PAPER

Revisiting I/O Scheduler for Enhancing I/O Fairness in Virtualization Systems

Sewoog KIM†a), *Member*, Dongwoo KANG†b), *and* Jongmoo CHOI†c), *Nonmembers*

SUMMARY As the virtualization technology becomes the core ingredient for recent promising IT infrastructures such as utility computing and cloud computing, accurate analysis of the internal behaviors of virtual machines becomes more and more important. In this paper, we first propose a novel I/O fairness analysis tool for virtualization systems. It supports the following three features: fine-grained, multimodal and multidimensional. Then, using the tool, we observe various I/O behaviors in our experimental XEN-based virtualization system. Our observations disclose that 1) I/O fairness among virtual machines is broken frequently even though each virtual machine requests the same amount of I/Os, 2) the unfairness is caused by an intricate combination of factors including I/O scheduling, CPU scheduling and interactions between the I/O control domain and virtual machines, and 3) some mechanisms, especially the CFQ (Completely Fair Queuing) I/O scheduler that supports fairness reasonable well in a non-virtualization system, do not work well in a virtualization system due to the virtualization-unawareness. These observations drive us to design a new virtualization-aware I/O scheduler for enhancing I/O fairness. It gives scheduling opportunities to asynchronous I/Os in a controlled manner so that it can avoid the unfairness caused by the priority inversion between the low-priority asynchronous I/Os and high-priority synchronous I/Os. Real implementation based experimental results have shown that our proposal can enhance I/O fairness reducing the standard deviation of the finishing time among virtual machines from 4.5 to 1.2.

key words: *storage, I/O virtualization, fairness, analysis tool, virtualization-aware I/O scheduler*

1. Introduction

Virtualization is a technology that allows multiple virtual machines to run concurrently on a physical machine by abstracting physical resources into multiple logical ones [1]. It plays a key role of utility computing, which flexibly provides computing resources for both computation and storage to users on demand. Also, it becomes the essential part for cloud computing owing to its merits such as elasticity, isolation, consolidation, and live migration [2].

The virtualization technology brings in a new software layer called hypervisor (also known as VMM (Virtual Machine Monitor)) that governs the behavior of virtual machines [3], [4]. The goal of the hypervisor is managing various logical resources efficiently and fairly among virtual machines. To achieve the goal, it makes use of a variety of policies and mechanisms. For instance, the borrowed virtual time scheduling [3], proportional-share algorithm [6] and virtualized multicore [7] are used for CPU virtualization, while the difference engine [8] and ballooning [4] are used for memory virtualization. Also, for I/O virtualization, the hypervisor makes use of the I/O emulation technique [9], IOMMU [10] and IDD (Isolated Device Domain) [3]. The IDD, which is commonly used in a XEN-based virtualization system, is an isolated and privileged I/O control domain that takes fully charge of the I/O device management.

This separation of the isolated software components, such as virtual machines, hypervisor and I/O control domain, provides several strengths including isolation, portability and easy development. However, the coexistence of these components requires interactions across protection domains, which causes performance and fairness issues, especially for I/O. For instance, an I/O request issued by an application is delivered to the corresponding I/O device through the virtual machine, hypervisor, and control domain using various complex data structures such as I/O ring and event channel [3]. Also, a response is returned in reverse order across the protection domains. These interpositions of the isolated domains in the I/O path often lead to performance degradation and unfairness in virtualization systems [12].

To explore these issues more quantitatively and with a system-wide viewpoint, we propose a novel virtualization-aware I/O fairness analysis tool. It consists of two modules, on-line monitor and off-line analyzer. The on-line monitor has two features, fine-grained and multimodal feature. It defines five focal layers, namely I/O request, I/O queuing, I/O dispatch, I/O completion and I/O response layer to examine I/O fairness in a fine-grained manner. In addition, it keeps track of not only I/O events but also scheduling data to provide multimodal information. The off-line analyzer has a multidimensional feature that presents the monitoring results with various viewpoints such as timeline or resource share.

Using this tool, we have conducted several observations based on our experimental XEN based virtualization system. Our observations have revealed that I/O fairness among virtual machines is broken frequently due to the combined reasons of diverse aspects including I/O scheduling, CPU scheduling and interactions between virtual machines and the I/O control domain. The superficial reason is due to the CPU scheduler in the XEN hypervisor that does not schedule some virtual machines during a specific period. Further investigation discloses that this non-scheduling is caused by the delay of I/O responses from the I/O control

Manuscript received March 5, 2014.
Manuscript revised July 16, 2014.
†The authors are with the Dankook University, Korea.
a) E-mail: ksu1024@gmail.com
b) E-mail: redcdang@gmail.com
c) E-mail: choijm@dankook.ac.kr (Corresponding author)
DOI: 10.1587/transinf.2014EDP7076

domain to virtual machines, which, in turn, makes the related virtual machines to be set as the inactive state.

The real reason behind this unfairness is due to the virtualization-unawareness of the CFQ (Completely Fair Queuing) I/O scheduler used in the I/O control domain. The CFQ I/O scheduler, which was originally designed for a non-virtualization system, supports fairness at the process basis, not at the virtual machine basis. This mismatch causes the I/O priority inversion problem, where the low-priority asynchronous I/Os hinder the progress of the high-priority synchronous I/Os, which eventually leads to the unfairness.

