Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/092L34

I/O Characteristics and Implications of Big Data Processing on Virtualized Environments

*Sewoog Kim, Dongwoo Kang and Jongmoo Choi**

Dankook University, 152, Jukjeon-ro, Suji-gu, Yongin-si, Gyeonggi-do, 448-701, Korea

Received: 18 Jun. 2014, Revised: 20 Aug. 2014, Accepted: 22 Aug. 2014
Published online: 1 Apr. 2015

Abstract: In this paper, I/O characteristics of data-intensive applications running on virtualized environments are explored. It is observed that virtual machines have a tendency to request I/Os in a bursty manner. Also, I/Os are triggered by several virtual machines at the same time. These concurrent and bursty I/Os cause the interference among virtual machines such as frequent context switches and long seek distances, which eventually deteriorates I/O performance significantly. To overcome this problem, a novel burstiness-aware I/O scheduler is proposed, which consists of three components: burstiness detector, coarse-grained dispatcher and starvation handler. The key idea of the proposed scheduler is detecting bursty virtual machines on-line and allowing a detected machine to consume most of disk bandwidth exclusively for a given time quantum to reduce the interference. In addition, it provides long-term fairness while avoiding starvation. Performance evaluation based on implementation shows that the proposed scheduler can improve the execution time and I/O throughput of the three real workloads by decreasing the number of context switches and seek distances.

Keywords: Virtualization, Interference, Big data, Burstiness-aware, I/O Scheduler, Implementation

1 Introduction

This study is inspired by two technologies, big data processing and virtualization. Big data processing drives nearly every aspect of our society, including mobile, financial, manufacturing and Internet services [1]. For efficient big data processing, several frameworks such as MapReduce [2] and Dryad [3] have been developed. MapReduce is a simple, however, it is a powerful programming model for data-intensive applications where big data is distributed into multiple chunks and processed in parallel by a large number of tasks, called map and reduce tasks. It also has a popular open-source implementation, Hadoop [4], which is actively used by Yahoo, Facebook and Amazon [5].

Virtualization is a technology that abstracts physical resources into multiple logical ones so that multiple virtual machines can run at the same time on a single physical machine [6]. It introduces a new software layer called hypervisor ,also known as VMM (Virtual Machine Monitor), that governs the behavior of virtual machines [7,8]. The hypervisor supports various virtualization techniques such as proportional-share algorithm [9] and credit scheduler [10] for CPU virtualization, ballooning [8] and difference engine [11] for memory virtualization, and IDD (Isolated Device Domain) [7] and VMM-bypass [12] for I/O virtualization.

Traditionally, the MapReduce framework works on a large physical cluster that is composed of physical machines such as commodity PCs and servers [2]. Recently, as the virtualization technology becomes prevalent, the framework also makes use of a virtual cluster consisting of virtual machines [5,13,14,15]. For instance, we can rent large numbers of virtual machines from Amazon's EC2 (Elastic Compute Cloud) at low cost (around $0.10 per CPU hour [14]) and use them to execute the map and reduce tasks. Furthermore, we can scale up and down flexibly by considering the workloads.

However, integrating big data processing with virtualization raises a new issue since virtual machines actually share the same physical resources including CPU and disks [16,17]. Shafer showed that, in virtualized environments, storage bandwidth is reduced to 51% and 77% of a non-virtualized disk for reads and writes, respectively [18]. We have also observed the similar phenomena; the I/O performance is degraded approximately 31% as we run data-intensive applications on multiple virtual machines concurrently. This

* Corresponding author e-mail: choijm@dankook.ac.kr

degradation gives a negative impact on big data processing using virtual machines.

To overcome this problem, we propose a new I/O scheduler that is motivated by our observation about the I/O characteristics of data-intensive applications in virtualized environments. We observed that each virtual machine has a tendency to issue I/Os in a bursty manner and several virtual machines trigger the bursty I/Os concurrently. Note that since virtual machines actually share the same physical resources, these concurrent and bursty I/Os incur the performance interference, resulting in considerable amount of context switch overhead among virtual machines and long seek distances in a disk.

