Who is Your Neighbor: Net I/O Performance Interference in Virtualized Clouds

Xing Pu, Ling Liu, *Senior Member, IEEE*, Yiduo Mei, Sankaran Sivathanu, Younggyun Koh, Calton Pu, *Senior Member, IEEE*, Yuanda Cao

Abstract—User-perceived performance continues to be the most important QoS indicator in cloud-based data centers today. Effective allocation of virtual machines (VMs) to handle both CPU intensive and I/O intensive workloads is a crucial performance management capability in virtualized clouds. Although a fair amount of researches have dedicated to measuring and scheduling jobs among VMs, there still lacks of in-depth understanding of performance factors that impact the efficiency and effectiveness of resource multiplexing and scheduling among VMs. In this paper, we present the experimental research on performance interference in parallel processing of CPU-intensive and network-intensive workloads on Xen Virtual Machine Monitor (VMM). Based on our study, we conclude with five key findings which are critical for effective performance management and tuning in virtualized clouds. First, co-locating network-intensive workloads in isolated VMs incurs high overheads of switches and events in Dom0 and VMM. Second, co-locating CPU-intensive workloads in isolated VMs incurs high CPU contention due to fast I/O processing in I/O channel. Third, running CPU-intensive and network-intensive workloads in conjunction incurs the least resource contention, delivering higher aggregate performance. Fourth, performance of network-intensive workload is insensitive to CPU assignment among VMs, whereas adaptive CPU assignment among VMs is critical to CPU-intensive workload. The more CPUs pinned on Dom0 the worse performance is achieved by CPU-intensive workload. Last, due to fast I/O processing in I/O channel, limitation on grant table is a potential bottleneck in Xen. We argue that identifying the factors that impact the total demand of exchanged memory pages is important to the in-depth understanding of interference costs in Dom0 and VMM.

Index Terms—cloud computing, performance measurement, virtualization.

– – – – – – – – – – **u** – – – – – – – – – –

1 INTRODUCTION

Virtualization technology [19], [21] offers many advantages in cloud based data centers, such as reducing total costs of ownership, improving energy efficiency and resource utilization. By providing physical resources sharing, fault isolation and live migration, virtualization allows diverse applications to run in isolated environments through creating multiple virtual machines (VMs) on shared hardware platforms, and manage resource sharing across VMs via virtual machine monitor (VMM) [2]. Although VMM has the ability to slice resources and allocate the shares to different VMs, our measurement study shows that the performance of applications running in one VM would be affected by the applications running on its neighbor VMs, especially when these VMs are running at high rates of network I/O workloads. More importantly, our study shows that the level of performance interference mainly depends on the degree of the competition that the concurrent applications running in separate VMs may have in terms of shared resources. We argue that the in-depth understanding of potential interference factors among VMs running on a shared hardware platform is critical for effective management in virtualized clouds, and an open challenge in current virtualization research and deployment.

In this paper, we study performance interference among multiple VMs running on the same hardware platform with the focus on network I/O processing. The main motivation for targeting our measurement study on performance interference of processing concurrent network I/O workloads is simply because network I/O applications are becoming dominating workloads in most cloud based data centers now. By carefully designing of measurement testbed and the set of performance metrics, we derive some important factors of I/O performance conflicts based on application throughput interference and net I/O interference. Our performance measurements and analyses also provide some insights into performance optimizations for CPU scheduler and I/O channel, and as well efficiency management of workloads and VM configurations, such as CPU assignment across VMs.

This paper is structured as follows: Section 2 gives related work. Section 3 presents the overview of Xen I/O model, testbed setups, I/O workloads and the metrics used for performance interference analysis. Section 4 reports our experimental results and analysis of the performance interferences with respect to throughput and net I/O under different co-location of workloads and different allocation of CPUs to VMs. Section 5 concludes the paper with five important contributions and future works.

- ***X. Pu and Y. Mei were visiting PhD students at the Distributed Data intensive System Lab (DiSL) of Georgia Institute of Technology from 2008-2010. X. Pu is with the school of Computer Science in Beijing Institute of Technology, Beijing, P. R. China, 100081. Y. Mei is with the department of Computer Science in Xi'An Jiaotong University, Xi'An, Shanxi, P. R. China, 710049. E-mail: {abenpu, meiyiduo} @gmail.com.***
- ***L. Liu, S. Sivathanu, Y. Koh, and C. Pu are with the College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332. E-mail: {lingliu, sankaran, young, calton} @cc.gatech.edu.***
- ***Y. Cao is with the school of Computer Science in Beijing Institute of Technology, Beijing, P. R. China, 100081. E-mail: ydcao@bit.edu.cn.***

Manuscript received 24 May 2011.

2 Related Work

Among mainstream VMM products, a relative efficient I/O performance and a safe execution environment are provided by the driver domain model, represented by Xen [27], [28], KVM [29], and Microsoft's Hyper-V [30]. A main feature of such a drive domain model is to ensure execution safety in the driver domain model by sacrificing I/O performance. Thus, a fair number of research projects have been dedicated to I/O optimization, CPU scheduling, measuring tools, and fault isolation. However, there still lacks of in-depth understanding of performance interference factors that impact the efficiency and effectiveness of resource multiplexing and scheduling among neighbor VMs.

Xen VMM was first introduced by Barham ***et al.*** [2], [3] from the University of Cambridge at 2003. Hypervisor layer and privileged driver domain are implemented to ensure better fault isolation effects. Earlier version of Xen VMM adopts "page-flipping" technique to exchange I/O data between driver domain and guest domain. It has been proved that the cost of mapping and unmapping pages was equivalent to the copy cost for large 1500 byte packets, which means page-flipping is less efficient than copy for small packet sizes [24], [25]. To address the problem of I/O efficiency in the driver domain model, K. Mansley ***et al.*** [16] proposed "direct I/O" to alleviate the pressure in driver domain by allowing guest domains to access hardware directly. Though the virtualized system can act as the native system in some cases, direct I/O is not considered in our work, as it lacks of dedicated driver domain to perform fault isolation, which is considered to be the most important function for system virtualization.

