ENGINEERING JOURNAL

Evaluating I/O Scheduling in Virtual Machines Based on Application Load

Peng Zhao*, and Guoquan Tan

China Telecom Corporation Limited, Beijing Research Institute, No. 118, Xizhimenneidajie, Xicheng District, Beijing 100035, China
*E-mail: zhaopeng@ctbri.com.cn (Corresponding author)

Abstract. In recent years, cloud computing services and virtualization technology have been widely used. Virtualization requires the access to underlying resources to go through a virtualization layer, which reduces the operation efficiency, especially the access to disk I/O will easily become the bottleneck of the whole system. Therefore, how to improve the I/O performance of virtualization applications has become a hot spot in current researches, especially on I/O scheduling algorithm. While the design and selection of traditional I/O scheduling algorithms are greatly restricted by the seek time and latency of the underlying disks, the virtualization layer in a virtual environment to some extent shields the perception of the scheduling algorithm of virtual machines on the characteristics of the underlying hardware. Whether the traditional algorithms are applicable and how the multi-layer I/O scheduling system in virtualization collaborates to better meet the I/O performance requirements have become pressing issues. In this paper, the authors will explain how the I/O scheduler in Linux system works under different application loads in two scenarios (real machine and virtual machine), and take open-source Xen as examples to test and evaluate the influence of combination of the Dom0 scheduling algorithm and the virtual domain scheduling algorithm on I/O performance under different application loads, and then put forward the preferred proposals of I/O scheduler in virtual domains.

Keywords: I/O performance, Xen, application load, scheduling algorithm.

ENGINEERING JOURNAL Volume 17 Issue 3
Received 17 September 2012
Accepted 13 February 2013
Published 1 July 2013
Online at http://www.engj.org/
DOI:10.4186/ej.2013.17.3.105

DOI:10.4186/ej.2013.17.3.105

1. Introduction

Since its introduction over 30 years ago, virtualization technology has not been widely applied and promoted until recent years. Now it becomes an important means for enterprises to achieve IT resources reuse, smooth expansion and enhance the convenience of maintenance. Virtualization technology can be classified in a variety of ways. Usually it can be divided into full virtualization, para-virtualization (PV) and hardware-assisted virtualization by the hardware access mode. The Xen PV architecture featured with driver separation has received extensive attention since being proposed. Now, it has become the first choice among a large number of virtualization solutions and one of the mainstream technical architectures in the virtualization field for its good performance.

Xen technical architecture has introduced a hierarchical model of disk I/O scheduling. The first layer schedules different virtual domains with fairness as the preferred principle, which is completed by the credit scheduler in Xen Hypervisor. The next layer is a two-level scheduling system composed of the I/O scheduler in Dom0 and I/O scheduler in virtual domains, in which traditional Linux scheduling algorithms, i.e., noop, deadline, anticipatory and cfq are still adopted. These scheduling algorithms are assumed to run in a real physical device and perceive the device's physical features, such as disk latency, which certainly do not apply to a virtual environment. Give the consideration of virtualization and the competition for resources of other virtual domains, the I/O performance of applications in virtual machines becomes weakened and unpredictable, especially for applications that are sensitive to bandwidth and I/O delay.

This paper attempts to explain whether Linux scheduling algorithm performs consistently in a virtual machine and a real machine, also the influence of two-level scheduling system on the I/O performance of different application loads by taking open source Xen as an example. To facilitate this discussion, this study only configures one virtual domain thus to ignore the impact that the Credit Scheduler in Xen Hypervisor throws on test results.

This paper is organized as follows: In part II, the authors introduce the Xen architecture, analyze the Xen PV I/O model, provide the typical disk read I/O process in this model, summarize the four I/O scheduling algorithms in Linux kernel and the credit scheduling algorithm in Xen, and then raise some issues and assumptions concerning the two-level scheduling system introduced by virtualization; In part III, Introduce the test environment, test tools and test methods; In part IV, Analyze the test results and discuss; In part V, summarize the progress of existing researches on virtual I/O scheduling; In part VI, draw a conclusion of this research and make suggestions for the future work.