The main objective of our proposed scheduler is to alleviate such interference. In other words, it identifies bursty virtual machines on-line and then schedules them in a round-robin fashion with relatively large time quantum. During the time quantum, a scheduled virtual machine can utilize most of disk bandwidth in an isolated manner, resulting in less interference. Moreover, in order to avoid starvation, we reserve the minimum amount of I/O bandwidth for the non-bursty virtual machines.

We have implemented the proposed scheduler in the Xen hypervisor using the control group mechanism [20]. For performance evaluation, we considered three I/O-intensive applications: Terasort [2], Fio [21] and IOzone [22]. Experimental results have shown that the proposed scheduler can improve the overall execution time of the Terasort application from 192 to 165 seconds, while enhancing the throughput of the FIO and IOzone benchmark up to 24% and 22%, respectively. These improvements are obtained by reducing the number of context switches among virtual machines and seek distances in a disk.

The rest of this paper is organized as follows. In Section 2, we describe the motivation of our work. Then, the burstiness-aware I/O scheduler is discussed in Section 3. In Section 4, we present the performance evaluation results. Related work is discussed in Section 5. Finally, conclusion and future work are summarized in Section 6.

2 Motivation

In this section, we describe the motivation of our work. We first explain our experimental setup. Then, we present our observations and discuss the reasons for the I/O performance degradation.

2.1 Experimental Platform

Figure 1 shows our experimental platform, consisting of the MapReduce framework based on virtual machines. Our system is composed of AMD Phenom 8 cores, 8GB DDR3 DRAM, and 500GB SATA disk as shown at the bottom of the figure.

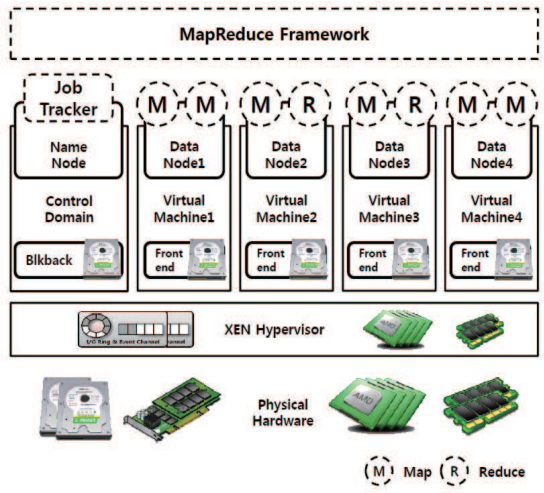


Fig. 1: MapReduce framework based on virtual environments.

On top of the physical hardware, we install the Xen hypervisor version 4.1.4 that abstracts physical resources into multiple virtual resources. On these virtual resources, we create one control machine and four virtual machines. Each virtual machine executes a GuestOS such as Microsoft windows or Linux, acting as an independent computing platform. The control domain takes a special role in Xen, managing virtual machines and I/O devices. It also performs I/O requests on behalf of GuestOSes [7].

On these virtual machines, we install the Hadoop version 1.1 Hadoop consists of a name node and several data nodes. When a MapReduce application is delivered to the name node, it creates map and reduce tasks, and assigns them into the data nodes. Each data node executes its assigned map and reduce tasks. In our experimental system, we use the control domain as the name node while using virtual machines as data nodes.

2.2 Observations

Now let us discuss the I/O behavior of Hadoop in virtualized environments. The input data of a MapReduce application is divided into multiple chucks, whose size is typically 128MB or 256MB. Each map task reads its chuck, performs the map operation, and writes intermediate files using the spill and merge operations [19]. Each reduce task performs the shuffle, merge and reduce operations in sequence, and finally generates output files.