Some recent researches contributed to optimize grant mechanism. The purpose was to replace page-flipping by ***grant copy*** without giving up driver domain. J. R. Santos ***et al.*** discussed grant reuse originally in their work [25]. Grant reuse decreases the frequency of mapping/unmapping and communicating obviously. Subsequently, an enhanced grant mechanism was designed by K. K. Ram ***et al.*** in [22] and implemented in [23]. They adopted previous grant reuse, and added an I/O Translation Table (ITT) between driver domain and hypervisor to facilitate guest domains to perform grant invoke and revoke.

Most existing researches mentioned above focused on optimizing I/O efficiency based on physical host running one VM, while only few studies are related to performance isolation among neighbor VMs. D. Gupta ***et al.*** implemented XenMon [8] to monitor the detailed system-level values of each VM, such as CPU usage, I/O count and execution count. Based on the monitoring results, ShareGuard and SEDF-DC [9] mechanisms are proposed to improve performance isolation. Y. Koh ***et al***. [12] studied the effects of performance interference between two VMs hosted on the same physical platform by collecting the runtime performance characteristics of different types of concrete applications. Through subsequent analysis of collected characteristics, they predicted the performance of some new applications from its workload characteristic values successfully within an average error of 5%. However, our work reported in this paper, to the best of our knowledge, is the first one that provided a dedicated performance interference study on network I/O workloads running in separate VMs under multiple alternative hardware platforms, ranging from a dual-core CPU with small L2 cache platform to single core CPU with large L2 cache to a multi-core CPU platform.

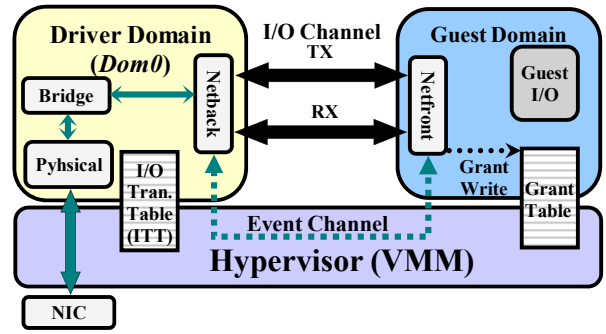


Fig. 1. Xen I/O Architecture.

3 Overview

In this section, we provide a brief background of Xen I/O architecture. Then we describe the experimental setup, including measurement method, I/O workloads and system-level metrics used for performance analysis.

3.1 Xen I/O Overview

Xen [2], [3] is a popular open-source x86 virtual machine monitor, supporting both full-virtualization and para-virtualization. Xen uses para-virtualization as a more efficient and lower overhead mode of virtualizations. In para-virtualization I/O mode, Xen VMM layer uses asynchronous ***hypercall*** mechanism to deliver virtual interrupts and other notifications among domains via event channel. A privileged domain called Dom0 is treated as driver domain hosting unmodified Linux drivers and has the access to hardware devices. Dom0 performs I/O operations on behalf of unprivileged guest domains which are ported to the virtual driver interface from Linux operating system (XenoLinux). Figure 1 shows the logical components of the latest Xen I/O model [6]. The virtual network interface in guest domain is called netfront acting as the real hardware drivers. In Dom0, netback is a counterpart for netfront. Netfront and netback use a bidirection ring of asynchronous requests to exchange data in I/O channel by sharing descriptors of memory pages pointed in I/O buffer. The bridge in Dom0 handles the packets from NIC and performs the software-based routine to destination VM.

When a network packet is received by the NIC (RX), it raises an interrupt to the upper layer. Before the interrupt reaches Dom0, hypervisor (VMM) handles the interrupt first. Hypervisor will determine whether or not Dom0 has the access to the real hardware. Upon receiving the interrupt, Dom0 starts to process the network packet. It first removes the packet from NIC and sends the packet to the bridge. Then the bridge de-multiplexes the packet and delivers it to the appropriate netback interface. Netback raises a ***hypercall*** to hypervisor, requesting an unused memory page. Hypervisor notifies corresponding guest

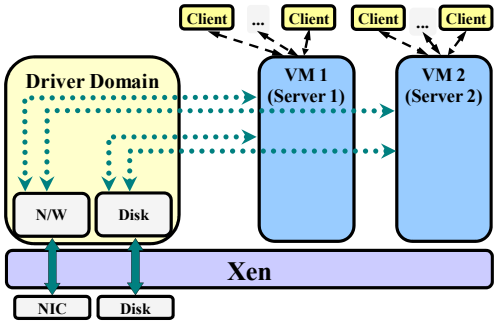


Fig. 2. Logical components of virtualized cloud environments. (Web servers reside in VM1 and VM2. Driver domain is Dom0).

domain to grant a page to keep the overall memory allocation balanced. Netback copies the received data to granted page in guest domain. Finally, guest domain receives the packet as if it comes directly from NIC. A similar but reverse procedure is applied to send a packet on the send path (TX), except that no explicit memory page exchange is involved. Only the ownership of physical page is transferred instead of the real page.

To provide a safe execution environment among VMs, Xen introduces a series of dedicated isolation mechanisms. The most outstanding one is grant mechanism, which is used to protect the I/O buffer in guest domain's memory and share the I/O buffer with Dom0 properly [22]. At the boot time, hypervisor creates a unique grant table for each guest domain, which only can be accessed by hypervisor and guest domain. Subsequently, guest domain initializes an associated memory I/O buffer shared with Dom0. When Dom0 requests to exchange the I/O data with guest domain, guest domain invokes a grant, and then simply write a free entry in grant table with three key information: 1) valid memory page address; 2) Dom0's id; 3) operation permission (***read-only*** for transmission or ***read-write*** for reception). After the grant writing operation is finished in grant table, guest domain issues a ***hypercall*** through hypervisor to Dom0 to pass a corresponding grant reference, which indexes the correct entry in grant table. Then Dom0 can find the grant entry by this grant reference, meanwhile, hypervisor validate Dom0 to perform the I/O operation in I/O buffer. When I/O exchange is accomplished or guest domain wants to repurpose the granted page, another ***hypercall*** is issued to hypervisor. And then, hypervisor synchronizes access permission of memory page between Dom0 and guest domain. If Dom0 gives up the granted page, guest domain revokes the grant by another simple writing operation in grant table. Dom0 loses the access permission.