2. Xen Overview

Xen, an open-source hypervisor with bare-metal architecture, supports multiple operating system instances run on a single physical machine. These instances can adopt full virtualization or paravirtualization. Xen full virtualization uses QEMU simulation, requires no modification to the guest operating system and has good compatibility to support various versions of operating systems; the paravirtualization modifies the guest operating system code, adopts the driver separation model and reduces the number of processor context switches, which improves its performance effectively but restricts the selection of guest operating systems. Xen3.x introduces two hardware-assisted virtualization technologies, Intel® VT and AMD-V, which make up for the defects of the traditional x86 architecture in supporting virtualization and enable the running of various versions of operating systems on Xen, including closed source operating system like Windows. Clients like this are called the HVM guest. It is noteworthy that PV front-end device drivers can be installed to the HVM guest operating system to improve performance. This research adopts this method to build the test environment.

2.1. Xen PV I/O Model

Figure 1 shows the Xen PV I/O model. The virtual machines running on the Xen Hypervisor can be divided into three categories: Dom0 (administrative domain), Dom1 (driver domain) and DomU (user domain). Dom0, serving as the extension of the Hypervisor, provides management services to other domains, including creation, deletion, save and recovery, and has direct access to the underlying hardware with the hardware drivers installed; Dom1 owns some physical devices and allows other guests to access the

device driver. DomU runs the front-end device drivers (virtual devices) and can access the physical devices by applying for services to the back-end device driver.

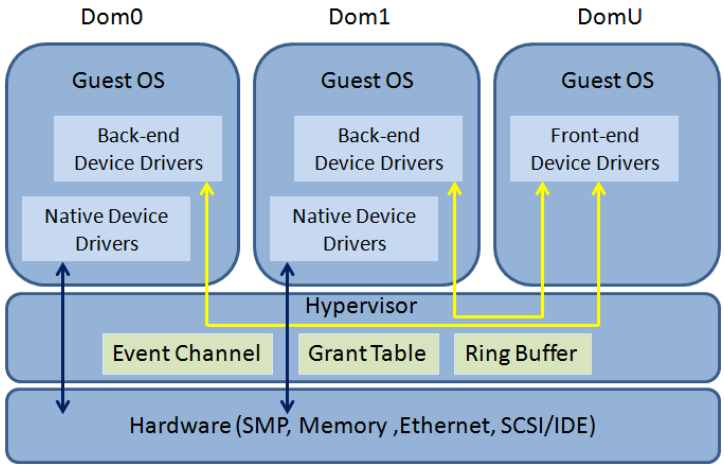

Fig. 1. Xen PV I/O model.

In the design of Xen paravirtualization structure, the hypervisor should be as lightweight and simple as possible. It interacts with the guest operating system through limited Hypercall interfaces to complete the following operations: creating and managing the virtual environments, scheduling virtual processors, managing physical resources such as CPU, memory and interrupts, and communication between different virtual domains. The virtualization and management of I/O devices are completed jointly by the Hypervisor and Dom0.

In the Xen PV I/O model, communications between different virtual domains can be achieved by the event channel, ring buffer, grant table and Xenstore mechanism provided by the Hypervisor.

The event channel is a major asynchronous communication mechanism of Xen and can be used in conjunction with the ring buffer to achieve effective message passing between different virtual domains and virtual domain to hypervisor. Xen events can be divided into four broad categories, namely, physical IRQ, virtual IRQs (VIRQ), inter-domain events and intra-domain events (inter-virtual processor events), among which inter-domain events are generated by the paravirtualized device to achieve communication between the front-end and back-end driver in the driver separation model. The event mechanism is closely related to the virtual machine scheduling. In essence, the scheduling of the virtual processor in Xen is performed by the scheduler in the Hypervisor. Suppose that the two virtual processors in the producer/consumer model are bundled in one physical server, in order to ensure the normal communication of intra-domain events, one end shall initiate the "yield" operation in time by Hypercall to inform the Hypervisor of releasing the occupation of the physical CPU.

The ring buffer, one of the main usages of the shared memory page, is a simple data structure containing requests and responses used for communication between PV front-end and back-end device drivers, i.e., one end is used to place requests and the other end used to obtain requests, processes it and return the processing results. For performance reasons, the real data is stored separately and the ring buffer only stores the description of requests and responses with fixed-size of data structures. In the scenario of transferring large amounts of I/O data, the two ends can adopt the polling approach to access. In case of infrequent communication, the event mechanism can be adopted to notify the arrival of new data.