Note that the spill and merge operations make use of the buffer mechanism. They keep data in the DRAM buffer and flush out all data when the buffer is almost full (specifically, when the size of the buffered data becomes larger than a threshold called *io.sort.spill.percent* in Hadoop [19]). Considering the buffering effect of the large intermediate and output files, I/Os triggered by a virtual machine tend to be bursty. This tendency has also

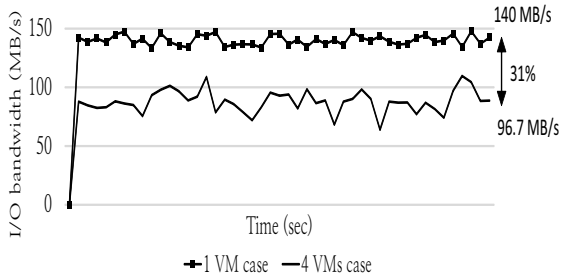


Fig. 2: Performance comparison between a single virtual machine and multiple virtual machines

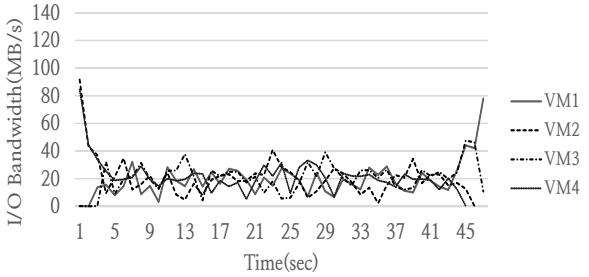


Fig. 3: I/O interference among multiple virtual machines

been observed in [23]. In addition, the name node dispatches and schedules the map tasks on virtual machines at the same time. It implies that the bursty I/Os are triggered by multiple virtual machines simultaneously.

Since virtual machines share the same physical resources, the bursty and concurrent I/Os generated by multiple virtual machines can cause the I/O performance degradation. To estimate this impact quantitatively, we measure the I/O bandwidth for two different cases. In the first case, a single virtual machine issues bursty I/Os alone. However, in the second case, four virtual machines issue bursty I/Os concurrently. To emulate bursty I/Os, we run the UNIX 'DD' command on each virtual machine that accesses a file with 1GB.

Figure 2 presents the measurement results. In the single virtual machine case, the I/O bandwidth is reported as 140MB/s, which is close to the upper bound supported by our experimental disk. On the contrary, in the four virtual machines case, the I/O bandwidth drops to the 96.7MB/s, which is the 31% reduction compared with the single virtual machine case.

To analyze the reasons for the performance degradation, we examine the I/O bandwidth of each virtual machine individually, as depicted in Figure 3. Each virtual machine achieves similar I/O bandwidth around 20∼25MBps, whose sum is 96.7MBps, which is quite smaller than that measured in the single virtual machine case.

Our sensitivity analysis shows that the performance degradation is due to the I/O interference among virtual machines. The control domain in XEN uses the CFQ (Complete Fairness Queuing) I/O scheduler for supporting fair I/O services to virtual machines [24]. It schedules I/Os in an interleaved manner among virtual machines, handling an I/O request from one virtual machine and the next one from the other virtual machine. In other words, I/Os triggered by a virtual machine is interfered by other I/Os triggered by different virtual machines.

In the control domain, I/Os triggered by a virtual machine are managed by the corresponding task, called *Blkback*, which will be discussed later in Figure 4. Therefore, the I/O interference among virtual machines incur frequent context switches among the Blkback tasks. Besides, since each virtual machine commonly uses a separated disk partition, the I/O interference causes long seek distances in a disk. These two overheads result in the performance degradation observed in Figure 2

3 Burstiness-aware I/O Scheduler

To alleviate the performance degradation due to the I/O interference among virtual machines, we propose a new burstiness-aware I/O scheduler. The key ideas of our proposed scheduler are as follows; 1) allows a bursty virtual machine to consume most of disk bandwidth for a given period exclusively to reduce the I/O interference, 2) when there are several bursty virtual machines, schedules them in a round-robin fashion to support fairness from the long-term viewpoint, and 3) reserves minimal disk bandwidth for non-bursty virtual machines to avoid starvation.

Figure 4 illustrates the overall structure of our proposed scheduler. It consists of three components: *Burstiness Detector*, *Coarse-Grained Dispatcher*, and *Starvation Handler*. Also, it has three control parameters: *B_threshold*, *S_quantum*, and *M_reserved*.

