TSOVM : Time Slice of OVM
$s = 0$
while(1){
    $IP =$ weight of $RVM$ / weight of $TVM$
    $s++$
    if ($time > bw$)
        $BWR = (\sum_{s-BW-1}^{s} VMI\_bandwitdth)/BW$
        $BWT = (\sum_{s-BW-1}^{s} VMS\_bandwitdth)/BW$
        $F = \frac{IP}{BWR/BWT}$
        $F_s = \left\{(\sum_{s-BW-1}^{s-1} F_{s-1})/W\right\} * F$
        $TSOVM* = \left(\sum_{time-BW-1}^{time} F_s\right)/W$
    else
        $BWR = (\sum_{1}^{s} VMI\_bandwitdth)/s$
        $BWT = (\sum_{1}^{s} VMS\_bandwitdth)/s$
        $F = \frac{IP}{BWR/BWT}$
        $F_s = \left\{(\sum_{s-BW-1}^{s-1} F_{s-1})/W\right\} * F$
        $TSOVM* = (\sum_{1}^{s} F_s)/s$
    endif
    $sleep(1)$
}
```

Fig.6: *Algorithm of MTS-CFQ*

based on the bandwidth value of the most recent BW seconds of the virtual machines after BW seconds. In this paper, the bandwidth allocation performance is experimented by changing the BW value as shown in Table 1. From the experimental results, it can be seen that the bandwidth distribution performance is the best when the BW value is 10. If the time to reference the bandwidth is too small, the deviation of the time slice adjustment of the workload where the value of the bandwidth changes a lot every second becomes large. On the other hand, if the bandwidth reference time is too large, the effect of the recent bandwidth is reflected too little, and the time slice adjustment is not accurate and the bandwidth distribution performance is degraded. For this reason, we used 10 seconds as the BW value in this paper.

Correction factor F (Factor) is calculated by the ratio of IP (Ideal Proportionality) set by the user in advance to *BandWidth of RVM* (BWR) and *Bandwidth of TVM* (BWT). It can be obtained by dividing it into divided values. As a practical example, assume that we have assigned weights of 1 and 10 to both virtual machines VM1 and VM2, respectively. However, if the average value of the bandwidth in the last 10 seconds is 1 to 5, the F value becomes 2. In other words, by doubling the size of the VM2 time slice, it is expected that the bandwidth is distributed by 1

Table 1: *BW value per bandwidth distribution ratio*

	VM1	VM2	VM3	VM4
weight / BW	10	7	4	1
5	8.3	7.3	4.4	1
10	10.3	7.6	4.4	1
20	10.8	8.2	4.7	1

Table 2: *KVM environment*

	W	core	mem	OS	Storage
Host	-	8	8GB	Fedora 14	Dedicated storage
1	10	1	1GB	CentOS 6.4	S840Pro 30GB
2	7	1	1GB	CentOS 6.4	S840Pro 30GB
3	4	1	1GB	CentOS 6.4	S840Pro 30GB
4	1	1	1GB	CentOS 6.4	S840Pro 30GB

to 10 times. However, the MoTS algorithm does not directly apply the F value to the time slice. The average value of the F values of the latest BW seconds is multiplied by$F$ to obtain the F value reflecting the F value of the W seconds and the average value of the F values of the latest BW seconds is multiplied by the time slice size of the corresponding virtual machine. Through this process, it is possible to prevent the time slice from being changed suddenly, and the latest bandwidth can be appropriately reflected in F.

4. EXPERIMENTS AND RESULTS

4.1 Experiments Environment

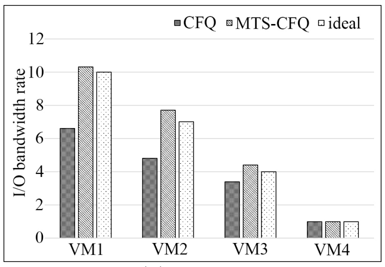
To evaluate the performance of the MTS-CFQ I/O scheduler, two types of benchmark tools were used : FileBench and TraceReplayer [13] [14]. We performed the experiment with a total of five kinds of workloads (Fileserver, Varmail, TraceReplayer, MSN, Exchange, and Financial)with FileBench. In order to measure bandwidth distribution performance, four KVM virtual machines were set up and operated, and various I/O weights were set from 1 to 10 from VM1 to VM4. The SSD used the recently sold Samsung 840 Pro 256GB model, and created four 30GB each in four partitions, allowing four virtual machines to use each partition independently. Table 2 summarizes the experimental environment.