The grant table, a page access-grant mechanism, is itself a structured data and describes the authorized domain identifier, pointed page frame and permission information. Entries in the table are identified by the grant reference. In the Xen PV structure, when transferring large amounts of data, block devices and network equipment adopt the ring buffer to place requests and responses, together with the grant reference pointing to specified page containing write data. The recipient can perform the read/write operations on the specified page after being authorized.

Xenstore, a simple hierarchical storage system maintained by Dom0, stores the configuration information of the guest operating system and provides a well defined user interface. The access to Xenstore is completed through the shared memory, ring buffer and event channel. A main function of

Xenstore is to provide device information. In the Xen PV I/O model, the front-end device driver gets access to device information available by querying Xenstore at the initialization stage.

In summary, the collaboration between the front-end driver and the back-end driver, along with the event channel, ring buffer, grant table and Xenstore mechanism provided by the hypervisor, achieves efficient disk read operations. The process of I/O read is as follows:

(1) The front-end driver allocates the shared memory page, initializes the ring buffer, shares this memory page through the grant table, writes the grant reference to Xenstore and creates an event channel;

(2) The back-end driver maps the shared memory page to its own address space according to the grant reference and connects to the event channel;

(3) When processing the request, the front-end driver first allocates the shared buffer to store data blocks to be read and then provides them to the back-end driver through the grant table;

(4) The front-end driver writes the request to the ring buffer and notifies the back-end driver through the event channel;

(5) The back-end driver is awakened to read and process the request, complete the I/O operation by real device drivers and store the data blocks that have been read in the shared buffer;

(6) The back-end driver releases the mapping to place a response in the ring buffer and notifies the front-end driver through the event channel;

(7) The front-end driver is awakened and the results have been stored in the shared buffer to complete a full disk read I/O request-response process.

2.2. Linux I/O Scheduling Algorithm

There are four I/O scheduling algorithms in Linux2.6, i.e., noop, deadline, anticipatory and CFQ. The details are as follows:

(1) Noop

Noop, short for No Operation, is the simplest scheduling algorithm and adopts the "first in and first out" scheduling strategy. It does not reorder the I/O requests, but merges adjacent requests to shorten the seek time and improve the efficiency of disk access. In general, noop is a very good choice for intelligent devices or devices without the track seeking mechanism, such as flash drives.

(2) CFQ

CFQ, short for completely fair queuing, is an algorithm designing multiple internal I/O queues. The first queue uses the round-robin algorithm to take out the requests of non-empty queues in succession and then put them into a dispatch queue, in which the principle of "reducing the seek time", rather than "first in and first out", is adopted to reorder and respond to the requests. Therefore, some I/O requests may starve to death in extreme cases. CFQ algorithm somehow ensures the fair distribution of I/O bandwidth, however, in actual cases, when I/O requests of multiple processes are mapped to a same queue, the fairness is restricted. Since Linux kernel-2.6.18, CFQ algorithm is the default scheduling algorithm. CFQ algorithm is relatively suitable for discrete reading of I/O applications, such as OLTP, and performs well in multimedia applications.

(3) Deadline

Deadline algorithm is designed to meet two objectives - to reduce the seek time and to ensure that requests are met within a certain time period. For the former, the algorithm introduces the red-black tree and linked list mechanisms to analyze and reorder the requests so as to reduce the latency; for the latter, the algorithm draws support from the timer mechanism in Linux kernel. In implementation, the algorithm designs three queues, i.e., request queue sorted by the sector number, the read queue and write queue by overtime. The priority of three queues is read queue> write queue> request queue. In case that the request in the read and write queue does not time out, the scheduler will batch process the request queue. Besides, the algorithm takes the starvation problems that the read operation causes to the write operation into account. If the number of times of write hunger exceeds the predetermined threshold, the scheduler will directly address a write request. Deadline algorithm is relatively suitable for database applications.

