VITS Test Suit: A Micro-Benchmark for Evaluating Performance Isolation for Virtualization Systems

Pingpeng Yuan, Chong Ding, Long Cheng, Shengli Li, Hai Jin
Service Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan 430074, China
hjin@hust.edu.cn

Abstract

Virtualization technology improves the resource utilization, but also raises the probability of resource contention. Thus, it arises one problem, namely how about performance isolation among VMs. To answer the problem, users highly require benchmarks which can evaluate performance isolation. However, few benchmarks which can give solutions to the problem are available. Especially, as we know, there does not exist a micro-benchmark which can measure how the performance of individual primitive operations varies with the type of "misbehavior".

In the paper, we present VITS - a micro-benchmark to evaluate the performance isolation of virtualization system. VITS Test Suite can test both software and hardware performance isolation. It is mainly consisted of six interference programs, which respectively test the performance isolation of cache, memory bandwidth, memory space, CPU, network and disk. VITS Test Suite provides a micro-result for user to analysis the performance isolation problem of the virtualization system along with the underlying hardware. Besides, we test Xen using VITS test suit. The experimental results show that, the cache, memory and disk performance isolation still have some big weak points to improve, in other words, the current virtualization system Xen is still unfair in fine-grained resource share.

1. Introduction

Since more powerful hardware technologies are available, a hardware platform can host more virtual machines. Thus, the hardware resources and software resources of the host are shared by virtual machines, no more belong to a single machine. This improves the resource utilization of host, but also raises the probability of performance influence. For example, the resource consumption of one virtual machine may impact the promised resource share of other VMs on the same physical host. Thus, running one virtual machine affects the performance of another. However, successful virtualization technology should isolate a virtual machines from one another [2].

Although many research paid attention isolation, for instance, D. Gupta et al proposed two mechanisms – SEDF-DC and ShareGuard to improve CPU and network resource isolation in Xen [18], virtualization system like Xen or VMware provides partly isolation, but there is a difference between them.

Then some questions arise. Firstly, how well do virtualization systems isolate virtual machines from each other? In the practical applications, the capability to isolate the crucial virtual machine from misbehavior virtual machine is an important factor for choosing virtualization production. Secondly, how to measure some operations so as to decide what are the reasons to damage performance isolation? To decide the reasons help optimize performance isolation of virtualization systems. Although those questions are very important and should be considered, it received less attention. In this paper, we provide VITS-test Suite (Performance Isolation Test Suite for Virtualization System) to answer those questions.

The rest of this paper is structured as follows: Section 2 outlines related research in the field of performance evaluation on virtualization system. Section 3 gives overview of VITS Test Suite. Section 4 introduces the design of VITS. In section 5 we present the results of experiments on Xen. Section 6 concludes the paper by briefly summarizing the major contribution of the research.

2. Related Work

Over the past decade, the use of virtualization technology has grown rapidly. This has spurred great advances in virtualization technology: a large number of hypervisors for different purposes exists. However, virtualization performance evaluation has received less attention. Many research generally used standard operating system benchmarking suites to evaluate the impact of virtualization on performance. However, operating system benchmarking suites are not aware of virtualization.

In the recent years, some approaches have been developed to evaluate or collect the performance of virtual machines [3, 4, 5, 6, 13, 15, 16]. All the work including standard OS benchmarks can be classified as micro-benchmark, macro-benchmark, profiling etc.

One of the typical operating system micro-benchmark is *lmbench* suite [8, 14]. *lmbench* is intended to give system developers insight into basic costs of key operations [8]. The design goal of *lmbench* was to develop a simple, portable, and realistic benchmark suite that accurately measures a wide variety of individual operations. Once the benchmark was used in virtual machine testing, so many tests it contained would inundate the key performance of virtual machine that we concerned. Although most of its algorithms in this benchmark suite were carefully designed to ensure the accuracy of results [8, 9], the results consist of too much content that they were time-consuming and occupied a considerable space of result files, which made results processing more complex. *lmbench* was lack of statistical rigor and their measurements were inconsistent [19]. Moreover, because of its non-parallelism, extreme cases in performance of virtualizations can hardly be measured. Thus, the benchmark suite is not really suitable for evaluating performance isolation.

Macro-benchmarks evaluate overall performance of actual specific applications under loosely defined conditions. Therefore, the behavior of other applications can only vaguely be inferred according to the results of macro-benchmarks. As mentioned in section 1, typical macro-benchmarks for virtual machine environments are *vConsolidate* [5], *VMmark* [6] which measure overall performance of server consolidation and were not designed for performance isolation evaluation.