3.2 Testbed Architechure

We carefully designed our experiments to exercise net I/O traffics and evaluate the performance interference. Figure 2 gives a sketch of experimental setup used in this paper. Two isolated guest domains (i.e., VM1 and VM2) with equal resource allocations are hosted on the same physical platform. Apache servers provide the HTTP services in VM1 and VM2 respectively. Clients using httperf [13] as HTTP "load generator" are designed to access virtualized servers remotely. They send requests to corresponding VM to retrieve a fixed size file: 1 KB, 4 KB, 10 KB, 30 KB, 50 KB, 70 KB, 100 KB or 1 MB. These I/O workloads are carefully selected from SPECweb'99, SPECweb'2005 and SPECweb'2009, which are representative log sizes in current data center. Additional, to reduce the disk readings, I/O workloads are cached in the buffers in advance. All tests were performed on three alternative platforms presented in Table 1. Platform I and Platform II have dissimilar hardware settings, especially the capacities of CPU and NIC. As we will show in the next section, I/O interference on these two platforms is quite obvious. Platform III is a four-core workstation, which is used to study the performance impact of different strategies of CPU allocation among VMs. Physical machines used in these three scenarios are all connected by a high speed Ethernet switch. The Linux Kernel 2.6.18.8-xen is recommended for Xen 3.4.0 and 2.6.32.10-xen is recommended for Xen 4.0.2 respectively.

TABLE 1
EXPERIMENTAL ENVIRONMENT

Capacity	Platform I	Platform II	Platform III
CPU Type	Pentium 4	Xeon™	Xeon™
Processors	2	1	2
Cores	2	1	4
CPU Freq. (GHz)	3.2	3.0	3.0
L1 Cache (KB)	16	256	256
L2 Cache (MB)	2	6	12
Memory (GB)	2	8	8
DISK (GB)	250	500	500
NIC (Mbit/s)	100	1000	1000
MTU (Byte)	1500	1500	1500
Kernel	2.6.18.8-xen	2.6.18.8-xen	2.6.32.10-xen
Xen	3.4.0	3.4.0	4.0.2

3.3 I/O Workloads

For each of the three alternative hardware platforms in Table 1, we first evaluate the actual performance results and characteristics of each I/O workload running in single guest domain, which serves as ***basecase*** in the rest of this paper. We only discuss the experimental details of ***basecase*** on Platform I, a representative physical host for performance interference analysis in next two subsections.

Table 2 shows the maximum performance results of one guest domain running under the selected net I/O workloads on Platform I. When server becomes saturated at full capacity, 1 KB and 4 KB files reach 0.5 to 15 times higher request throughput than others respectively, and consume more than 97% CPU resource (approximately, remaining 2.5% CPU is charged by idle loop and monitor tool), while network bandwidth utilizations are only around 20% and 60% respectively. The web mix performance of these two workloads is limited by CPU resource. Although the achieved request rates are not higher than 1200 req/sec, 10-100 KB workloads saturate the server by consuming all network bandwidth, which is 10 KB × 1104 req/sec ≈ 30 KB × 375 req/sec ≈ 50 KB × 225 req/sec ≈ 70 KB × 160 req/sec ≈ 100 KB × 112 req/sec ≈ 100 MB/sec. Meanwhile, when VM1 is serving one of these five workloads (10-100 KB), the total CPU utilization is less than 75%. These reveal that 1 KB and 4 KB workloads are CPU bounded and 10-100 KB workloads are network bounded, consistent with the observation made from prior

TABLE 2
MAXIMUM PERFORMANCE OF WORKLOADS IN BASECASE ON REPRESENTATIVE PLATFORM I

Workload	Major Resource Used	Throughput (Req/sec)	Net I/O (KB/sec)	Response Time (ms)	CPU (%)
1 KB	CPU	1900	2018	1.52	97.50
4 KB	CPU	1750	7021	5.46	97.46
10 KB	Network	1104	11048	2.36	70.44
30 KB	Network	375	11271	2.52	54.87
50 KB	Network	225	11262	2.7	49.62
70 KB	Network	160	11255	2.84	47.10
100 KB	Network	112	11208	2.08	44.40

research [4], namely short file is CPU bounded and long file is network bounded. With higher hardware capacities on Platform II and III, 1-100 KB workloads are CPU bounded and 1 MB workload is network bounded.

3.4 Initial Interference Analysis and Metrics

In this section we outline the methodology and metrics used for our measurement study.

Let Dom_0, Dom_1 … Dom_n be the VMs running on the same host, where Dom_0 is Dom0. Suppose that Dom_i is serving workload i, we define the maximum throughput of Dom_i as T_i. We use B_i to denote the maximum throughput of Dom_i in ***basecase*** scenario where n equals 1, i.e., only Dom0 and one guest domain are hosted. Then, the combined normalized throughput is:

$$Combined\ Score = \sum_{i=1}^{n} \frac{T_i}{B_i}$$

Figure 3 presents the set of experiments conducted on the setup of two guest domains with each serving one of the selected workloads on Platform I. We measure the normalized throughput scores of different combinations of selected net I/O workloads. The base-case throughput used in this experiment is the throughput of workloads in the single guest domain ***basecase*** on Platform I. To present the results clearly, each combination of two workloads running in VM1 and VM2 is denoted in a tuple of two elements, with the first element x is xKB file retrieving from VM1, and the second element y is the yKB file retrieving from VM2. For example, the expression (1K, 30K) says VM1 serves 1 KB file and VM2 serves 30 KB file. (1K, Idle) refers to the case where VM1 serves 1 KB workload and VM2 is idle. From Figure 3, we observe some interesting facts regarding performance interference. First, (1K, 100K) achieves the best performance with its combined throughput score of 1.58. Given that 1 KB workload is CPU bounded and 100 KB workload is network bounded, this combination clearly incurs the least resource contention compared to other combinations. Similarly, (1K, 50K) is the next best pairing for the similar reason. (30K, 100K) and (50K, 100K) offer better combined throughput than the worst combinations of (1K, 1K) and (1K, 10K), which incur highest resource contention. This set of experiments indicates that given Xen I/O model and inherent resource sharing principle across VMs, net I/O performance interference is unavoidable. Although Xen uses Dom0 model to provide fault isolation successfully, Dom0 may easily become the bottleneck when VMs wish to get accesses to underlying hardware or communicate