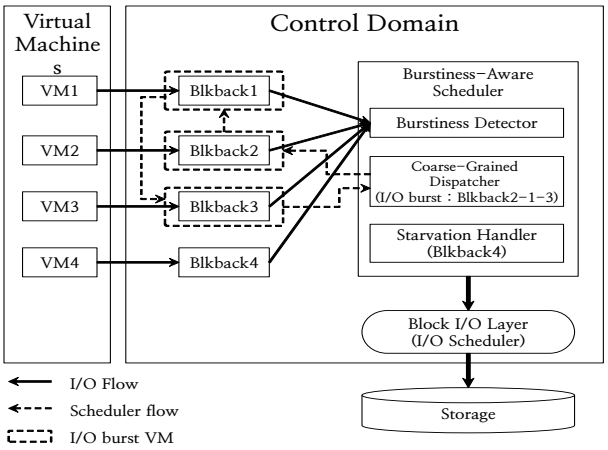


Fig. 4: Structure of the burstiness-aware I/O scheduler

When a virtual machine requests I/Os, it delivers them to the corresponding Blkback task in the control domain. The Blkback task is a kind of daemon process, actually performs I/Os on a disk on behalf of the related virtual machine. It sends the requested I/Os to the Block I/O layer that consists of the CFQ I/O scheduler and native device drivers. Our proposed scheduler is located between the Blkback tasks and the Block I/O layer, orchestrating the I/O flows between them.

Firstly, the Burstiness Detector examines I/Os managed by each Blkback task and determines whether the I/Os are bursty or not. In order to check the burstiness, we introduce the control parameter, B_threshold. When the total bandwidth required by a Blkback task is larger than the B_threshold, the task is treated as bursty.

When there are several bursty Blkback tasks, the Coarse-Grained Dispatcher schedules them in a round-robin fashion. The scheduled task is allowed to consume most of disk bandwidth during a given time quantum, namely S_quantum, which is our second control parameter. In Figure 4, we assume that the Blkback 1, 2 and 3 are determined as bursty and scheduled by the Coarse-Grained Dispatcher, as denoted the scheduling flow in the figure.

Finally, the Starvation Handler monitors the non-bursty tasks and handles their I/Os if they are read requests. But, the total I/Os that can be handled here are restricted to be under the M_reserved, minimal reserved bandwidth, to decrease the I/O interference caused by non-bursty tasks. This mechanism is devised to avoid the starvation situation of the non-bursty tasks. Note that a task may suffer from delayed services due to the restriction. However, the delayed requests enlarge the required disk bandwidth, which eventually leads the task to be determined as bursty and managed by the Coarse-Grained Dispatcher. If there is no bursty task, all I/Os are processed in the similar way as the original I/O scheduler does.

4 Experiments

We have implemented the burstiness-aware I/O scheduler in our experimental system depicted in Figure 1. We use the XEN hypervisor 4.1.4 as the virtualization software, Linux kernel 3.5.5 as GuestOS for both control domain and virtual machines. The Hadoop 1.1 is used as the MapReduce framework. Our proposed scheduler is implemented in the control domain, using the blkio subsystem supported by the control group mechanism [20]. This mechanism allows us to configure tasks as several groups and allocates I/O bandwidth to each group independently.

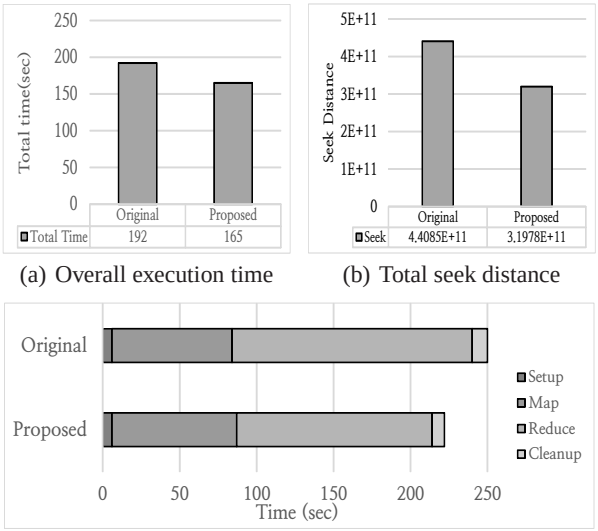
Physical resources and virtual resources per each virtual machine are summarized in Table 1. This table also shows how we set the control parameters of the proposed scheduler in this experiment.