(4) Anticipatory

Anticipatory algorithm assumes that read requests of the process are not completely at random, instead, two requests are related to a certain degree. When a read request is submitted by an application to the kernel for processing, this algorithm assumes that the next relevant read request will arrive within a certain time interval, which is consistent with the situation when the disk receives read requests while busy with write operations and waits for a certain time interval to anticipate the arrival of the next read, so as to avoid

the movement of the magnetic head. Based on this assumption, the anticipatory algorithm expands based on the deadline algorithm and sets 6s waiting time for each read request. It is easy to understand that anticipatory algorithm performs quite well in addressing the read I/O requests in a consecutive order, such as Web server, but performs poorly in a large number of random read I/O applications, such as database. This algorithm has been removed from Linux kernel-2.6.33 and later versions.

The selection of I/O scheduling algorithm depends on the features of the underlying hardware and the application of the upper level and needs to strike a balance between the I/O throughput and the response time.

2.3. Xen Credit Scheduler

Credit scheduling algorithm is the default scheduling algorithm used since Xen3.0. It is designed for equitable distribution of processor resources on Hypervisor domains, by setting a weight and cap value to control the proportion of CPU resources allocated to the client operating system as well as resource limit. If cap is overlooked, the domains with the same weight will receive the same amount of CPU resources. By setting a cap, physical CPU time slice allocation can be controlled. For example, if cap=50, the domain can only use half of all CPU time slice resources.

Credit algorithm designs two queues, respectively referred to as Under and Over queues. Initially all domains will be placed in Under queue. Credit is set as weight, and credit algorithm will schedule in accordance with FIFO principle. Each scheduled virtual domain with sufficient credit will receive up to 30ms time slice. Each 10ms will trigger a scheduler interruption, and running domains will be deducted 100 credits. When a virtual domain's credit is negative, it will be placed in Over queue. When all domains' total credits are negative, credits will be reallocated for them. Under queue will always be scheduled before Over queue. Only when the Under queue is idle, it is allowed to schedule domains in Over queue, i.e., only when CPU is idle, it is allowed to provide extra time slice resources to the domain.

The main advantage of Credit Algorithm is its ability to manage the physical CPU resources globally and to allocate time slice fairly and efficiently to each domain. But on the other hand, this algorithm is not real-time, and cannot guarantee fairness of the response delay. It does not distinguish between domains that are computing-intensive and I/O-intensive. If a domain consumes CPU resources for a long time, it will ignore I/O events waiting in queue. In fact, the domain's location in Under queue is essential for its scheduling in a timely manner, and affects I/O response time. To solve this problem, credit algorithm adds a new state, i.e., boost state, of which the domain will be given priority above domains in Under and Over queues in scheduling. If idle domains receive events through the event channel and enter boost state, rather than into the end of run queue, the scheduler will replace currently running domains in order to execute them, so as to reduce I/O response delay. But this solution is limited. If multiple I/O intensive domains are running, and enter boost state at the same time, it is difficult to reduce response delay as expected.

2.4. Problems with Two-Tier Scheduling System

The design and selection of traditional I/O scheduling is heavily impacted by the seek time and latency of the underlying disks and application I/O characteristics, with the aim to maximize application I/O throughput, and minimize the application I/O response delay.

In the virtual world, in principle, I/O requests in virtual domains are processed by the virtual machine's operating system scheduler before submitted to the hypervisor. I/O requests are merged and reordered. In xen PV mode, disk sector information is also passed to the back-end driver, which completes read and write operations according to the disk information after reading data from the shared buffer.

In this process, the front-end driver does not know the access characteristics of underlying hardware, i.e., its perception of delay performance of underlying hardware is obtained through collaboration with back-end driver. And accurate perception of the characteristics has certain reference value for selecting suitable I/O scheduling algorithms in virtual domains. Due to the introduction of hypervisor to establish a two-tier scheduling system, i.e., the scheduler in virtual machine and the other one in dom0, combined with xen credit scheduler for virtual domains, the application of I/O process becomes not only even more complicated, but also causes some new problems. Firstly, due to the introduction of virtualization, I/O performance is reduced, with the necessity to analyze which applications are loaded and which scheduling algorithm is most affected. Secondly, the introduced two-tier scheduling system needs to be analyzed:

which scheduling algorithm is most helpful to improve I/O performance, how to select suitable scheduling algorithms for different kinds of application loads in virtual domains.

3. Test Environment

The testing tool is FFSB (Flexible File System Benchmark), which is a multi-threaded file system-based performance evaluation tool. Distinguished from other tools, it can customize configuration on application load through configuring profile file, providing configuration regarding read and write, I/O block size, number of concurrent threads and so on. It can well meet test requirements.