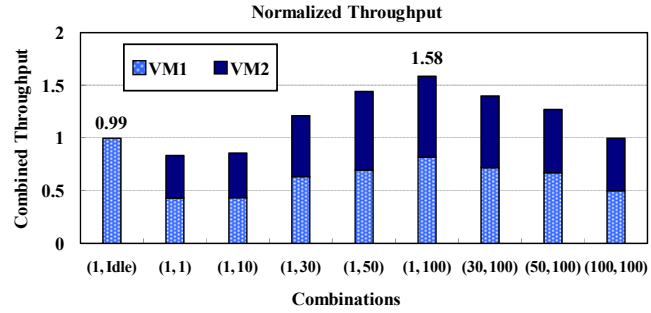


Fig. 3. Normalized throughput on Platform I.

with others. Three suspicious points exist here: 1) all events have to go through Dom0 and VMM layer. This is supposed to cause communication or control interferences among VMs; 2) the multiplexing and demultiplexing of I/O channel may incur memory management interference, such as grant management and fragmentations; 3) the default ***Credit scheduler*** may affect the overall performance of neighbor VMs.

To understand these three suspicious points, we collect eight system-level characteristics from I/O workloads:

VMM events per second (Event). The number of events per second among VMs in event channel. This metric reflects communication interference among VMs

VM switches per second (Switch). The number of VMM switches controls among VMs per second. This metric reflects control interference among VMs.

I/O count per second (Pages Exchange). During the period of one second, the number of exchanged memory pages in I/O channel is captured.

Executions per second (Execution). This metric refers to the number of execution periods per second for each VM.

I/O count per execution (I/O Execution). During the period of one execution, the number of exchanged memory pages is captured. This metric could help to reflect the efficiency of I/O processing in I/O channel. We divide "Pages Exchange" by "Execution" to calculate "I/O Execution".

CPU utilization (CPU). We measured the average CPU utilization of each VM, including CPU usage of Dom0, VM1 and VM2 respectively. The total CPU consumption is the summation of them all.

VM state (Waiting, Block). At any point of time, every VM must be in one of following three states: ***execution state***, ***runnable state*** and ***blocked state***. When one VM is currently using CPU, it is in the ***execution state***. The metric "Waiting" is the percentage of waiting time when a VM is in ***runnable state***, or in other words, the VM is in the CPU run queue but do not utilize CPU. The metric "Block" is the percentage of blocked time when a VM is in ***blocked state***, which means a VM is blocked on I/O events and not in the run queue.

The metrics of Event, Switch and I/O Execution directly reflect the interference of communication, control and I/O processing costs respectively. The remaining metrics can analyze the details of CPU utilization. These eight metrics are critical means for us to measure and understand the communication interference, VMM control interference and I/O processing costs, which are

TABLE 3
SYSTEM-LEVEL CHARACTERISTICS OF I/O WORKLOADS WITH WEB SERVER RUNNING IN VM1 ON PLATFORM I. (VM2 IS NOT CREATED, I.E. *BASECASE*)

Workload	CPU (%)	Event (events/sec)	Switch (switches/sec)	I/O Execution (pages/exe)	Driver Domain (*Dom0*)			Guest Domain (VM1)		
					CPU (%)	*Waiting (%)*	*Block (%)*	*CPU (%)*	*Waiting (%)*	*Block (%)*
1 KB	97.50	*224104*	*9098*	*4.93*	46.83	4.81	10.54	50.67	38.97	4.76
4 KB	97.46	*242216*	*10525*	*6.38*	46.82	4.77	10.68	50.65	37.72	5.65
10 KB	70.44	*279663*	*15569*	***7.28***	39.7	3.13	13.73	30.74	2.29	23.48
30 KB	54.87	*345496*	*26118*	*3.87*	36.78	1.92	17.78	18.09	1.31	34.32
50 KB	49.62	*342584*	*26981*	*3.54*	34.57	1.22	19.33	15.05	1.14	36.76
70 KB	47.10	*341898*	*27436*	*3.36*	33.41	0.90	19.46	13.70	1.07	37.62
100 KB	44.40	*332951*	*27720*	*3.17*	32.19	0.77	21.04	12.21	1.01	39.08

important for our performance interference analysis. First, we use these eight metrics to illustrate underlying details of previous Platform I ***basecase*** study. Then the in-depth factors of performance interference are studied.

Table 3 shows the ***basecase*** results of seven selected net I/O workloads for the most interesting metrics outlined on Platform I. The values of each workload characteristics are presented at 100% load for the given workload type. Comparing with network-intensive workloads of 30 KB, 50 KB, 70 KB, and 100 KB files, CPU-intensive 1 KB and 4 KB workloads have at least 30% and 60% lower event and switch costs respectively for the network I/O processing is more efficient in these cases. Concretely, for 1 KB and 4 KB workloads, by comparing waiting and block time, Dom0 has to wait about 2.5 times longer on the CPU run queue for being scheduled into the ***execution state*** and the guest domain has 30 times longer waiting time. However, they are blocked infrequently for acquiring more CPU resource, especially in guest domain the block time is less than 6%. We notice that the borderline network-intensive 10 KB file owns the most efficient I/O processing ability (7.28 pages/exe), while the event and switch numbers are 15% larger than CPU-intensive workloads. Interesting to note is that initially, I/O execution is getting more and more efficient as file size increases from 1 KB to 10 KB. However, with file size of the workload grows larger and larger (30-100 KB), more and more packets need to be delivered for each request. The event and switch number are increasing gradually as observed in Table 3. Note that the events per second are also related to request rate of the workload. Though it drops slightly for the workload of file size 30-100 KB, the overall event number of network-intensive workloads is still higher than CPU-intensive ones. With increased file size of shorter workloads (1 KB to 10 KB), VMs are blocked more and more frequently from 4.76% to 23.48%. Finally, I/O execution starts to decline from 7.28 to about 3, when file size is larger than 10 KB. Network I/O workloads that exhibit CPU bound are now transformed to network bounded as file size of workloads exceeding 10 KB and the contention for NIC is growing higher as workload size increasing.