Table 1: Experimental Environment

Virtualzation software		XEN-4.1.4
GuestOS		Linux-3.5.5
MapReduce framework		Hadoop-1.0.0
Workload		Terasort, Fio, IOzone
Physical resource	CPU	AMD Phenom Processor
	Memory	8 GB
	Disk	500 GB
Virtual resource	CPU	Credit scheduler
	Memory	1 GB
	Disk	100 GB
	Type	Full-Virtualization
Control Parameters	B_threshold	4 MB/s
	S_quantum	1 second
	M_reserved	4 MB/s

Figure 5 presents the performance evaluation results for the Terasort application [3] under the original and proposed burstiness-aware scheduler. The overall execution time is reduced from 192 to 165 seconds. This improvement is due to the reduction of the seek distance in a disk, as shown in Figure 5(b).

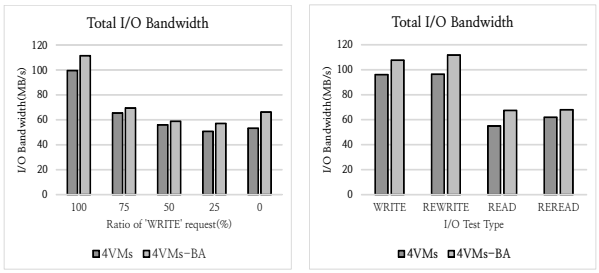
Figure 5(c) reveals an interesting point. Our proposal gives more opportunities to a bursty virtual machine for performing I/Os, which enables map tasks on the machine to be finished earlier. The earlier completion of map tasks triggers the beginning of reduce tasks earlier, which eventually shortens the execution time of the reduce phase. On the contrary, some map tasks on a



(a) Overall execution time (b) Total seek distance

(c) Execution time of each phase

Fig. 5: Terasort results: Overall execution time, seek distance and execution time of each phases



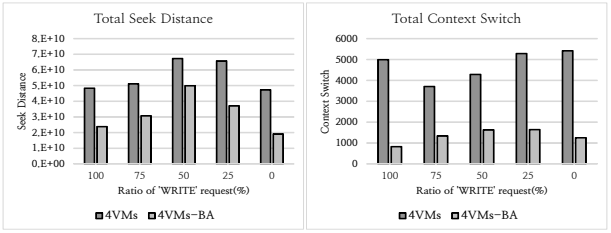
(a) FIO benchmark (b) IOzone benchmark

Fig. 6: Total I/O bandwidth

non-scheduled machine become straggled, which increases the execution time of the map phase shown in the Figure 5(c). Note that the execution of map and reduce phases are overlapped, hence the sum of the execution time of the phases is larger than the overall execution time.

In addition, we observe the I/O behavior and performance results vary noticeably each time we execute the Terasort application. This is because the execution time of the Terasort application depends not only on the I/O scheduler but also on Hadoop runtime decisions such as task allocation and scheduling [19].

To focus on the effect of I/O scheduling, we experiment with another two I/O-intensive benchmarks, namely Fio [21] and IOzone [22]. Specifically, we run each benchmark on the four virtual machines at the same time and measure the aggregate I/O bandwidth under the two conditions. One is based on the original I/O scheduler (denoted as 4VMs in the figure) and the other is based on our proposed burstiness-aware I/O scheduler (denoted as 4VMs-BA).



(a) FIO benchmark

(b) IOzone benchmark

Fig. 7: I/O interference: number of context switch and seek distance

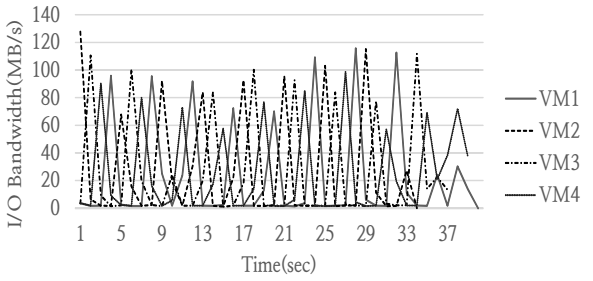


Fig. 8: Revisit the Figure 3 under the burstiness-aware I/O scheduler

Figure 6 shows the total I/O bandwidth for the two benchmarks. For Fio, we change the write ratio from 100 to 0 and measure the I/O bandwidth with and without the proposed burstiness-aware I/O scheduler. The results reveal that our proposal improves performance up to 24% with an average of 12%. The IOzone supports various operations such as write, rewrite, read, and reread. For these operations, our proposal enhances performance on average 15% and at maximum 22%.