4.2 Results

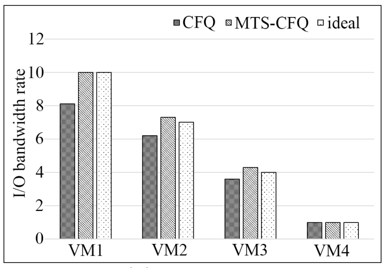
The MTS-CFQ performance evaluation is based on the characteristics of the garbage collection, and the SSD is classified into the aged state and the SSD with no data. Table 3 summarizes the characteristics of the workloads used in the experiment. The existing I/O scheduler, CFQ, differs in bandwidth allocation performance from aged SSDs to clean SSDs in most workloads. Experiments have shown that when writ-

Table 3: *Properties of experiment workload*

workload	write/read ratio	average request size(KB)
Fileserver	2	72
varmail	1	24
MSN	66.7	22.5
Exchange	1.5	12.5
Financial	0.24	2.38

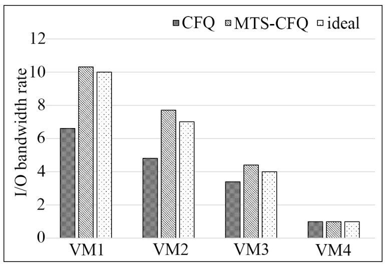


(a)Aged SSD

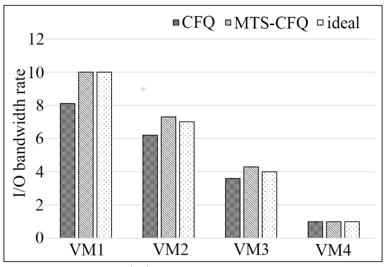


(b)Cleaned SSD

Fig.7: *Results of fileserver workload*

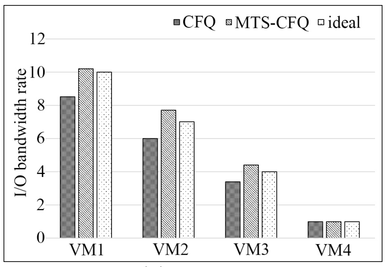


(a)Aged SSD

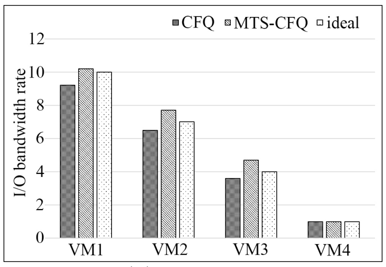


(b)Cleaned SSD

Fig.8: *Results of varmail workload*



(a)Aged SSD



(b)Cleaned SSD

Fig.9: *Results of MSN workload*

ing to an aged SSD by performing a lot of write operations on the SSD, the garbage collection operation occurs as described above, thereby causing a problem in bandwidth distribution performance.

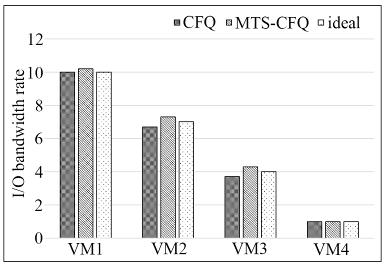
Clean State SSDs exhibited better bandwidth distribution performance than aged SSDs, but did not show ideal bandwidth distribution performance across all workloads. However, since the MTS-CFQ scheduler proposed in this study adjusts the time slice by measuring the bandwidth in real time, the result shows that it is close to the ideal bandwidth distribution in most cases.

Fig. 7 and 8 show that the error rates of fileserver and varmail workloads are about 20% for the aged SSDs and 15% for the clean SSDs. However, MTS-CFQ showed error rates of 8% for the fileserver workload and 2% for the varmail workload. In the case of the MSN workload shown in Fig. 9, the performance of the SSD was improved from 14.2% to 7.6% in the aged state, but the performance was slightly lowered in the clean state. The exchange workload shown in Fig. 10 shows good performance within error rate of 5% for both CFQ and MTS-CFQ.

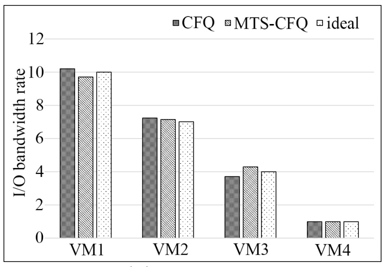
5. CONCLUSION

This study aims to improve bandwidth distribution performance when multiple virtual machines share one storage medium in a virtualized environment. In particular, we focused on SSD, which is the newest storage medium, and observed that the bandwidth allocation performance when the garbage collection occurs in SSD is different from that in the case where the garbage collection does not occur. To improve this, MTS-CFQ I/O scheduler.