Similar measurements and analyses can also be applied to ***basecase*** on Platform II and III, whose observations are consistent with Platform I. Understanding these basic system-level characteristics in ***basecase*** scenario helps us to better understand factors that cause different levels of performance interference with respect to throughput, net I/O and multi-core on Platform I, II and III.

4 PERFORMANCE INTERFERENCE ANALYSIS

In this section, first, performance interference on Platform I and II are presented with workload load rates varying from 10% to 100%, running selected net I/O workloads. These sets of measurements are used to analyze throughput and net I/O interference due to resource contentions displayed by various workload combinations. Then, we analyze multi-core performance interference on Platform III by varying CPU assignments among VMs.

4.1 Throughput Performance Interference

This subsection focuses on studying interference of running multiple net I/O workloads in isolated VMs. We first show the performance interference on Platform I and II, and then provide analysis of performance interference in each of the two setups respectively, focusing on understanding the impacts of running different workloads in conjunction on the aggregate throughput.

4.1.1 Interference in Resource-competiting Workloads

In this group of experiments, the most CPU intensive workload and the most network intensive workload are chosen to study different co-location combinations for each platform. 1 KB file is a representative CPU bounded workload for both two platforms. The representative network bounded workload on Platform I is 100 KB file while on Platform II is 1 MB file. 1 MB workload on Platform II demands the same request rate as 1 KB file on Platform I, making comparison easily.

Figure 4 and Figure 5 show the set of experimental results of throughput interference on Platform I and II with three combinations of workloads running concurrently in two VMs, which are presented as (1K, 1K), (1K, 100K) and (100K, 100K) on Platform I in Figure 4 and (1K, 1K), (1K, 1M) and (1M, 1M) on Platform II in Figure5. In all figures, we use the sequence of tuple expression to denote the workloads running in VMs respectively. Take the tuple (1K, 100K) for example, it means the workload of 1 KB running in VM1 and the workload of 100 KB running in VM2. Also we use the notation of (1K, 100K)_1K to denote the measurement of 1 KB workload in VM1. To get a clear overview of curves on the same scale in x-axis, the load rates of different workloads in both figures were converted into the percentage of maximal achieved throughput in ***basecase***. For example, 100% load rate of 1 KB workload on Platform I (Figure 4) refers to the maximal request rate of 1900 req/sec in Table 2. Similarly,

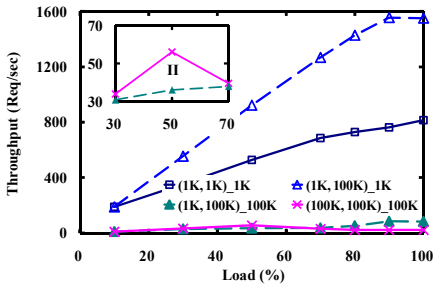


Fig. 4. Throughput performance on Platform I.

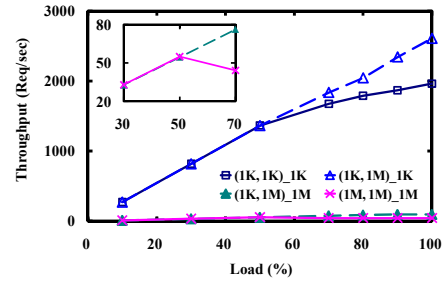


Fig. 5. Throughput performance on Platform II.

for 100 KB workload on Platform I, the 100% load rate means the maximum request rate 112 req/sec. The maximal ***basecase*** throughput of 1 KB and 1 MB workloads on Platform II are 2730 req/sec and 110 req/sec respectively, shown as 100% load in Figure 5.

Figure 4 shows the throughput of three combinations on Platform I: (1K, 1K), (1K, 100K), (100K, 100K). We observe that (1K, 100K)_1K reaches around 1560 req/sec, which is twice of the maximal throughput of (1K, 1K)_1K, even though it is still 18% lower than the maximal throughput 1900 req/sec in ***basecase***. For (1K, 100K) combination, (1K, 100K)_100K achieves 85 req/sec at 100% load, which is 76% of maximal throughput in ***basecase*** and 52% higher than the throughput of (100K, 100K)_100K. Note that (100K, 100K) combination causes high network contention and saturates the host at 50% load (embedded chart in Figure 4). Figure 5 shows the throughput measurements of the workload combinations of (1K, 1K), (1K, 1M) and (1M, 1M) on Platform II. In the combinations of (1K, 1M) and (1K, 1K), 1 KB workloads achieve maximum rate both at 100% load, 2608 req/sec and 1962 req/sec respectively. The maximum throughput of (1K, 1M)_1K is only 4% lower than 2718 req/sec of 1 KB workload request rate in ***basecase*** and 33% higher than the maximum throughput of (1K, 1K)_1K. For network-intensive 1 MB workload, the achieved maximum rate is 97 req/sec at 100% load in (1K, 1M), which is 88% of the maximum throughput of 110 req/sec in ***basecase*** and 76% higher than the maximum rate of 55 req/sec for 1 MB workload in (1M, 1M). It is also observed that (1M, 1M) saturates the host at 50% load in the enhanced chart.

Recall Figure 3, workload combinations competing for the same resource, either CPU or NIC, stand a good chance of performance degradation compared to ***basecase***. Furthermore, in Figure 5, two workloads that exhibit distinct resource demands, such as one CPU bound and another network bound, running together also gets better performance than same workloads running concurrently on Platform II. Compare with 1 KB workload on Platform I, with increased L2 cache size, throughput degradations of CPU intensive 1 KB file are decreased on Platform II: from 18% to 4% in the combination of one CPU bound and one network bound; from 58% to 28% in the combination of both CPU bound workloads. Because of lower L2 cache misses [14], even with limited processor numbers and relative poor CPU frequency, Platform II achieves better performance for CPU intensive workload. However, L2 cache shows fewer effects on network intensive workload, they both saturate the hosts around 50% load rate.

4.1.2 Platform I: Throughput Interference Analysis

In this and next section, we provide some in-depth analysis of key factors that lead to different performance interference. This section gives the measurements and analysis on Platform I (Figure 6 to Figure 14). The details on Platform II are plotted and analyzed in the next section (Figure 15 to Figure 23).