Figure 7 presents the number of context switches among Blkback tasks and the seek distances during the execution of each benchmark on four virtual machines. From the results, we find out that the performance improvement of the proposed scheduler is due to two sources: one is reducing the context switch overhead and the other is decreasing the seek distances in a disk.

Figure 8 shows the I/O behavior under the proposed burstiness-aware I/O scheduler when we execute the same workloads observed in Figure 3. In comparing two figures, we notice that the proposed scheduler indeed allows a virtual machine to utilize most of the disk bandwidth exclusively during a given time quantum. This burstiness based coarse-grained allocation results in the reduction of I/O interference among virtual machines.

Figure 9 illustrates the amount of I/O bandwidth each virtual machine obtains. This figure shows that four virtual machines achieve similar I/O bandwidth. It implies that, even though our scheduler allows a virtual machine to monopolize most of the disk bandwidth during a given time quantum, it provides fairness at a long-term scale.

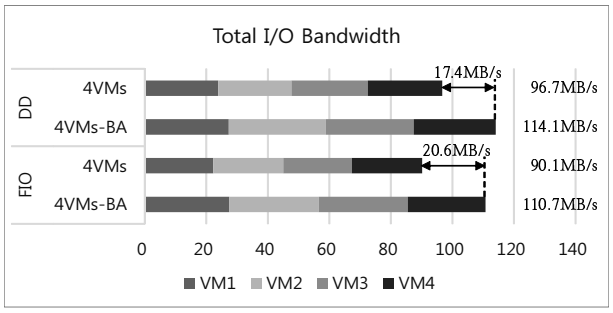


Fig. 9: Fairness under the burstiness-aware I/O scheduler

As a result, the proposed burstiness-aware scheduler can enhance both performance and fairness.

5 Related work

I/O performance in virtualized environments have been studied intensively in recent decades. Boutcher and Chandra examined whether the traditional disk I/O scheduling still provides benefit in a layered system consisting of virtual machines and underlying hypervisor [16]. They found out that using the default Linux schedulers does not appear to be the optimal and suggested several research areas of investigation such as reducing the number of transitions among virtual machines and hypervisor, and cooperative schedulers.

Yang et al. showed that the I/O performance of a virtual machine can be affected negatively by co-resident virtual machines [17]. Also, they proposed a framework, called vExplorer that can be used to understand the I/O scheduling characteristics within a hypervisor. Shafer showed that VMs can achieve only 51% and 77% of non-virtualized performance for storage writes and reads, respectively, becoming bottlenecks in cloud computing [18]. Kesavan et al. examined the VM-level I/O scheduler and its ability to enforce isolation and fair utilization [28].

Ongaro et al. explored the relationship between virtual machines scheduling in the XEN hypervisor and I/O performance [25]. They found that the credit scheduler used as default scheduler in XEN has a significant impact on I/O performance especially when bandwidth and latency-intensive applications are introduced. In addition, they suggested several optimization techniques such as sorting virtual machines based on their remaining credits and placing latency-sensitive applications in their own virtual machine.

Gulati et al. observed that the I/O throughput available to a virtual machine can fluctuate widely based on the behavior of other virtual machines accessing the shared disks [26]. To overcome this fluctuation, they presented a new I/O scheduler, called mclock that supports three controls, namely shares, limits, and reservations and provides predictable I/O allocation with strong isolation.

Seelam and Teller's study proposed a new scheduler, called VIOS (Virtual I/O Scheduler) that provides absolute performance virtualization and performance isolation [27]. It controls the coarse-grain allocation of disk time to the different virtual machines, whose idea is similar to ours. The difference between theirs and our approach is that ours orchestrates the I/O allocation adaptively based on burstiness monitored during the execution of virtual machines.