Our observation drives us to design a virtualization-aware I/O scheduler for enhancing fairness. The key idea is giving scheduling opportunity to asynchronous I/Os in a well controlled fashion so that they can be completed and their responses are delivered to the corresponding virtual machine. Then, the virtual machine becomes the active state, having a chance to be scheduled and to issue the pending I/O requests. To this end, we devise an integrated queue that orchestrates synchronous and asynchronous I/Os all together while handling synchronous I/Os with higher priority.

We have implemented our proposed I/O scheduler in a XEN-based virtualization system. The system equips with the quad-core 3.2 GHz Phenom processor, 8GB DRAM, and two WD 500GB hard disks. On this hardware platform, we install the XEN hypervisor version 4.1.2-pre and make one control domain and four virtual machines. The Linux kernel version 2.6.32.45 is used for the control domain and virtual machines. Experimental results have exhibited that our proposal can enhance I/O fairness without causing considerable management overheads.

This paper makes the following contributions:

- We design a new tool for analyzing I/O fairness in a virtualization environment. It gathers diverse information at the fine-grained level and visualize them with multiple viewpoints.
- We observe that virtual machines often suffer from I/O unfairness. This unfairness is the combined results of several virtualization techniques such as VM scheduling, I/O ring mechanism and I/O scheduling. To our knowledge, this is the first paper that investigates how the interactions among I/O scheduler, CPU scheduler, and inter-VM communication affect I/O unfairness in a virtualization system.
- We devise a virtualization-aware CFQ I/O scheduler that gives more scheduling chances to the asynchronous I/Os in a controlled manner, preventing the related virtual machine from being inactivated (not scheduled). Also, we claim that applying a scheme originally developed for a non-virtualization system into a virtualization system requires virtualization-awareness.
- We implement the tool and scheduler in a XEN-based real system and evaluate their effectiveness quantitatively.

This paper is organized as follows. In Sect. 2, we describe the background of this study. Then, the I/O fairness analysis tool are presented in Sect. 3. Analysis of I/O behaviors and details of our proposed I/O scheduler are elaborated in Sects. 4 and 5, respectively. In Sect. 6, we discuss experimental results. Related work is given in Sect. 7. Finally, conclusion and future work are summarized in Sect. 8.

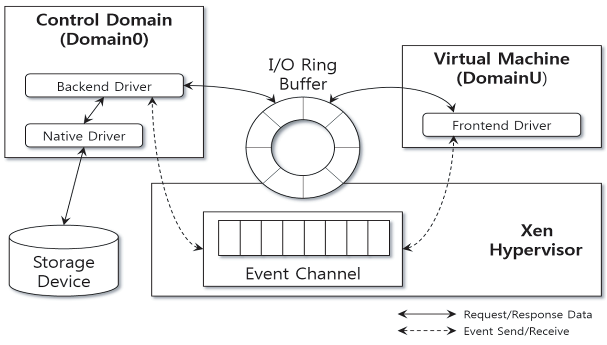


Fig. 1 I/O structure in a XEN-based virtualization system.

2. Background

In this section, we first explain an I/O path in a XEN-based virtualization system. Then, we present three mechanisms, namely communication mechanism, I/O scheduler, and CPU scheduler, that are closely related to the analysis of I/O fairness.

Figure 1 shows the structure of a XEN-based virtualization system. It consists of three types of isolated software component, XEN hypervisor, control domain, and a couple of virtual machines. The control domain and virtual machine are also called as Domain 0 and Domain U, respectively, in XEN's nomenclature. Each virtual machine can execute any operating system (also called as Guest OS) such as MS windows and Linux, supporting an independent computing platform. The control domain is in charge of managing I/O devices and carries out I/O accesses on behalf of Guest OSes.

The XEN hypervisor employs the split device driver model [3]. It supports two types of driver, a front-end driver in a Guest OS and a back-end driver in the control domain. For communication between the two drivers, the XEN hypervisor provides two mechanisms, I/O ring and event channel. The former is used for data transfer while the latter for notification. When a I/O request is triggered by an application, the front-end driver in a Guest OS puts the request into the I/O ring and notifies it to the back-end driver through the event channel. Then, the back-end driver delivers the request to the native driver that actually performs I/O operations by directly manipulating related registers in devices. When the request is completed, a response is returned through the inverse order; device, control domain, hypervisor, and Guest OS. Note that the I/O ring and event channel are limited resources. Hence, when they are full, a virtual machine can not send further requests even though it has lots of pending I/O requests.

One of the mechanisms that affects I/O fairness is I/O scheduling. In the control domain, a I/O scheduler locates between the back-end driver and the native driver. In this experimental environment, we use the Linux kernel version 2.6.32.45 as the operating system for the control domain. In Linux, there are four types of I/O scheduler; Noop, deadline, anticipatory, and CFQ (Completely Fair Queuing) [13].

The Noop scheduler arranges requests based on the FCFS (First Come First Serve) algorithm. Both the deadline and anticipatory schedulers are trying to minimize head movements by serving requests in ascending (or descending) order of their sector numbers. The anticipatory goes one step further in that it waits for new in-coming read requests for a predetermined period for handling bursty I/Os more efficiently. Finally, the CFQ scheduler provides a separate queue for each process and serves requests of queues in a round-robin order which enables to support fair share of I/Os among processes. The default one used in the control domain is the CFQ scheduler.