Considering it is necessary for both accurate hypervisor characterization and decomposition analysis of virtualization's performance impact, Kim-Thomas Möller [10] introduced *VMbench* which focused on low-level cross-hypervisor comparison and prediction of virtual machine performance. *VMbench* uses a three-stage approach to characterize performance of a virtual machine environment. In the first stage, *VMbench*'s micro-benchmark which is similar to those from the *lmbench* suite was used to determine the best-case performance of a virtual machine's primitive operations for a given combination of hardware, hypervisor, operating system and workload. The second stage combines the outcome of the first stage using a linear model to predict best-case results for realistic applications. The third stage examines virtual machine interference. However, *VMbench* accounts time using the time stamp counter, and it leaves all statistical analysis to dedicated statistics software, which applies the linear model.

In addition to traditional micro-benchmark and macro-benchmark, there are also some profiling or monitoring tools such as *XenMon* [3], *Xenoprof* [4] available to monitor the performance of Xen-based virtual machines. *XenMon* is performance monitoring tool for the Xen-based virtual environment [3]. *XenMon* supports several metrics, such as different time measures (CPU usage, blocked time, waiting time), execution counts, and I/O operation counts. It presents a performance case study that demonstrates and explains how different metrics reported by *XenMon* can be used in gaining insight into an application's performance and its resource usage/requirements, especially in the case of I/O intensive applications. *Xenoprof* [4] is a system-wide statistical profiling toolkit implemented for the Xen virtual machine environment. The toolkit enables coordinated profiling of multiple virtual machines in a system to obtain the distribution of hardware events such as clock cycles, cache and TLB misses. *Xenoprof* required virtualizing the hardware performance counter, however, hardware performance counter virtualization was costly. Moreover, *Xenoprof* is tied to paravirtualized *XenoLinux*.

Although there exist some benchmarks for virtualization technology, few benchmarks for virtualization performance isolation are available. Considering it is an important issue of isolation from misbehaving VMs, J. N. Matthews etc. developed a macro-benchmark for evaluating performance isolation [11]. The test suite included six different stress tests - a CPU intensive test, a memory intensive test, a disk intensive test, two network intensive tests (send and receive) and a fork bomb, and used response time reported by the SPECWeb [7] as the performance metric. But the test suite firstly is a macro-benchmark. And it cannot evaluate isolation in the cache, memory bandwidth and disk with different size requests.

3. VITS Test Suite Overview

To evaluate the performance isolation of a virtualization system, VITS test suite includes several different tests – cache test, memory access and memory bandwidth test, disk test, network test which are achieved by interference program set.

VITS is consisted of four parts: User interface, monitor, the programs in VM side and interference program set. VITS provides a web-based user interface, thus users can remotely start, control and terminate the test. Similar as user interface, the monitor can also be in remote machine. The monitor is responsible for the testing process control, including new test deployments, test process control, test results collection etc. The monitor also includes a component – parser which is the utility to parse the log files from target virtual machine so as to ease the generation of result diagrams.

The programs in VM side run in target virtual machine. The target virtual machine runs virtualization environment which a certain virtual machine monitor like Xen creates on a physical machine. The virtual machine monitor provides interface to manipulate the guest virtual machines. Guest virtual machines execute I/O operations through the assistance of hypervisor. The programs of VITS running in the host machine are used to deploy programs to guest virtual machines, to start and stop guest virtual machines, and collect test results.

Interference program set are consisted of six kinds of programs which run in interference virtual machines and test CPU, cache, memory (bandwidth, utilization), disk and network respectively. The interference programs for CPU are mainly to evaluate the impact on target VM when other VMs run computing intensive tasks. Generally, cache is shared by multiple VMs, cache interference programs are included since cache miss influences the performance of computer. Memory is another important resource shared by virtual machines. When running or executing I/O, virtual machines highly requires memory and access memory frequently which both compete for memory bandwidth and memory space. Thus, the isolation tests on memory are consisted of two parts: bandwidth and memory consumption. Moreover, the interference programs for disk of VITS Test Suite are used to evaluate the interference when VMs read/write disk. Network interference programs are designed for evaluating performance isolation of network.

4 Design and Implementation

In this section, we discuss the detailed design of VITS-test suite.

4.1 Guarantee accuracy

VITS test suite adopts several methods to guarantee accuracy of test results. For example, every program requires memory to run them. If there are not enough memory available, then test suite will trap to operating system page fault frequently. Operating system decides whether the page is not in memory. If it is true, operating system gets empty frames and swap page into frame. Then operating system resets tables and sets validation bit. Finally, operating system restarts the instruction that caused the page fault. Since memory access time is consisted of page fault overhead, swap page out, swap page in, restart overhead etc. when page fault occurs, the memory access time in this case is larger than a general memory access time. Thus, it will induce worse performance in disk or memory.