In order to simplify the experimental operation and make test results more convincing, only one virtual domain is created in this experiment, neglecting the impact of the Xen credit scheduler. All tests run in the same hardware environment, i.e., an HP DL380G7 server configured with two Intel ® Xeon ® 2.4GHz CPU, 128G memory, local 450G SAS hard disk. The linux kernel version for both Dom0 and virtual domain is 2.6.32. Open source Xen version is 4.1.2. In order to eliminate the impact of memory caching, mechanisms such as directIO are enabled in the test.

When comparing I/O performance of real machine and virtual machine, FFSB is set as single-threaded. Different scheduling algorithms in virtual domains are selected so as to test I/O performance loss under load characteristics such as random read, random write, sequential read, sequential write and mixed random read and write;

In two-tier scheduling system test, FFSB is set as 8 threads. Scheduling algorithms in dom0 and virtual domain are set respectively to test I/O performance under load characteristics such as random read, random write, sequential read, sequential write and mixed random read and write.

From the test, the impact on I/O scheduling algorithm execution with the introduction of virtualization can be seen, thus providing recommendation for deployment of application in virtual environment.

4. Test Results and Discussion

4.1. Comparison of Real Machine and Virtual Machine

The introduction of virtualization has some impact on application I/O performance, as can be seen from Figure 2. The impact is mainly related with the characteristics of application loads, and is relatively less related to what scheduling algorithm is selected in virtual domains. For random read and random write, performance loss is about 30%. For sequential read and write, performance loss is between 50% and 55%. For mixed random read and write, I/O performance impact is the greatest, the loss being at more than 55%, nearly reaching 60%.

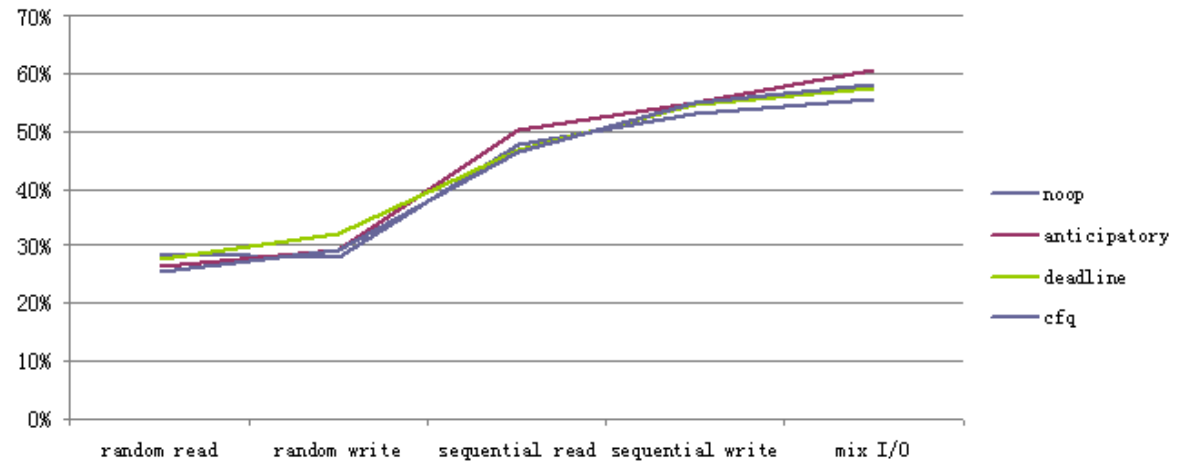

Fig. 2. Comparison results of real machine and virtual machine.

4.2. The Impact of Two-Tier Scheduling System

This project selects scheduling algorithm combinations in Dom0 and virtual domains for five typical application I/O characteristics, i.e., random read, random write, sequential read, sequential write and mixed random read and write. Figure 3 shows the test results, in which the horizontal axis represents the scheduling algorithm in Dom0. For random read, random write and mixed random read and write, the vertical axis represents TPS value. For sequential read and sequential write, the vertical axis represents the I/O throughput. The different columnar colors represent different scheduling algorithms in the virtual domain.