Another mechanism that affects I/O behaviors is CPU scheduling. The XEN hypervisor originally used the BVT (Borrowed Virtual Time) scheduler [3], but recently adopts the credit scheduler [14]. The credit scheduler manages virtual CPUs based on their credits, a kind of time quantum. The credit determines the priority of virtual CPUs; BOOST, UNDER and OVER. When a virtual CPU has a remaining credit, it has the UNDER priority, meaning schedulable. Otherwise, it becomes the OVER priority. Finally, a virtual CPU has the BOOST priority when it wakes up from a long sleep for giving more opportunity to handle the pending I/Os.

In addition to the priority, each virtual CPU has its own state; active or inactive. When a virtual CPU is idle for a long time (specifically, when it accumulates a credit above a 30ms worth [14]), the XEN credit scheduler marks it as an inactive state and excludes it from scheduling candidates.

3. I/O Fairness Analysis Tool

In this paper, we propose a novel virtualization-aware I/O fairness analysis tool, that provides the following three features. The first feature is a fine-grained monitoring. One characteristic of a virtualization system is the coexistence of multiple isolated software components such as virtual machine, control domain, and hypervisor. The processing steps of I/O requests are interposed across several protection domains. Therefore, to perform I/O analysis more accurately, it is indispensable to investigate I/O behaviors down to the details at the appropriate points.

As those appropriate points, we define five focal layers, namely I/O Request layer, I/O Queuing layer, I/O Dispatch layer, I/O Completion layer, and I/O Response layer, as shown in Fig. 2. Each layer represents an inter-relation between isolated components. For instance, the I/O Request layer is a point where we can examine the interactions between a Guest OS and the XEN hypervisor. Similarly, we can inspect the interactions between the XEN hypervisor

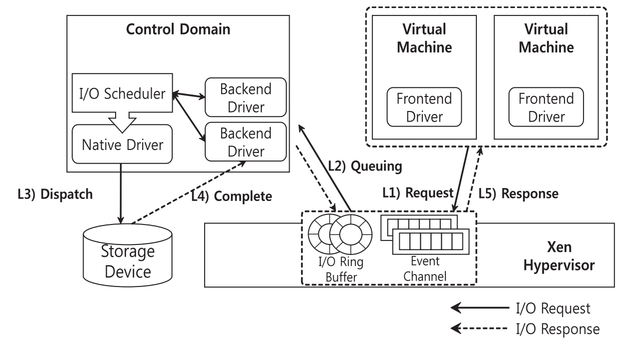


Fig. 2 File layers for fine-grained I/O analysis.

and control domain at the I/O queuing layer, the interactions between the control domain and devices at the I/O Dispatch layer, and so on.

The strong point of this fine-grained analysis is that it can disclose useful information on the potential causes when it monitors unfairness at each layer. For instance, when we observe a fairness violation at the I/O Request layer, we can infer that the delayed virtual machine suffers from the shortage of the virtual CPU, possibly due to the lack of virtual CPU's credit. Also, when we observe a fairness anomaly at the I/O Queuing layer, we can suspect the I/O ring or event channel handling mechanisms. Similarly, unfairness observed at the I/O Dispatch, Completion, and Response layer indicates that there might be some causes or problems in the I/O scheduling policies, device characteristics and optimization techniques, and inter-domain communication mechanisms, respectively. This inference gives us a relevant clue to understand and solve a fairness violation.

The second feature of our proposed tool is a multimodal monitoring. Note that the I/O fairness analysis is closely associated with CPU scheduling decisions. Hence, to carry out I/O analysis in a system-wide viewpoint, the tool needs to monitor not only I/O related but also CPU related data. Specifically, it monitors I/O related data occurred in the I/O ring, event channel, back-end driver's queue, and native driver's queue. Besides, it collects the virtual CPU status of each virtual machine such as credit, priority and activity. The collected data are stored in main memory. Since the overhead for collecting is quite small, compared with the processing time of an I/O request, we expect that this monitoring rarely affects the I/O analysis results.

The third feature is a multidimensional presentation. It shows the records with various aspects. One is a time dimension. It displays I/O behaviors as time goes by, which will be illustrated in Fig. 3. Another aspect is a share dimension. It shows how much I/O bandwidth a virtual machine uses at a certain time or period, which will be illustrated in Fig. 4. The final aspect is an integrated dimension. It depicts I/O behaviors and CPU scheduling information at the same time scale, which will be further discussed using Fig. 5.