For the purpose of acquiring accurate test results, VITS Test Suite will check current memory size of the virtual machines before VITS test suite runs. If current memory size of virtual machine is enough to run interference programs, then VITS test suite will continue run. Otherwise, VITS test suite will wait until enough memory is available.

To determine whether the memory is enough, according to the above fact that the memory access time when page fault occurs is larger than a general memory access time, VITS accesses every page and compute the access time. When access time becomes sharply bigger, then it can conclude that we reach the memory bound allocated to the process.

4.2 Cache

When the processor needs to read or write data in main memory, it first checks whether that data is in the cache. If the processor finds that the data is in the cache, namely a cache hit has occurred; otherwise, it is a cache miss. In the case of a cache miss, they require the data to be transferred from main memory. In order to make room for the new entry on a cache miss, the cache generally has to evict one of the existing entries.

Cache interference program aims to evaluate the cache performance isolation of virtual machines. According to cache replacement algorithms, we allocate a memory space which is a little bigger than cache capacity, and construct a circular linked list in the memory as shown figure 1. The distance of every two elements in the list is a cache line size. We access the circular linked list from the head, when arrival the tail, we begin from the head again. Because the difference between previous memory access location and the next memory access location is a cache line, thus, new data will be loaded into cache line. Through the above approach, we can achieve better interference on cache.

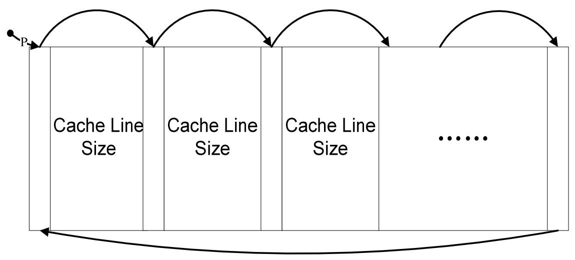

Figure 1 Circular linked list in sequence space

4.3 Interference programs for memory

Virtual memory management is a significant issue for most virtualization systems. Although techniques for optimizing CPU and I/O device utilization are widely available, there are few optimization techniques in use since it is much more difficult for virtualizing time-sharing of physical memory.

Currently, virtualization systems are responsible for managing the allocation of physical memory to domains, and ensuring safe use of the paging and segmentation hardware. The page tables are marked as read-only by the hypervisor. The hypervisor requires guest operating systems to explicitly request any modifications. Any attempt to write to them triggers a sequence of actions.

Currently, Xen does not swap pages allocated to a domain out to disk. To permit a guest domain to expand its memory usage, Xen provides a balloon driver. By using the balloon driver, a guest domain can either release or request more memory. If a guest domain requests more memory than really needed, a well-behaved guest domain should free some buffers or swap data out to the disk and return unused blocks of memory to the hypervisor. However, not all virtual machines behave well, they may constantly require additional memory and compete for memory bandwidth.

Interference program for memory consumption aims to simulate those badly VMs, and find the reasons to badly memory isolation. Interference programs for memory are consisted of two parts: one for memory consumption and the other for memory bandwidth competition. The goal of interference program for memory is to simulate misbehaviors of virtual machines. Thus they need try to require more memory and access memory frequently. In operating systems, statistics show that there are three operations which require the data block copy: read or write data from files; transmit or receive data from network; remap data to memory. Especially in web server, the time required by block data copy uses is more than the fifty-five percent of processing time in kernel [20]. We analyze the three operations and function *bcopy*(), and find that the basic memory operation is essentially memory assignment as shown in figure 2.

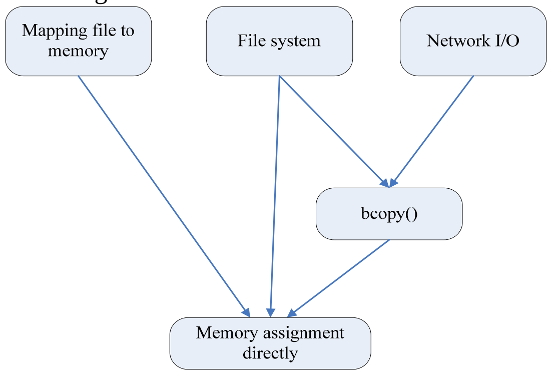

Figure 2 Relationship between three operations and memory assignment

So memory interference program of VITS adopts direct memory byte copy as the basic operation. To achieve this, firstly, two memory blocks are allocated and two register variables which store the beginning address of two memory blocks are defined. The reason to store the address into registers is to reduce interferences from VITS as few as possible. Then VITS repeatedly copy data from one memory block to another block as shown in figure 3.