From Figure 6 to Figure 14, we vary the load rate in x-axis and plot measured different performance metrics on y-axis. First, let us examine at the combination of two network-bound workloads of (100K, 100K) on Platform I. Figure 6 and Figure 7 show switch numbers and event numbers when varying load rates from 10% to 100%. Note that the number of events follows exactly same trend as switches, though the concrete values are different in scale. At 50% load, (100K, 100K) reaches the highest switch and event numbers, and starts to drop until load rate increases to 70%, then remains almost flat when load rate increases from 70% to 100%. Comparing with other two combinations, (100K, 100K) has at least 25% higher switch and event costs under the peak costs. This implies that main sources of overhead and throughput interference for the combination of (100K, 100K) may come from the high switch and event overheads in Dom0 and VMM.

Figure 8 shows the I/O Execution. As request rate getting higher, (100K, 100K) exchanges less than four pages per execution duration. It experiences the worst efficiency in I/O channel. Practically, heavy event and switch costs lead to a lot more interrupts that need to be processed, resulting in few pages exchange during each execution cycle. Figure 10 and Figure 13 present the block time of Dom0 and two guest domains respectively. For (100K, 100K), block time of Dom0 is around 30% after 50% load. In addition, the block time of guest domains are around 48% and both are relatively high compared to other two combinations. This indicates that VMs are frequently blocked for I/O events and waiting for the next CPU scheduling. Figure 11 and Figure 14 measure the waiting time of Dom0 and guest domains. We observe that (100K, 100K) has the lowest waiting time in both Dom0 and guest domains. This is mainly due to the high blocking time, as it reveals that CPU run queues are not crowd and could serve the VMs much faster.

In summary, we conclude that due to heavy event and switch costs in (100K, 100K) on Platform I, Dom0 and VMM are quite busy to do notifications in event channel, resulting in the fact that Dom0 needs more CPU while guest domains are waiting for I/O events, demanding

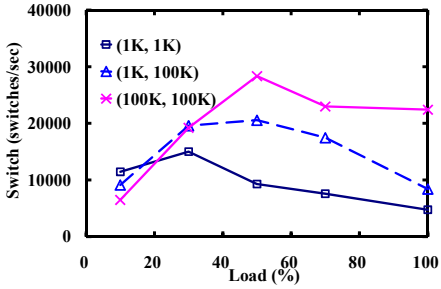


Fig. 6. Switches per second on Platform I.

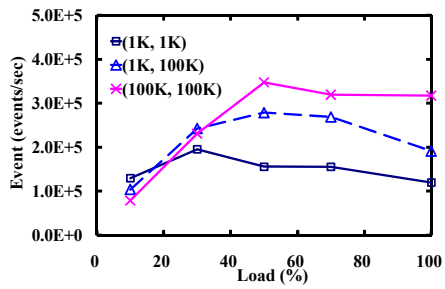


Fig. 7. Events per second on Platform I.

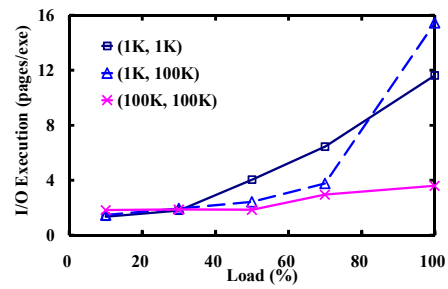


Fig. 8. Pages per execution on Platform I.

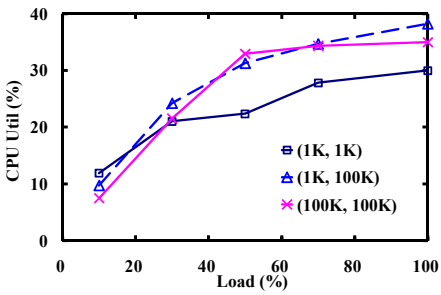


Fig. 9. *Dom0* CPU utilization on Platform I.

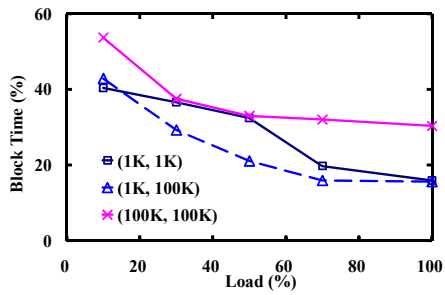


Fig. 10. *Dom0* Block Time on Platform I.

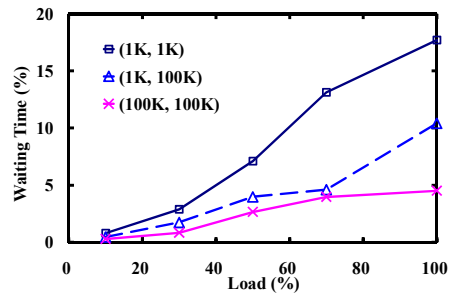


Fig. 11. *Dom0* Waiting Time on Platform I.

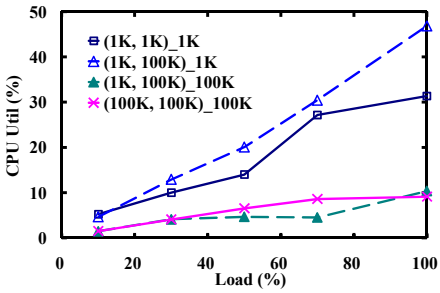


Fig. 12. Guests CPU usage on Platform I.

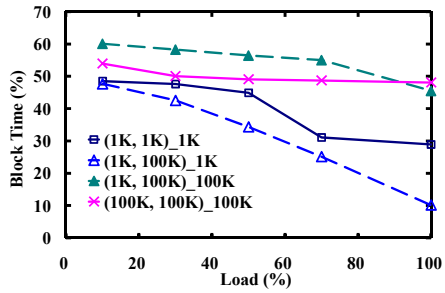


Fig. 13. Guests Block Time on Platform I.