4. Analysis Results

We have implemented the tool and conducted several ob-

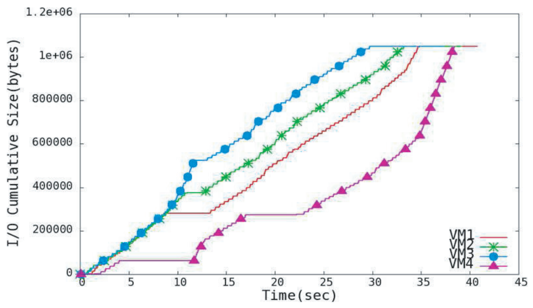


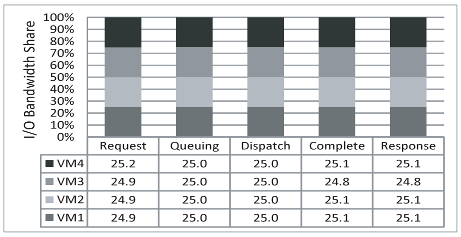
Fig. 3 I/O fairness comparison among 4 virtual machines.

servations based on our experimental virtualization system, consisting of the quad-core 3.2 GHz Phenom processor, 8GB DRAM, and two WD 500GB hard disks. On this hardware platform, we install the XEN hypervisor version 4.1.2-pre and configure one I/O control domain and four virtual machines. The XEN hypervisor creates eight virtual CPUs and assigns four into the control domain and other four into the four virtual machines, separately. Also, it allocates 1GB memory to each virtual machine. Finally, it divides a disk into four partitions with the size of 125GB and allots a partition into a virtual machine. As the results, each virtual machine has one virtual CPU, 1GB memory and 125GB disk space. On this virtual machine, we install the Linux kernel version 2.6.32.45, the same one used by the I/O control domain.

Then, we execute the Linux 'dd' command in each virtual machine that reads a file with the size of 1GB at the same time. While executing, the tool monitors various information such as the number of I/O requests at the five layers, I/O size, elapsed time and CPU scheduling data. Figure 3 presents one monitoring result with a timeline viewpoint, where the x-axis is the elapsed time and the y-axis is the cumulative I/O size. In the figure, four lines, denominated as VM 1 to 4, represent the four virtual machines.

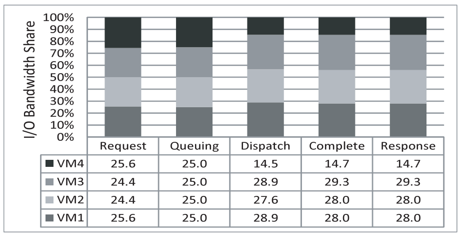
Figure 3 shows that there is indeed a fairness violation among the virtual machines. In this case, the finishing time of the command in the virtual machine 4 is around 38 second, which are 1.31 times slower than the command in the virtual machine 1, whose finishing time is 29 second. We also observe that such unfairness occurs frequently, roughly 4 out of 5 trials even though the 'dd' application performs I/Os only without any complex CPU or memory operations and the CFQ is used as the default I/O scheduler. For the comparison purpose, we conduct the same experiment in a non-virtualization system, installing the identical Linux kernel on the aforementioned hardware platform and executing the 'dd' command using the four tasks. In this case, we hardly observe the unfairness. It implies that the unfairness observed in Fig. 3 is mainly due to the software complexity introduced by the virtualization technology.

For further investigation, we examine I/O behaviors with a different viewpoint by exploiting the multidimensional feature of our proposed tool. Figure 4 illustrates the monitoring results in a share dimension, while Fig. 3 is illustrated in a time dimension. Specifically, we plot the I/O bandwidth shares of the four virtual machines at the interesting moments, 15, 17 and 20 second, respectively, at the timeline of Fig. 3. Figure 4 (a) shows the shares at the 15 second. Obviously, the four virtual machines consume I/O bandwidth fairly through all five focal layers. However, at the 20 second as presented in Fig. 4 (c), the virtual machine 4 does not use I/O bandwidth at all. Interestingly, this phenomenon is observed from the first layer that is the I/O Request layer.



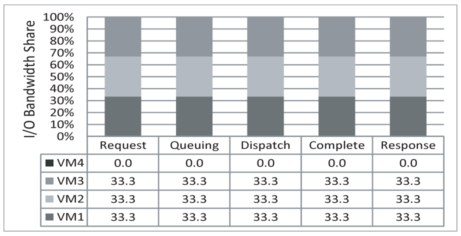
	Request	Queuing	Dispatch	Complete	Response
■ VM4	25.2	25.0	25.0	25.1	25.1
■ VM3	24.9	25.0	25.0	24.8	24.8
■ VM2	24.9	25.0	25.0	25.1	25.1
■ VM1	24.9	25.0	25.0	25.1	25.1

(a) 15second



	Request	Queuing	Dispatch	Complete	Response
■ VM4	25.6	25.0	14.5	14.7	14.7
■ VM3	24.4	25.0	28.9	29.3	29.3
■ VM2	24.4	25.0	27.6	28.0	28.0
■ VM1	25.6	25.0	28.9	28.0	28.0

(b) 17second



	Request	Queuing	Dispatch	Complete	Response
■ VM4	0.0	0.0	0.0	0.0	0.0
■ VM3	33.3	33.3	33.3	33.3	33.3
■ VM2	33.3	33.3	33.3	33.3	33.3
■ VM1	33.3	33.3	33.3	33.3	33.3

(c) 20second

Fig. 4 I/O bandwidth shares among 4 virtual machines at the five layers.

At the beginning, we suspect that this unfairness is due to the XEN CPU scheduling policy since the 'dd' application tries to trigger I/Os whenever possible. As we already discussed in Sect. 2, the XEN hypervisor uses the credit scheduler with three priorities; UNDER, OVER, and BOOST [14]. We doubt that the virtual machine 4 is in the OVER priority, that means no remaining credit, which leads to the lack of the share at the I/O Request layer.