Interference programs for memory consumption invokes *malloc*() constantly, but does not initiate the memory so as to reduce impact from write operation.

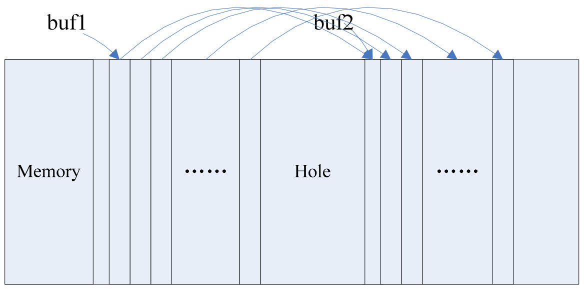

Figure 3 Interference program for evaluating performance isolation of memory

4.4 Interference programs for Disk

Virtual machines need to support the virtual block device driver, typically a virtual hard disk, which presents an interface to an abstract block device. For example, the Xen virtual block device uses the I/O ring mechanism and supports three operations: reading and writing blocks, write barrier. The third operation is not supported by all back ends.

Like real block devices, the virtual block device also supports command-reordering. In other words, commands issued to it may not complete according to FIFS (first in first service). Since VMs hosted on a physical machine may access the hard disk at the same time, reordering their commands issued to the disk may achieve good throughput in a virtualized environment.

When the disk responds the commands, data transfer is started. Generally, data size in each transfer is at least several KB. The best way to transfer those data is DMA transfers, not host-based copying. To facilitate this, the guest VM grants back end of the driver to access the destination page by grant table. Thus, the destination page can be used directly for DMA transfer.

In disk performance isolation testing, we use an interference program which can produce variable request size. The previous benchmarks did not consider the variable size of application's disk request. But when many virtual machines are hosted on one physical machine and disk request size of every virtual machine is different. In this case, the hypervisor's disk performance isolation is very important for crucial service. Thus, we introduce variable request size of disk into the design interference program of disk as shown in figure 4.

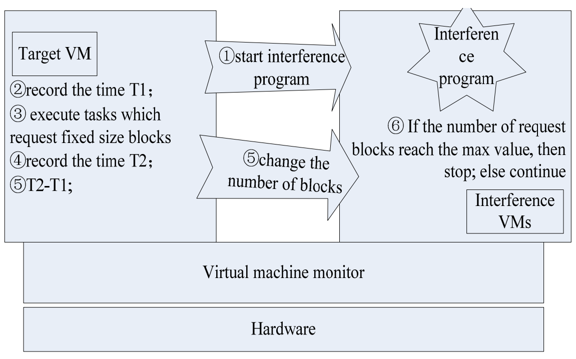
Target VM
②record the time T1；
③ execute tasks which request fixed size blocks
④record the time T2；
⑤T2-T1;
①start interference program
⑤change the number of blocks
Interferen ce program
⑥ If the number of request blocks reach the max value, then stop; else continue
Interference VMs
Virtual machine monitor
Hardware

Figure 4 Interference program for disk

4.5 Network I/O test

The virtual machines need to communicate with each other and the outside machines. Generally, virtual machine monitors, for example Xen provides several different networking options: Bridged networking, Routed networking, and VLAN with NAT. Bridging is the default option for Xen networking. In this case, Xen creates the virtual bridge which connects all the virtual machines to the outside world through virtual network interfaces.

Networking functionality of guest domain is implemented by the virtual interface driver which uses two I/O rings, one for transmission and one for receiving packets. Each data transmission request contains a grant reference and an offset within the granted page. Receiving packets is similar with data transmission: The guest domain puts a receive request into the ring indicating where to store a packet.

Network I/O processing in virtualization environment primarily involves two components: the real device driver and the back-end (virtual) device driver. When the hardware receives the packet, Xen trapped by the interrupt sends a virtual interrupt to the corresponding driver domain. The driver domain invokes the interrupt handler which hands the packet over to the back-end driver and grants the memory page containing the packet to the target guest domain. Then, back-end driver forwards the packet to the front-end driver. The front-end driver in the target guest domain will deliver the packet to higher layers of the networking stack. The transmission process is similar as receiving.

One approach for accurate evaluation is to instrument these components and collect trace on the I/O path. In network I/O test, trace programs are designed. Through the trace program, VITS can record the time that every network packet goes through the physical NIC, the back-end and the front-end driver. The trace programs include a multithreading packet capture program and data exchange programs that help ease communication between Dom0 and DomU. The packet capture program captures the packet through the physical NIC, the backend driver, frontend driver simultaneously. Thus, some information about packets, for example timestamp can be recorded.