The MTS-CFQ I/O scheduler modified the existing Linux I/O scheduler CFQ to improve I/O bandwidth distribution performance. MTS-CFQ first observes whether the bandwidth of the partition of the storage medium shared by the virtual machine is divided according to the user's intention. If the bandwidth allocation is not better than the user-specified



(a)Aged SSD



(b)Cleaned SSD

Fig.10: *Results of Exchange workload*

weight, a dynamic time-slice control method is used that increases the I/O time slice of the virtual machine and reduces the I/O time slice when the bandwidth distribution is good.

As a result of performance evaluation of MTS-CFQ I/O scheduler using benchmark tool, bandwidth distribution performance when SSD garbage collection occurs and when it does not occur improved performance compared to existing CFQ I/O scheduler, And the I/O bandwidth is distributed according to a predetermined weight.

References

[1] X. Song, J. Yang, and H. Chen, *Architecting Flash-based Solid-State Drive for High-performance I/O Virtualization*, Computer Architecture Letters, vol. 13, no. 2, pp. 61-64, July 2013.

[2] T. Luo, S. Ma, R. Lee, X. Zhang, D. Liu, and Li Zhou, *S-CAVE: Effective SSD Caching to Improve Virtual Machine Storage Performance*, In Proc. of International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 103-112, Sep. 2013.

[3] H. Tan, C. Li, Z. He, K. Li and K. Hwang, *VMCD: A Virtual Multi-Channel Disk I/O Scheduling Method for Virtual Machines*, IEEE Transactions on Services Computing, vol. 9, no. 6, pp. 982-995, May 2015.

[4] F. Meng, L. Zhou, X. Ma, S. Uttamchandani, and D. Liu, *vCacheShare: Automated Server Flash Cache Space Management in a Virtualization Environment*, In Proc. of USENIX Conference on USENIX Annual Technical Conference(ATC), pp. 133-144, June 2014.

[5] Y. Yang and J. Zhu, *Analytical modeling of garbage collection algorithms in hotness-aware flash-based solid state drives*, 2014 30th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1-10, June 2014

[6] W. Shin, M. Kim, K. Kim and H. Y. Yeom, *Providing QoS through host controlled flash SSD garbage collection and multiple SSDs*, In Proc. of 2015 International Conference on Big Data and Smart Computing (BigComp), Feb. 2015.

[7] S. Seelam, R. Romero, P. Teller and B. Buros, *Enhancements to Linux I/O Scheduling*, In Proc. of the Linux Symposium, pp. 175-192, Sept. 2014.

[8] Q. Niu, J. Dinan, Q. Lu and P. Sadayappan, *PARDA: A Fast Parallel Reuse Distance Analysis Algorithm*, IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS), pp. 1284-1294, May 2012.

[9] D. Kang, C. Kim ; K. Kim and S. Jung, *Proportional Disk I/O Bandwidth Management for Server Virtualization Environment*, International Conference on Computer Science and Information Technology(ICCSIT '08), pp. 647-652, Sept. 2008.

[10] Q. Deng, Y. Luo, and J. Ge, *Dual threshold based unsupervised face image clustering*, In Proc. of the 2nd International Conference on Industrial Mechatronics and Automation, pp. 436-439, May 2010.

[11] SIMGRID Project, http://simgrid.gforge.inria.fr.

[12] iostat, http://www.freebsd.org/cgi/man.cgi?iostat.

[13] filebench, http://sourceforge.net/projects/filebench.

[14] ioreplay, https://code.google.com/p/ioapps/wiki/ioreplay.

Jung Kyu Park received the M.S. and Ph.D. degrees in computer engineering from Hongik University in 2002 and 2013, respectively. He has been a research professor at the Dankook University since 2014. From 2016 to 2017, he was a visiting professor at Department of Digital Media Design and Applications, Seoul Women's University. In 2018, he joined the assistant professor of Department of Computer Software Engineering, Changshin University. His research interests include operating system, new memory, embedded system and robotics theory and its application.

Jaeho Kim received the BS degree in information and communications engineering from Inje University, Gimhae, Korea, in 2004, and the MS and PhD degrees in computer science from the University of Seoul, Seoul, Korea, in 2009 and 2015, respectively. He is currently a postdoctoral researcher in the Department of Electrical and Computer Engineering at Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg in Virginia, US. His research interests include storage systems, operating systems, and computer architecture.