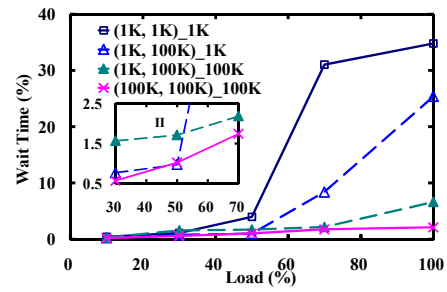


Fig. 14. Guests Waiting Time on Platform I.

fewer CPU (Figure 12).

For (1K, 1K) combination on Platform I, we observe that it has the lowest event and switch numbers in Figure 6 and Figure 7. Because (1K, 1K) incurs severe CPU contention, the poor throughput performance shown in Figure 4 should come from other factors. In Figure 8, (1K, 1K) combination processes the I/O communication more efficiently than (100K, 100K), which has almost three times I/O Execution under heavy load. Moreover, before 80% load, (1K, 1K) has the highest I/O Execution compared to other combinations. Thus, we infer that the throughput interference of (1K, 1K) may be caused by fast I/O processing between guest domains and Dom0. This conclusion can be further validated using experimental results shown in Figure 11 and Figure 14. We observe that waiting time of Dom0 and guest domains are both the longest in comparison with other curves, approximately achieving 17% and 34% respectively, i.e., CPU run queues are crowded with more vCPUs waiting for being put into ***execution state***. All VMs in the combination of (1K, 1K), including Dom0, are eager to use CPU resource. The ***Credit scheduler*** used in our tests is configured with equal ***weight*** for all VMs, i.e., each VM is dispatched with same CPU. In Figure 9 and Figure 12, CPU usage of Dom0 and guest domains have similar trend and sharing ($100\%/3 \approx 33\%$) when they are all demanding for CPU resource.

Briefly, to achieve high throughput, all VMs in (1K, 1K) need to perform fast I/O processing, which results in higher interference in CPU scheduling and lower throughput performance shown in Figure 4.

By analyzing throughput interference in combinations of (100K, 100K) and (1K, 1K) on Platform I, we understand that frequent memory page exchange in I/O channel leads to severe CPU contention among VMs in (1K, 1K), while (100K, 100K) combination incurs higher event and switch costs in VMM and Dom0, leading to high level of network contention. In comparison, the combination of (1K, 100K) founds a balance to achieve higher throughput with load rate increasing. Concretely, comparing with (1K, 1K)_1K, Dom0 and VM1 (i.e., (1K, 100K)_1K) experience ***blocked state*** infrequently and shorter waiting time, finally allocated over 10% additional CPU resource (Figure 10 and Figure 13). Comparing with (100K, 100K)_100K, for (1K, 100K)_100K, VM2 is blocked frequently and waiting longer to reduce event and switch costs (Figure 11 and Figure 14). Finally, page exchanges become more efficient under high load rate (Figure 8) and Dom0 is better utilized.

From this group of experiments on Platform I hardware capacity, we draw four concluding remarks:

1. Due to larger number of packets to be routed per HTTP request, the combination of both network-intensive workloads leads to at least twice higher

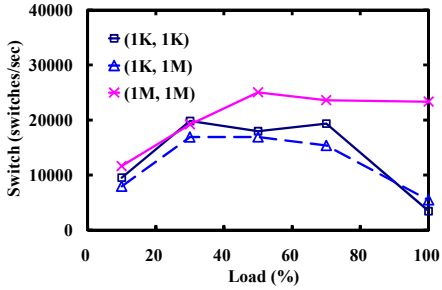


Fig. 15. Switches per second on Platform II.

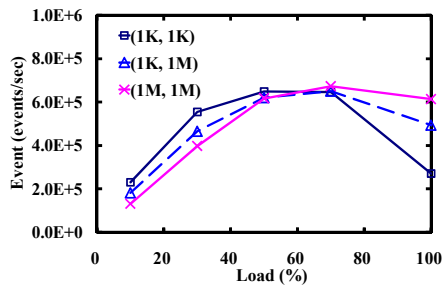


Fig. 16. Events per second on Platform II.

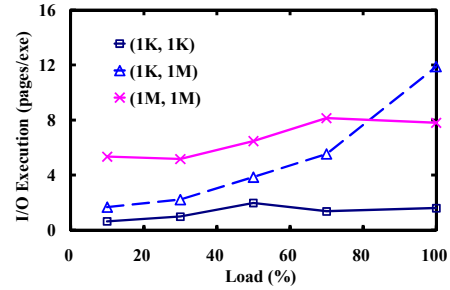


Fig. 17. Pages per execution on Platform II.

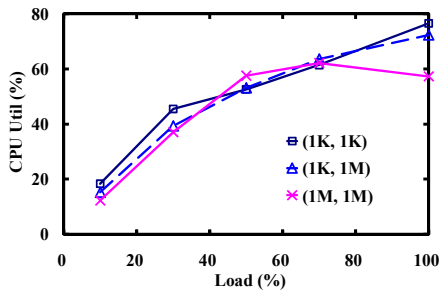


Fig. 18. *Dom0* CPU utilization on Platform II.

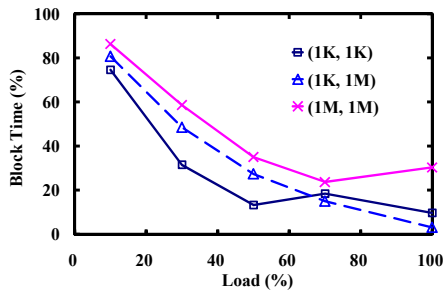


Fig. 19. *Dom0* Block Time on Platform II.

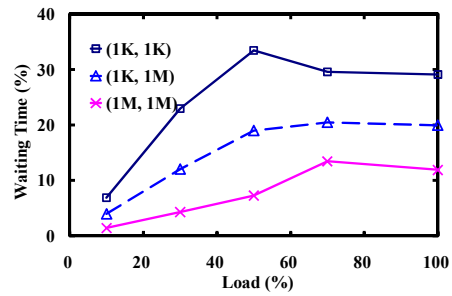


Fig. 20. *Dom0* Waiting Time on Platform II.

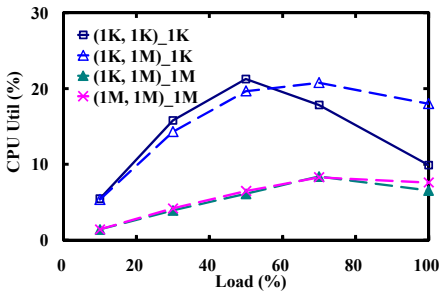


Fig. 21. Guests CPU usage on Platform II.