There is still another problem. The network packet can be lost during the transmission between Dom0 to DomU. Thus, the trace collected at different components in I/O path may be inconsistent. So, we must align traces so as to acquire correct performance data. Since packets are transmitted between Dom0 to DomU through page remapping, thus, the order of the packets will not be changed dramatically. Moreover, VITS identifies each packet using a tag. As for the TCP packet, the tag is the TCP sequence number. It is complicated for UDP packet since there doesn't exist a sequence number in UDP packets. Thus, we write a unique tag in the first byte of every UDP payload. If a packet received in next component of I/O path is not the expected one, it is for sure that some packets must be lost during the transmission. If this packet tag is the same with the packet captured, a matched pair are found.

Since different applications hosted on multiple virtual machines may send different size packets, for evaluating the network isolation, interference programs send packets with different size.

5. Experimental Results

In this section, we use VITS test suite to test the performance isolation characteristics of Xen on Xeon server.

5.1 Experiment setup

All the following experiments are performed on a server with two CPUs. Each CPU has 4 cores of 1.6GHz. The memory of the server is 4GB. The model of the disk is ST3160815A5 and its size is 160GB, Interface type is SATA. We choose virtual machine monitor Xen 3.2.1 for our tests respectively. The host operating systems are Debian 5 with the linux kernel 2.6.26 and guest operating systems are Debian 4 with the kernel 2.6.18. Every guest OS has 2 VCPUs and 256M memory space as default, as table 1 shows.

Before using VITS to test, we need to establish some baseline data. We run vBench in one of the virtual machines and record the results as a baseline data. Then interference programs of VITS are run in the other virtual machines to quantify the degree of performance isolation.

Table 1: The configuration of the server

Hardware	CPU	Dual-CPU, each has 4 cores, 1.6GHz
	Memory	4GB
	Disk	160GB
Software	Linux	Debian 5 Kernel 2.6.26
	Xen	Xen 3.2.1
	Guest OS	Debian 4 Kernel 2.6.18

5.2 Cache interference evaluation

Figure 5 shows the results in the normal case (not running interference programs) and interference case in which interference programs run in other virtual machine. According to figure 5, the average cache latency of target virtual machine is 8.818*ns* in normal case. However, when we run interference programs for cache in other virtual machine, the latency time of cache in target virtual machine reaches 8.859*ns*.

The reason is that current virtualization technology does not provide isolation solution in cache. If the data which application needs exceed the capacity of cache or access violation, the cache miss will occur. Cache misses will lead to larger average latency of cache. The replacement strategy doesn't know the replaced cache line belong to which virtual machine, so it is very likely to clean out a virtual machine's cache line for another virtual machine. When the first virtual machine is scheduled back, the data in cache have been changed. This influences the performance of virtual machine.

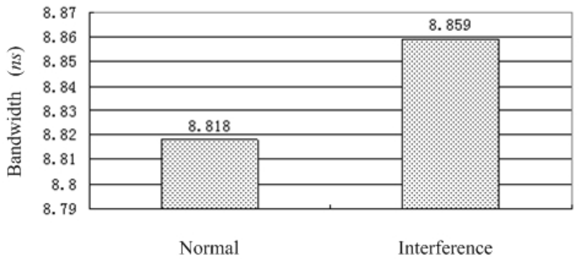

Figure 5 Cache performance isolation results

5.3 Memory interference evaluation

Figure 6, 7 show memory access latency and communication bandwidth among local processes respectively. In both figures, normal case is the case where no interference virtual machines run, and interference case indicates interference virtual machines are available. We can see that both random and sequential memory access latency time rise up, but the memory space expansion' influence is not obvious. Beside that, the memory bandwidth of tested virtual machine has suffered serious.

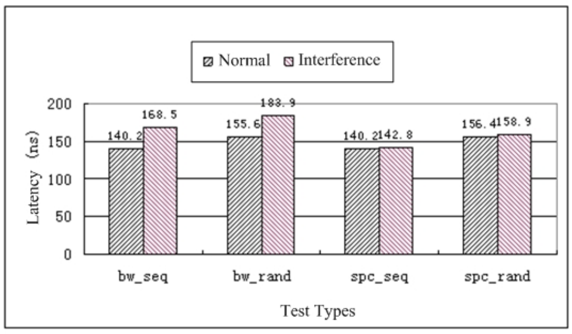

Figure 6 Performance isolation results on memory bandwidth

The first two charts in figure 6 are results under memory bandwidth interference, the next two are the results under memory space consumption. "*seq" and "*rand" denote sequential access and random access respectively. Here, sequential access means that a group of elements is accessed in an ordered sequence, for example if we simply want to process a sequence of data elements in order. And random access is to access an arbitrary element of data. According to figure 6, with interference programs available, the performance of the target virtual machine, especially memory bandwidth declines: the sequential access latency varies from 140.2 *ns* to 168.5 *ns*, and random access latency from 155.6 *ns* to 183.9*ns*. However, when interference programs of memory consumption run, the sequential access latency increases from to 140.2 *ns* to 142.8 *ns*, and random access latency from 156.4*ns* to 158.9*ns*.