To explore our speculation, we observe the virtual CPU information during the 'dd' execution, as depicted in Fig. 5. Note that the scale of the x-axis of this figure is the same as that of Fig. 3, owing to the multimodal feature of our tool. Unfortunately, it shows that our speculation is incorrect. The virtual CPU for the virtual machine 4 has a positive credit, resulting in having the UNDER or BOOST priority, as shown at the middle graph in Fig. 5.

The actual reason of the unfairness at the I/O request layer is the activity state of the virtual CPU, as shown at the

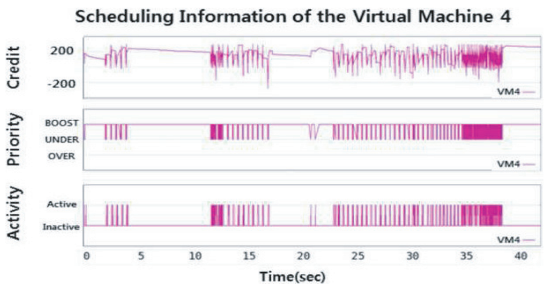


Fig. 5 CPU scheduling information for the virtual machine 4.

bottom graph in Fig. 5. When a virtual CPU is idle for a long time due to the lengthy delay of I/Os, the XEN credit scheduler marks it as an inactive state and excludes it from scheduling candidates. When an interrupt is occurred and there is a pending job in the event channel, it becomes an active state and gets a chance to be scheduled according to its priority. Since the virtual CPU related to the virtual machine 4 is in the inactive state during the time interval between 4∼12 and 17∼21 second, it cannot use the I/O bandwidth during those intervals.

Now the question is why the virtual CPU becomes inactive. This is because the responses of the previously issued requests are delayed from the control domain to the virtual machine 4. We perform a time travel from 20 second, as shown in Fig. 4 (c), to 17 second, as shown in Fig. 4 (b), which is the starting point of the unfairness presented in Fig. 3. Figure 4 (b) shows that the I/O bandwidth share is reduced between the I/O Queuing and Dispatch layer, from 25% to 14% in the virtual machine 4. When we plot this figure, we use the moving averaged value of successive monitoring results. If we plot using the monitoring results individually, the share often goes down to 0%, not dispatched at all even though there are I/O requests in the queue of the back-end driver.

These phenomena are more evident when we examine the raw collected data directly, presented in Table 1 and illustrated in Fig. 6. It shows that the request listed at the 3rd entry in the table is inserted into the queuing layer at 17.38196 second. Then, the request is moved into the dispatch layer at 17.38198 second by the CFQ I/O scheduler. However, the request listed at the 4th entry is moved into the dispatch layer at 20.53013 second even though it is inserted into the queuing layer at 17.38266 second. There exist considerable gap, roughly 3 seconds. This delayed dispatching defers the completion of this request, which making the following requests to be queued after that gap, as shown in the 6th entry. In other words, the delayed dispatching postpones the completion response from the control domain to the virtual machine 4, which eventually makes the virtual machine 4 as inactive state, shown in Fig. 5.

Figure 4 (b) and Table 1 reveal that the real cause of the unfairness is the CFQ I/O scheduler that governs the dispatching decisions between the queuing and dispatch layers. These decisions affect the communication between the control domain and virtual machines, which, in turn, influence

Table 1 I/O Information of the virtual machine 4.

(a) Queueing Layer

Entry	Time (sec)	RW	Size	Sector
1	17.38099	Read	45056	336010368
2	17.38106	Read	45056	336010456
3	17.38196	Read	45056	336010544
4	17.38266	Write	4096	367018256
5	17.38269	Write	20480	371476136
6	20.53105	Read	45056	336010624
7	20.53109	Read	45056	336010712

(b) Dispatch Layer

Entry	Time (sec)	RW	Size	Sector
1	17.38101	Read	45056	336010368
2	17.38108	Read	45056	336010456
3	17.38198	Read	45056	336010544
4	20.53013	Write	4096	367018256
5	20.53049	Write	20480	371476136
6	20.85929	Read	45056	336010624
7	20.85929	Read	45056	336010712

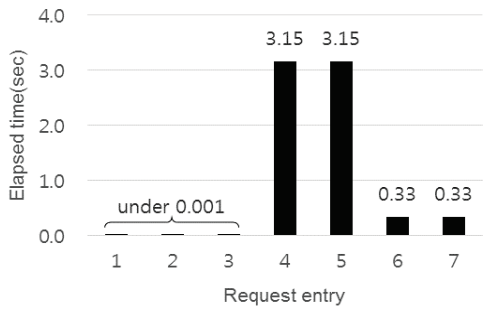


Fig. 6 Elapsed time between layers (numbers are differences between Queueing time and Dispatching time at eacy entry).

the XEN CPU scheduling decisions. Our further examination shows that the problem of the CFQ I/O scheduler is due to the virtualization-unawareness, which will be discussed in details in the next section.