Running the MapReduce applications on virtual machines and investigating their performance have been studied actively. Matsunaga et al. developed the CloudBLAST that integrates the MapReduce, virtual machine and virtual network technologies for the execution of large-scale bioinformatics applications [13]. Park et al. examined the dynamic virtual machine reconfiguration technique for data-intensive computing on clouds [14].

Zaharia et al. noticed that virtual machines, which are used as data nodes in Hadoop on virtualized environments, are not homogeneous. This heterogeneity gives a negative effect to the speculative execution of stragglers. To overcome this problem, they proposed a new Hadoop scheduler, called LATE, that is robust to heterogeneity [5]. Kang et al. designed a novel Xen CPU scheduler, called MRG (MapReduce Group) that can reduce the domain switch overhead and support scalability and fairness in terms of the MapReduce cluster [30].

Ibrahim et al. investigated the impact of disk pair schedulers in Hadoop [15]. In their study, the term of the pair schedulers means a pair of disk schedulers: one within the hypervisor and the other within the virtual machines. They suggested an approach that tunes the pair schedulers adaptively during the execution of a MapReduce job for improving performance. Blagojevic et al. introduced the priority-based I/O scheduling that deliver priority information from Hadoop to kernel-level scheduler for enhancing I/O latency [29]. It is a kind of a cooperative scheduler across multiple layers. Fang et al. proposed two strategies to accelerate the performance of MapReduce applications on virtual machines [31]; one is dynamic adjusting the control domains weight and the other is selecting a physical machine as the master node.

6 Conclusion

In this paper, we explored the feasibility of the I/O intensive applications on virtualized environments. One concern is the I/O bottleneck, which is caused by the I/O interference among virtual machines, especially during the processing of big data. To mitigate this problem, we proposed a new I/O scheduler that can identify bursty virtual machines and give a chance for them to use most of the disk bandwidth without violating fairness.

We are considering two research directions as future work. I/O behavior in virtualized environments depends not only on the I/O scheduler but also on CPU scheduler for virtual machines and event delivery mechanism among virtual machines. The first research direction is to extend our proposal so that it can cover the interactions among these aspects. The second direction is applying our proposal to different types of storage systems such as SSDs (Solid State Drives) and SAN (Storage Area Network).

Acknowledgement

The present research was conducted by the research fund of Dankook University in 2014.

The authors are grateful to the anonymous referee for a careful checking of the details and for helpful comments that improved this paper.

References

[1] D. Agrawal, P. Bernstein, E. Bertino, S. Davidson, U. Dayal, M. Franklin, J. Gehrke, L. Haas, A. Halevy, J. Han, H. V. Jagadish, A. Labrinidis, S. Madden, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, K. Ross, C. Shahabi, D. Suciu, S. Vaithyanathan, and J. Widon, "Challenges and Opprotunities with Big Data", A community white paper developed by leading researchers across the United States, 2012.

[2] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", USENIX OSDI, 2004.

[3] Y.Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. K. Vunda, and J. Currey, "DryadLINQ: A system for General-Purpose Ditributed Data-Parallel Computing Using a High-Level Language", USENIX OSDI, 2008.

[4] Hadoop MapReduce Tutorial. http://hadoop.apache.org/docs/current/

[5] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments", USENIX OSDI, 2008.

[6] M. Rosenblum and T. Garfinkel, "Virtual Machine Monitors: Current Technology and Future Trends", IEEE Computer, pp. 39-47, May, 2005.

[7] 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", In Proceeding of SOSP, pp. 164-177, 2003.

[8] C. A. Waldspurger, "Memory Resource Management in VMware ESX Server", In Proceedings of SOSP, pp. 181-194, 2002.

[9] VMware white paper, "The CPU Scheduler in VMware ESX 4", http://www.vmware.com/files/pdf/perf-vsphere-cpu_scheduler.pdf.

[10] G. W. Dunlap, "Scheduler development update", www.xen.org/files/xensummit_intel09/George_Dunlap.pdf.

[11] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat, "Difference Engine: Harnessing Memory Redundancy in Virtual Machines", In Proceeding of OSDI, pp. 85-93, 2008.