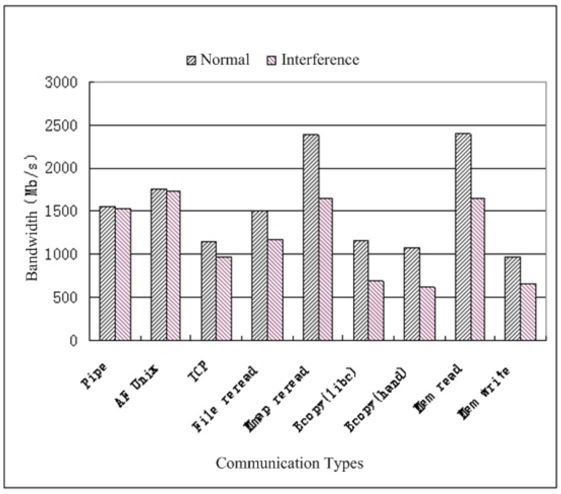

Figure 7 Performance isolation results on local communication bandwidth

Figure 7 also shows communication bandwidth declines when interference virtual machines run. Moreover, we can find the impact on memory consumption test is relatively little.

The reason is that VMM provides few policies to control memory access and guest operating systems behaves as they use entire computer exclusively. Xen does not guarantee that a domain will receive contiguous regions of physical memory. However, most operating systems do not have good support for operating in a fragmented physical address space. For the purpose of aid running such operating systems on top of Xen, Xen distinguishes between two kinds of memory: machine memory and pseudo-physical memory. Xen maintains a globally readable machine-to-physical (M2P) table which maps machine addresses to pseudo-physical addresses. In addition, a physical-to-machine (P2M) table which maps pseudo-physical addresses to machine addresses is also available in each domain. When a virtual machine has started, its memory size has been set by hypervisor [2]. When the virtual machine requests a new page, it submits a page request to the hypervisor then traps in the hypervisor. The hypervisor will set up some page table entries in P2M and M2P [2]. If the interference virtual machines request and free memory frequently, although the interference programs for memory space has introduced somewhat slight impacts on memory space consumption of the target virtual machine, this brings about the high latency and low local communication bandwidth.

5.4 Disk interference evaluation

For evaluating performance isolation of disk under virtualization environment, except domain 0, we firstly start two domains: domain 1 and domain 2 on the server. Domain 1 is target VM, and domain 2 is interference VM. In domain 1, we run a task which is consisted of a series of file-I/O operations. Those file-I/O operations are to read, directly read (dioread), write, directly write (diowrite), reread or rewrite 20,000 8KB blocks. In domain 2, we run misbehavior programs in which file-I/O operations are same as domain 1 and request size is changed from 1 to 16 blocks (1 block is 8KB). We also evaluate performance isolation of different file-I/O operations are executed in 2 domains. We run vBench and results are shown in figure 8. the "*readwrite*" curve in figure 9 means domain 1 executes read operation and domain 2 executes write operation.

The reason is that the default I/O scheduler of the real device driver is CFQ (Completely Fair Queuing), although there are other three schedulers: noop scheduler, anticipatory scheduler and deadline scheduler. CFQ works by placing synchronous requests submitted by processes into a number of per-process queues and then allocating time slices for each of the queues to access the disk. Asynchronous requests for all processes are batched together in fewer queues, one per priority. CFQ can only process 4 requests in a queue every schedule, in other words, at most 4 requests of a process are responded. Thus, it guarantee fairness in which equal number of requests of every process is processed. However, CFQ does not consider data size of requests. Thus a process can be serviced longer when it submits a large data size request. It will lead to unfairness.