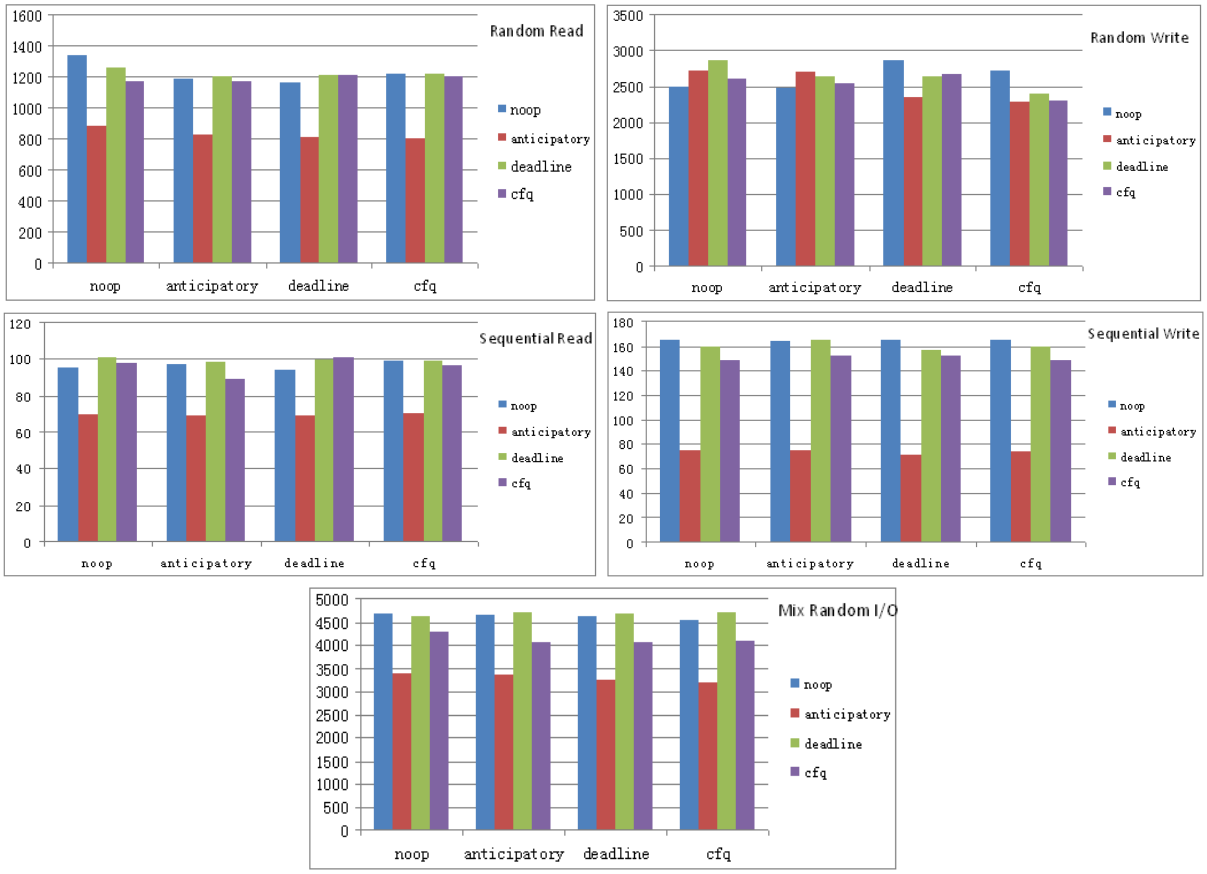

Fig. 3. Open source Xen two-tier scheduling test results.

As it can be seen from the results, changing scheduling algorithm in Dom0 does not affect overall performance of application I/O, while scheduling in the virtual domain will have a greater impact. Application I/O throughput and TPS performance mainly depends on the scheduling algorithm selected in virtual domain.

For random read, sequential read, sequential write and mixed random read and write, both noop and deadline have the better performance, followed by the cfq. Anticipatory behaves the worst.

For random write, no significant differences showed regarding to all the scheduling algorithms.