5. Virtualization-Aware I/O Scheduler

Figure 7 shows the details of the CFQ I/O scheduler used in the I/O control domain. The CFQ I/O scheduler has two objectives; 1) providing fair share of I/O bandwidths among processes that have the same priority and 2) preferring synchronous I/Os to asynchronous I/Os. Note that, in general, read requests are classified into synchronous, while writes into asynchronous. To accomplish these objectives, the CFQ I/O scheduler assigns different queue into each process for synchronous I/O requests and serves them in a round-robin order. Also, it use another separated queue (or queues) for asynchronous I/O requests and serve them only when the synchronous queues are all empty.

The CFQ I/O scheduler works well in a non-virtualization system [13]. However, in a virtualization system, some conditions assumed by the scheduler are not valid. Specifically, in the control domain, each back-end driver shown in Fig. 2 is treated as a process. However,

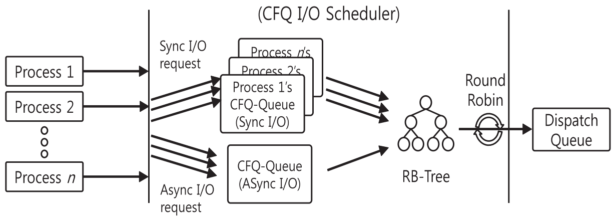


Fig. 7 Interaction among processes and CFQ I/O scheduler in a non-virtualization system.

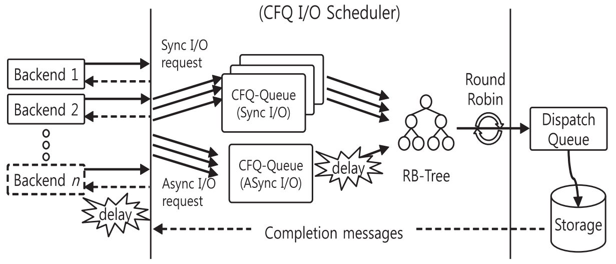


Fig. 8 Interaction among backend drivers and CFQ I/O scheduler in a virtualization system.

in actuality, the back-end driver corresponds to a virtual machine, not a process, as illustrated in Fig. 8. A virtual machine consists of a set of heterogeneous processes, issuing various types of synchronous and asynchronous I/Os together. In addition to this conceptual mismatch, in a virtualization system, I/O requests and responses need to be transferred between a virtual machine and a back-end driver through the inter-domain communication mechanisms such as I/O ring and event channel. Since the capacity of the I/O ring and event channel is limited, sending I/O requests can be blocked until responses for the previously issued requests are returned to a virtual machine.

Let us elaborate the observed unfairness using example with Fig. 8. Assume that a virtual machine sends some read and write requests together. Then, read requests are inserted into the corresponding synchronous queue while write requests are into the asynchronous queue. The synchronous requests are served in a round-robin fashion while the asynchronous requests wait until the synchronous queues are empty. By the way, when another read requests are issued by other virtual machines, the dispatching of the asynchronous requests are delayed further. This delayed dispatching, in turn, postpones the completion of the asynchronous requests. If the postponement is longer than a certain threshold, it makes the virtual machine as inactive as observed in Fig. 5. As the result, the virtual machine cannot send the remaining I/Os to the control domain, as shown in Fig. 4 (c), until the completion of the postponed I/Os.

Note that, in this example, the low-priority asynchronous I/Os are delayed by synchronous I/Os issued from other virtual machines, which eventually blocks the progress of the high-priority synchronous I/Os, requested later from the same virtual machine. It looks like the well-known priority inversion problem in a real-time system [20]. We call

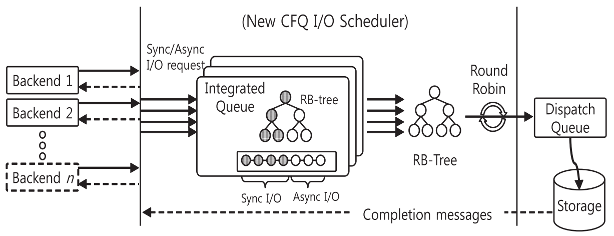


Fig. 9 Virtualization-aware CFQ I/O scheduler based on Integrated Queue.

this situation as the I/O priority inversion problem in a virtualization system.

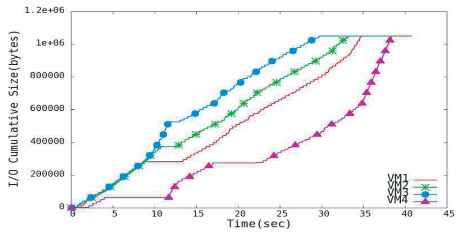
To overcome this problem, we propose a new virtualization-aware CFQ I/O scheduler. The key idea of our proposal is giving more scheduling opportunity to asynchronous I/Os in a controlled manner. The controlled manner implies the following two principles; 1) still giving preference to synchronous I/Os in the same virtual machine and 2) avoiding starvation of asynchronous I/Os due to synchronous I/Os issued from other virtual machines.

To materialize the principles, we devise an integrated queue, as depicted in Fig. 9. The integrated queue, that is assigned into each back-end driver separately, contains both synchronous and asynchronous I/Os. In the queue, synchronous I/Os get higher priority than asynchronous ones using RB-tree structure. Also, all queues are served in a round-robin manner. Hence, when there exist asynchronous I/Os only in a queue, they can be dispatched and served, enabling to respond the completion message to the related virtual machine. Hence, the virtual machine can send remaining I/Os without blocking.