The driver is unaware of requests from different virtual machines, and has no idea about protecting one virtual machine from another. So this leads to the unfairness among virtual machines. Thus, when other VMs request more blocks, the file-I/O performance of target VM is impacted. And the performances of direct read and direct write are impacted more seriously. Because direct read and direct write call lower level operations to access disk without buffer, thus these cases introduce more interference to other VMs. Comparing the performance of read with the performance of write, the read performance is more easily impacted than write performance. The reason is that read operation must wait for completion of disk operation before return. If disk responses requests of other VMs, it will lead to bigger response time of request. However, when guest OS write data to files, guest

OS generally mark the memory dirty and return. The kernel finally writes data to disk. Therefore, write operation is not sensitive to interference from other VMs.

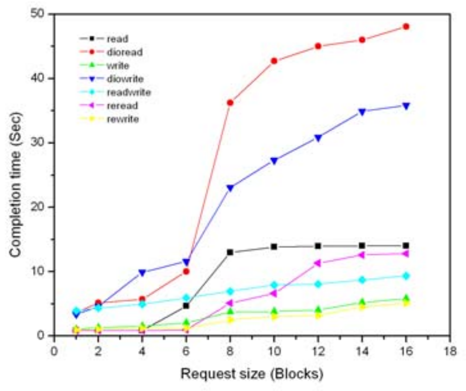

Figure 8 Completing time of the task on target VM when interference VM increases request size

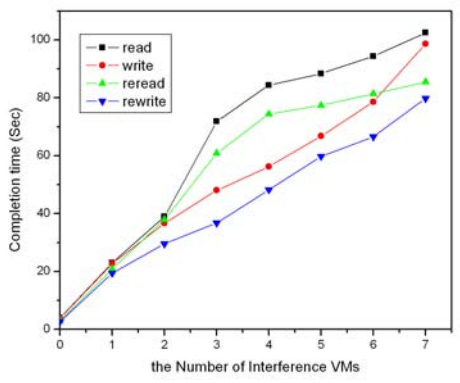

Figure 9 Completing time of the task on target VM under varied VMs

VITS can also evaluate performance isolation when the number of VMs varies. For the purpose of running more VMs on the server, each VM has 128M memory and 1 VCPU. Both target VM and interference VMs execute same I/O operations, such as *read*, *write*, *reread* and *rewrite*. The experimental results are shown in figure 9. Regardless of I/O operations executed, we find that the performance of four file-I/O operations on the target VM is impacted with the growth of VMs running on same physical machine. We also find that the *write*, *reread* performance is better than *read* performance and *rewrite* performance is the best among four I/O operations.

5.5 Interference evaluation for network

We performed experiments involving sending packets of different sizes at a fixed rate to a guest VM. In particular, we varied the datagram size from 0 to 16384 bytes. The experimental results are shown in figure 10.

According to figure 10, when the datagram size from interference VM is bigger, the netwrok bandwidth of target VM declines. The reason is that two VMs share a physical NIC, thus, target VM and interference VM contend the network bandwidth.

In the experiments, the datagram size of target VM is kept same as 256B. When interference VM sends 1 B to 256B data, target VM can send more data during every transmission comparing with interference VM since the datagram size of interference VM is smaller. Thus target VM will seize more network bandwidth. When the datagram size sent by the interference virtual machine increases from 256B, 512B to 1024B, the network bandwidth of target VM is unchanged. The reason is that the datagram size is less than MTU which is 1024B here and datagram can be encapsulated into a packet for transmission. Moreover, the overhead of network I/O is consisted of: data copying among domU to dom0; replicate data among dom0 to real device driver through virtual bridge. However, CPU time required to deal with datagram is mainly relevant with memory exchange, not relevant with datagram size [18].

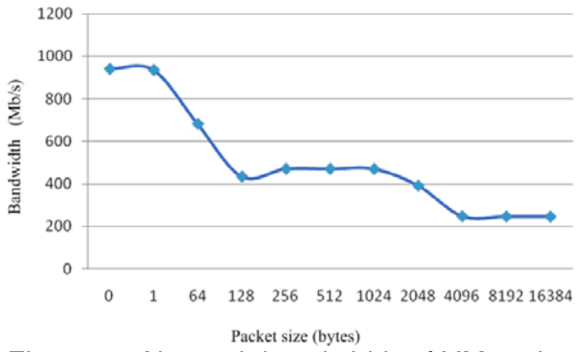

Figure 10 Network bandwidth of VM varies with packet size of interference VM

It can be further confirmed when we send datagrams the size of which are larger than MTU. The results are also shown in figure 10. When interference virtual machine sends datagrams the size of which is larger than 1024B, due to limitation of MTU, the datagram will be broken into a number of pieces that can be reassembled later. Thus, interference virtual machine will send more packets and contend for more bandwidth and fewer bandwidth is available for target virtual machine.