5. Related Research

Disk I/O is critical for performance of virtual applications. If not handled properly, it can easily become the bottleneck of the whole system. Therefore, in recent years, the research on virtual resource scheduling, in particular disk I/O scheduling performance has received widespread attention within the industry. Boutcher *et. al.* [3] studied and maximized VM throughput, and ensured the VM fairness with combination of virtual domain and host I/O scheduler. One of the conclusions drawn is that the choice of virtual domain scheduler should be based on the type of application load. Mukil and other people's [5]

experimental data show that in virtual environment, the delay performance of the VM virtual disk is very different from that of the physical hard disk. When adjusting the VM's I/O performance, it is necessary to consider VM, VMM and hardware factors. His experiments adjusted anticipatory and cfq scheduling algorithms in VM, and figured out optimization requirements of scheduling in VMM. The VMM proposed in the first two solutions refers to Dom0 in Xen architecture. However, when running multiple virtual domains, the impact of Xen credit Scheduler's VM scheduling policy on VM I/O performance cannot be ignored. Diego et.al [4] studied the relationship between hypervisor scheduling and I/O performance. They selected a variety of computing-intensive, bandwidth-intensive and delay-sensitive combinations to run simultaneously in multiple virtual domains. After adjusting 11 scheduling parameters in VMM, they observed the changes in bandwidth and response time, and came to a conclusion that Xen Credit scheduler has a major impact on application I/O performance.

6. Conclusions and Future Work

In this paper, we showed the performance of Linux I/O scheduling algorithms in virtual environment, as well as in the case of a single virtual domain under Xen architecture, the impact of Dom0 and virtual domain scheduling algorithms on I/O performance of different application loads. The main conclusions we have come to under this test environment are:

1) In a virtual environment, the performance trend of Linux I/O scheduling algorithm is similar with that of real machine, but there is performance loss, depending on the type of running application load in virtual domain;

2) The impact of scheduling algorithm in Dom0 on application I/O performance is weak. In virtual domain, choice of scheduling algorithm depends on I/O characteristics of running application load, which is consistent with the observation of Boutcher;

3) Under open source Xen environment, for all types of application loads, noop and deadline algorithms are basically good choices.

Virtualization technology is evolving and new technologies are emerging such as SR-IOV VMM-bypass. By skipping the virtualization layer processing and directly mapping hardware to virtual machine, SR-IOV greatly reduces I/O delay. These technologies play a huge role in enhancing virtual domain application I/O performance. At the same time, there are ongoing improvements in virtualized I/O scheduling algorithm. For example, in Xen4.2, the Credit Scheduler introduces new parameters to improve the vm response. After this research work, we will continue to pay attention to the development of these new technologies, and carry out research into I/O performance of multiple virtual domains. We plan to design an optimized I/O scheduling framework according to different storage environments and application load types. We will also test and validate them in order to better meet virtual I/O performance requirements.

References

[1] D. Heger and R. Quinn, "Linux 2.6 IO performance analysis, quantification, and optimization," in *Proceeding of the International Conference for Performance and Capacity Management - CMG2010*, Orlando, FL.

[2] D. Heger and S. Pratt, "Workload Dependent Performance Evaluation of the Linux 2.6 I/O Schedulers", in *Proceeding of the 2004 Linux Symposium (OLS)*, Ottawa, 2004.

[3] D. Boutcher and A. Chandra, "Does virtualization make disk scheduling passe'?," in *Proceedings of the Workshop on Hot Topics in Storage and File Systems (HotStorage '09)*, October 2009.

[4] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling I/O in virtual machine monitors," in *VEE'08: Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, New York, NY: ACM, pp. 1-10, 2008.

[5] M. Kesavan, A. Gavrilovska, and K. Schwan, "On disk I/O scheduling in virtual machines," in *Proceedings of the 2nd Conference on I/O Virtualization*, 2010.

[6] C. Takemura and L. S. Crawford, *The Book of Xen: A Practical Guide for the System Administrator*, No Starch Press, 2009.

[7] D. Chisnall, *The Definitive Guide to the Xen Hypervisor*, Prentice Hall, 2007.

[8] R. Love, "Kernel Korner: I/O schedulers," *Linux J.*, vol. 118, p. 10, 2004.

[9] S. Seelam, R. Romero, and P. Teller, "Enhancements to Linux I/O scheduling," in *Proceedings of the Linux Symposium*, vol. 2, pp. 175-192, Ottawa Linux Symposium, July 2005.