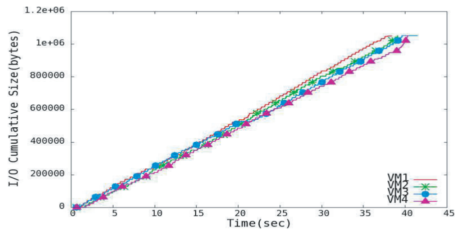
6. Experimental Results

To validate the effectiveness of the virtualization-aware CFQ I/O scheduler, we have implemented the scheduler in our experimental system and conducted the same experiment, discussed in Fig. 3. Note that the workload used in this experiment also has been used in several previous studies [21], [22] since it represents the characteristics of large and sequential I/Os observed in a virtualization environment [23].

Figure 10 presents the experimental result. Figure 10 (a) is the I/O behaviors under the traditional CFQ I/O scheduler while Fig. 10 (b) is those under our proposed scheduler. From the figure, we notice that our proposal indeed enhances the fairness among virtual machines reducing the standard deviation of the finishing time of applications on virtual machines from 4.5 to 1.2. It implies that our proposal can overcome the I/O priority inversion problem in a virtualization system. It is noteworthy that asynchronous I/Os occurs frequently from the kthread daemon or journaling requests from the Ext4 file systems, even though users request synchronous I/Os only. The result also uncovers that our proposed analysis tool supports quite relevant and useful information for analyzing I/O behaviors in a virtualization

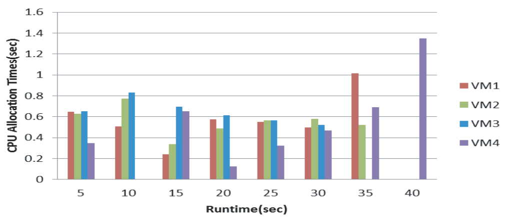


(a) CFQ I/O Scheduler

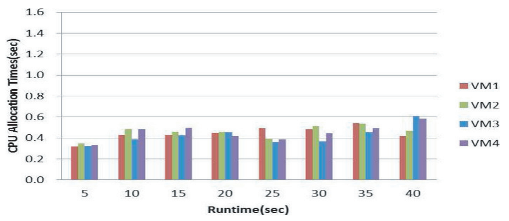


(b) Intergrated Queue based CFQ I/O Scheduler

Fig. 10 Comparison of I/O cumulative size among virtual machines.



(a) CFQ I/O Scheduler



(b) Intergrated Queue based CFQ I/O Scheduler

Fig. 11 Comparison of CPU allocation among virtual machines.

system.

One concern of our proposed scheduler is that it may hurt the overall performance. Note that the finishing times of the four applications in Fig. 10 (a) are relatively shorter than those in Fig. 10 (b). One reason is that giving a dispatching chance to asynchronous I/Os increases the seek distance for handling those requests, reducing the possibility of the seek optimization in devices. Another and more important reason is that the finishing times measured in Fig. 10 (b) includes the completion of the background asynchronous I/Os while those in Fig. 10 (a) do not. Achieving both performance and fairness is left as future work.

Figure 11 shows the CPU allocation among virtual machines during the experiment. It shows that the virtualization-aware CFQ I/O scheduler can handle the I/O requests of virtual machines fairly, which, in turn, balances the CPU allocation among virtual machines. It also shows that I/O scheduling decisions affects not only I/O fairness but also CPU allocation decisions, and vice versa. This is why we design the I/O fairness analysis tool with the consideration of multimodal feature.

7. Related Work

As the I/O virtualization becomes an important issue in term of performance and fairness in a virtualization system, several notable researches have been carried out over the last decades [12]. Barham et al. have proposed the IDD (Isolated Device Domain) and split-driver model for making use of the reusability of native drivers [3]. I/O emulation and dynamic translation have been suggested in [9] and [4], respectively, where I/O requests of a virtual machine are trapped into the hypervisor, which makes direct access to I/O devices.

Ben-Yehuda et al. have suggested the IOMMU (IO Memory Management Unit) for safe direct device access from a virtual machine [10]. Liu et al. have designed a VMM-Bypass technique that allows time-critical I/O operations to be carried out directly in a virtual machine without involvement of the hypervisor or the control domain [11]. Kim and Lee have proposed a novel distributed hash table based mechanism for flexible storage virtualization [15].

Santos et al. have observed the overheads due to the interactions among virtual machines, control domain and hypervisor, and have proposed several optimization techniques such as a lightweight page grant mechanism [18]. Gordon et al. have proposed ELI (Exit Less Interrupt) that handles interrupts within a virtual machine without involvement of the hypervisor [19].

Two researches are closely related to our work. Gulati et al. have devised an algorithm, called mClock, that supports proportional-share fairness on the IO allocations for virtual machines [16]. Kim et al. have noticed that the XEN CPU scheduler can affect the I/O performance and fairness among virtual machines [17].

The schemes proposed in [16] and [17], and our scheme have the same goal, enhancing I/O fairness and performance in a virtualization system. The difference between [17] and ours is that our scheme is an I/O scheduler, while [17] is a VM scheduler. Compared with [16], our scheme does not require any QoS related parameters, while [16] can provide more flexible I/O controls among VMs. We think that these schemes are orthogonal and can be used independently each other, creating a synergy effect.