6 Conclusion

As virtualization systems for commodity platforms become more and more common, the importance of benchmarks that compare virtualization environments increases. The issue of isolation from misbehaving virtual machines is an important one to consider, especially for a commercial hosting environment. We have demonstrated the importance of having a performance isolation test suite that considers multiple types of misbehavior. We have designed such a suite and present results we obtained from using it to Xen. We found that the cache, memory and disk performance isolation have some big weak points to improve, in other words, the current virtualization system Xen is still unfair in fine-grained resource share.

The contributions of the paper are: firstly, we design a micro-benchmark for evaluating performance isolation of virtualization technology. Secondly, VITS test suite is equipped with some programs which can evaluate performance isolation of cache, memory, disk and network. What is more important is, the interference programs are fine granularity, since VITS Test Suite is designed for evaluate micro performance isolation.

Acknowledgements

This work is funded by the National 973 Basic Research Program of China under grant No. 2007CB310900.

References

[1] B. Clark, T. Deshane, E. Dow, S. Evanchik, M.Finlayson, J. Herne and J. Matthews. *Xen and the art of repeated research*. In Proc. of the USENIX 2004 Annual Technical Conference, pp. 135-144, June 2004.

[2] 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 Proc. of the 19th Symposium on Operating System Principles, 2003: 164–177

[3] D. Gupta, R. Gardner, and L. Cherkasova. *XenMon: QoS monitoring and performance profiling tool*. Technical Report HPL-2005-187, HP Laboratories, Palo Alto, CA, USA, 2005

[4] A. Menon, J. R. Santos, Y. Turner, G. (John) Janakiraman, and W. Zwaenepoel. *Diagnosing performance overheads in the Xen virtual machine environment*. In Proc. of the 1st International Conference on Virtual Execution Environments, 2005: 13–23

[5] J. P. Casazza, M. Greenfield, and K. Shi. *Redefining server performance characterization for virtualization benchmarking*. Intel Technology Journal, 10(03), 2006.

[6] V. Makhija, B. Herndon, P. Smith, L. Roderick, E. Zamost, and J. Anderson. *VMmark: A scalable benchmark for virtualized systems*. Technical Report VMware-TR-2006-002, 2006

[7] SPECweb 2005, *http://www.spec.org/web2005*, Accessed January 2007.

[8] L. McVoy and C. Staelin. *lmbench: Portable tools for performance analysis*. In Proc. of the USENIX'96, January 1996: 279–295

[9] C. Staelin, L. McVoy. *mhz: anatomy of a micro-benchmark*, In Proc. of the USENIX'98, USA, June 1998: 27-41

[10] K. T. Möller. *Virtual machine benchmarking*, dilpoma thesis, Karlsruhe University, 2007

[11] J. N. Matthews, W. Hu. *Quantifying the performance isolation properties of virtualization systems*, In Proc. of the 2007 workshop on Experimental computer science, 2007

[12] J. Sugerman, G. Venkitachalam, and B. Lim. *Virtualizing I/O devices on VMware Workstation's hosted virtual machine monitor*. In Proc. Of USENIX'01, Jun 2001.

[13] K. Adams and O. Agesen. *A comparison of software and hardware techniques for X86 virtualization*. In Proc. of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems. USA: ACM Press, 2006: 2-13

[14] C. Staelin, *lmbench -- an extensible micro-benchmark suite*, Technical Reports HPL-2004-213

[15] J. Wei, Jeffrey R. Jackson, John A. Wiegert, *Towards scalable and high performance I/O virtualization - a case study*. In Proc. of 3rd International Conference on High Performance Computing and Communications, Houston, USA, September 26-28, 2007: 586-598

[16] W. Yu, J. Vetter, *Xen-based HPC: a parallel I/O perspective*, In Proc. of the 8th IEEE International Symposium on Cluster Computing and the Grid, 19-22 May 2008:154 – 161

[17] H. Raj, K. Schwan, *High performance and scalable I/O virtualization via self-virtualized devices*, In Proc. of the 16th international symposium on High performance distributed computing, Monterey, California, USA, June 25 - 29, 2007:179-188

[18] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. *Enforcing performance isolation across virtual machines in Xen*. In Proc. of the ACM/IFIP/USENIX 7th International Middleware Conference, Melbourne, Australia, LNCS 4290, Springer, November 2006: 342–362

[19] A. B. Brown and M. I. Seltzer. *Operating system benchmarking in the wake of lmbench: a case study of the performance of NetBSD on the Intel x86 architecture*. In Proc. of SIGMETRICS'97: 214–224

[20] A. S. Tanenbaum, A. S. Woodhull. *Operating system design and implementation*. Prentice Hall, January 04, 2006.