[12] J. Liu, W. Huang, B. Abali and D. K. Panda, "High Performance VMM-Bypass I/O in Virtual Machines", In Proceeding of USENIX ATC, pp. 29-42, 2006.

[13] A. Matsunaga, M. Tsugawa and J. Fortes, "CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications", IEEE International Conference on eScience, 2008.

[14] J. Park, D. Lee, B. Kim, J. Huh, and S. Maeng, "Locality-Aware Dynamic VM Reconfiguration on MapReduce Clouds", ACM HPDC, 2012.

[15] S. Ibrahim, H. Jin, L. Lu, B. He and S. Wu, "Adaptive Disk I/O Scheduling for MapReduce in Virtualized Environments", IEEE ICPP, 2011.

[16] D. Boutcher and A. Chandra, "Does virtualization make disk scheduling passe?", ACM Operating System Review, **44**, 2010.

[17] Z. Yang, H. Fang, Y. Wu, C. Li, B. Zhao, and H. H Huang, "Understanding the Effects of Hypervisor I/O Scheduling for Virtual Machine Performance Interference", CloudComp, 2012.

[18] J. Shafer, "I/O Virtualization Bottlenecks in Cloud Computing Today", USENIX Workshop on I/O Virtualization (WIOV), 2010.

[19] H. Herodotou and S. Babu, "Profiling, Whatif Analysis, and Costbased Optimization of MapReduce Programs", VLDB, 2011.

[20] Control Group, http://www.kernel.org/doc/Documentation-/cgroups.

[21] Flexible I/O Tester Synthetic Benchmark, http://www.storagereview.com/fio_flexible_i_o_tester_synthetic_benchmark

[22] IOzone File System benchmark, http://www.iozone.org/

[23] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy and R. Sears, "MapReduce Online", NSDI, 2010.

[24] D. P. Bovet and M. Cesati, "Understanding the Linux Kernel (3rd Edition)", OREILLY, 2006.

[25] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling I/O in Virtual Machine Monitors", ACM VEE, 2008.

[26] A. Gulati, A. Merchant, and P. Varman, "mClock: Handling Throughput Variability for Hypervisor I/O Scheduling", USENIX OSDI, 2010.

[27] S. R. Seelam and P. J. Teller, "Virtual I/O scheduler: a scheduler of schedulers for performance virtualization", ACM VEE, 2007.

[28] M. Kesavan, A. Gavrilovska and K. Schwan, "On Disk I/O Scheduling in Virtual Machines", USENIX Workshop on I/O Virtualization (WIOV), 2010.

[29] F. Blagojevic, C. Guyot, Q. Wang, T. Tsai, R. Mateescu and Z. Bandic, "Priority IO Scheduling in the Cloud", HGST Research Paper,http://www.hgst.com/.

[30] H. Kang, Y. Chen, J. Wong, R. Sion and J. Wu, "Enhancement of Xens Scheduler for MapReduce Workloads", ACM HPDC, 2011.

[31] J. Fang, S. Tang, W. Zhou and H. Song, "Evaluating I/O Scheduler in Virtual Machines for MapReduce Applications", IEEE GCC, 2010.

Sewoog Kim received the BS degree in computer engineering from Chosun University, Korea, in 2007 and the MS degree in computer science from Dankook University, Korea, in 2012. He is currently Ph.D. student in computer science from Dankook University, Korea. His research interests include Storage, SSD, multi-core, linux kernel.

Dongwoo Kang received the BS and MS degrees in computer science from Dankook University, Korea, in 2009, 2010, respectively. He is currently Ph.D. student in computer science from Dankook University, Korea. His research interests include Storage, SSD, RT-OS, multi-core, scheduler, memory management.

Jongmoo Choi received the BS degree in oceanography from Seoul National University, Korea, in 1993 and the MS and Ph.D. degrees in computer engineering from Seoul National University in 1995 and 2001, respectively. He is an associate professor in the Department of Computer Science, Dankook University, Korea. Previously, He was a senior engineer at Ubiquix Company, Korea. He held a visiting faculty position at the University of California, Santa Cruz from 2005 to 2006. His research interests include microkernels, file systems, flash memory, and embedded systems.