In actuality, this paper is an extended version of the paper appeared in [5]. In [5], we have proposed the analysis tool only, while in this paper, we investigate the analysis results in depth, design a new virtualization-aware I/O scheduler, and evaluate its effectiveness through implementation-based experiments.

8. Conclusion

As the usage of the virtualization technology increases, a bunch of mechanisms that are used popularly in a non-virtualization system are migrated into a virtualization system. However, a naïve migration incurs unexpected performance degradation and/or fairness violation due to the environmental and conceptual mismatches between two systems. In this paper, we have proposed a new I/O fairness analysis tool to examine such mismatches with fine-grained, multimodal and multidimensional viewpoints. Also, we have designed a virtualization-aware CFQ I/O scheduler for enhancing fairness among virtual machines.

We are considering three research directions as future work. The first direction is further evaluating the effectiveness of the proposed tool with various applications including I/O and CPU intensive workloads. Another direction is analyzing the tradeoffs between performance and fairness, and exploring new scheme to enhance both. The final direction is extending our virtualization-aware CFQ I/O scheduler to guarantee the different QoS (Quality of Service) requested by cloud computing users.

Acknowledgements

This work was supported partly by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2012R1A2A2A01014233) and by the Industrial Strategic technology development program (10039143, Multi-Gbps Parallel I/O Virtualization and DB PerformanceOptimization for the Data-centric Computing on Many-core System) funded by the Ministry of Knowledge Economy, Korea, and the present research was conducted by the research fund of Dankook University in 2013.

References

[1] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: Current technology and future trends," Computer, vol.38, pp.39–47, May 2005.
[2] F. Huang, D. Li, Z. Chen, and X. Lu, "iVDA: A efficient and load-balanced virtual machine deployment algorithm in large cloud environment," Information Journal, vol.15, no.2, pp.831–846, 2012.
[3] 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," Proc. SOSP, pp.164–177, 2003.
[4] C.A. Waldspurger, "Memory resource management in VMware ESX server," Proc. SOSP, pp.181–194, 2002.
[5] S. Kim, D. Kang, and J. Choi, "Fine-grained I/O fairness analysis in virtualized environment," Proc. ACM RACS, pp.403–408, 2012.
[6] VMware white paper, "The CPU scheduler in VMware ESX 4," http://www.vmware.com/files/pdf/perf-vsphere-cpu-scheduler.pdf
[7] Z. Shao, H. Jin, Y. Li, and J. Huang, "XenMVM: Exploring potential performance of virtualized multi-core systems," Information Journal, vol.14, no.7, pp.2315–2326, 2011.
[8] 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," Proc. OSDI, pp.85–93, 2008.
[9] J. Sugerman, G. Venkitachalam, and B. Lim, "Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor," Proc. USENIX ATC, pp.1–14, 2001.
[10] M. Ben-Yehuda, J. Mason, J. Xenidis, O. Krieger, L. van Doorn, J. Nakajima, A. Mallick, and E. Wahlig, "Utilizing IOMMUs for virtualization in Linux and Xen," Ottawa Linux Symposium (OLS), 2006.
[11] J. Liu, W. Huang, B. Abali, and D.K. Panda, "High performance VMM-bypass I/O in virtual machines," Proc. USENIX ATC, pp.29–42, 2006.
[12] C. Waldspurger and M. Rosenblum, "I/O virtualization," Commun. ACM, vol.55, no.1, pp.66–72, 2012.
[13] D.P. Bovet and M. Cesati, Understanding the Linux Kernel, 3rd ed., O'Reilly, 2006.
[14] G.W. Dunlap, "Scheduler development update," http://www.xen.org/files/xensummit_intel09/George_Dunlap.pdf
[15] J. Kim and S. Lee, "Efficient and flexible storage virtualization system based on distributed Hash tables," Information Journal, vol.16, no.1(A), pp.383–392, 2013.
[16] A. Gulati, A. Merchant, and P. Varman, "mClock: Handling throughput variability for hypervisor IO scheduling," Proc. OSDI, pp.437–450, 2006.
[17] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee, "Task-aware virtual machine scheduling for I/O performance," Proc. VEE, pp.101–110, 2009.
[18] J.R. Santos, Y. Turner, G. Janakiraman, and I. Pratt, "Bridging the gap between software and Hardware techniques for I/O virtualization," Proc. USENIX ATC, pp.29–42, 2008.
[19] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, A. Schuster, and D. Tsafrir, "ELI: Bare-metal performance for I/O virtualization," Proc. VEE, pp.411–422, 2012.
[20] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," IEEE Trans. Comput., vol.39, no.9, pp.1175–1185, 1990.
[21] M. Zaharia, A. Konwinski, A.D. Joseph, R.H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," OSDI, 2008.
[22] L. Cherkasova and R. Gardner, "Measuring CPU overhead for I/O processing in the XEN virtual machine monitor," USENIX ATC, 2005.
[23] F. Meng, L. Zhou, X. Ma, S. Uttamchandani, and D. Liu, "vCache-Share: Automated server flash cache space management in a virtualization environment," USENIX ATC, 2014.

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 division of information and 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.