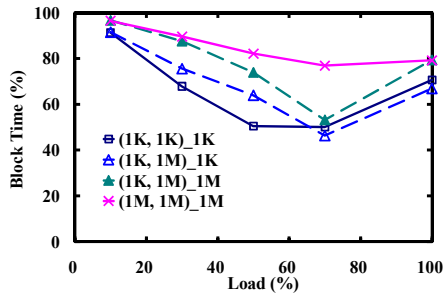


Fig. 22. Guests Block Time on Platform II.

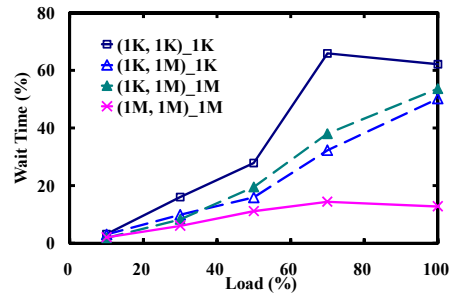


Fig. 23. Guests Waiting Time on Platform II.

event and switch costs in event channel, making driver domain busy for processing I/O interrupts, while leaving guest domains spending longer time on waiting for corresponding I/O events.

2. To achieve high throughput, the combination of both CPU-intensive workloads results in guest domains competing with driver domain for CPU resource to perform fast I/O processing, while the switch cost is the lowest, leading to certain level of performance interference due to CPU contention, though the degree of interference is relatively less severe when compared to (100K, 100K).
3. The workload combination with least resource competition is (1K, 100K). This is mainly due to the fact that it alleviates stresses in driver domain by scheduling extra CPU to ***Dom0*** and guest domain (VM1) serving CPU-intensive workload, meanwhile, it increases the block time and waiting time of guest domain (VM2) serving network-intensive workload to reduce switch overhead.
4. Interference is highly sensitive to the effectiveness of driver domain due to multiple VMs are competing for communication efficiency and fast I/O processing in driver domain.

4.1.3 Platform II: Throughput Interference Analysis

From Figure 15 to Figure 23, we present the measured metrics details on Platform II, to which the previous similar analysis approaches on Platform I could also be applied. Most previous conclusions still work under Platform II, only except the instance of two CPU-intensive workloads combination (i.e., (1K, 1K)). Let us focus on some interesting phenomena of (1K, 1K) on Platform II.

Figure 15 illustrates switch costs per second from load 10% to 100%. After 50% load, (1M, 1M) combination achieves the highest and relative stable switch cost. The other two combinations achieve the peak in the range 20% to 50% and start to drop with load increasing from 60% to 100%. Figure 16 gives measured event number per second in event channel. Comparing with cases on Platform I: 1) during the range 10% to 50%, (1M, 1M) does not exhibit highest event cost, while (1K, 1K) and (1K, 1M) spend at least more than 17% higher cost, and then (1M, 1M) gains the highest event cost after 70% load rate (the test is extended to 150% load); 2) with increased CPU capacities, messages in event channel get raised for CPU-intensive workload, while the trends of VMM switch controls among VMs remain relatively unchanged. In general, Figure 15 and Figure 16 are consistent compared with the corresponding cases on Platform I (Figure 6 and Figure 7).

The interference for co-locating network-intensive workloads comes from high event and switch costs.

Figure 17 shows the I/O Execution on Platform II. It is quite interesting that (1K, 1K) experiences the worst memory pages exchange efficiency in I/O channel. It has no more than 2 pages per CPU execution. Comparing with the details on Platform I (Figure 8), which indicates co-locating CPU-bound workloads leads to CPU contention for fast I/O processing, Figure 17 shows that there is another factor for performance interference: during each CPU execution time, the most burdensome job is not limited to memory pages exchange among VMs.

The details of vCPUs should tell us further information about this phenomenon. In Figure 18, the measured Dom0 CPU trends for three diverse combinations are plotted. (1M, 1M) combination maintains a flat CPU usage around 60% under heavy load. However, other two combinations go up straight to 70% CPU usage with load increasing from 10% to 100%. Especially, (1K, 1K) reaches almost 80% CPU usage. Comparing with the cases on Platform I in Figure 9, the highest CPU utilization is less than 40%, which is a half of Platform II. Dom0 has some kinds of jobs to do and demands more CPU on Platform II, because of quite few pages processed during each CPU execution. Figure 19 gives the block time of Dom0 and Figure 20 shows waiting time of Dom0. For (1K, 1K) combination, the block time of Dom0 is around 15% at 50%load and higher, which is the lowest block time curve, and the waiting time of Dom0 is around 30% after 50% load, which is the highest waiting time curve. This indicates that Dom0 is busy: Dom0′s vCPUs are scheduled into CPU run queues frequently, and CPU run queues are crowded with Dom0′s vCPUs.

At this point, it can be concluded that for (1K, 1K) combination on Platform II, the fast I/O processing is not the primary interference factor causing CPU contention, while other potential factors are in the leading position here. We notice that (1K, 1K) exhibits some specific numbers of the total CPU utilization on Platform II: with load rate increasing, the maximum CPU usage is 94.57% at 100% load. Eliminating the system idle and monitor tool costs, which are round 2.5% in totally, about 3% CPU resource are remaining. It seems that CPU resource is not the bottleneck here. Actually, grant mechanism incurs poor I/O efficiency and rises to be a main interference factor for the combination of two CPU-bound workloads on Platform II. We will illustrate more details and analysis with "zoom in" results in section 4.3.

From Figure 21 to Figure 23, these three figures present CPU utilization, block time and waiting time of guest domains for each representative combination respectively. The overall trends of these three figures hide some useful information. For CPU-intensive workloads combination (1K, 1K), guest domains experience relative higher block time percentage above 55% and the longest waiting time percentage around 60% under heavy load. In comparison with the Dom0, which has 15% block time and 30% waiting time after 50% load rate, guest domains do not have too much works to do, as Dom0 needs to get into CPU run queue so frequently. And since CPU run queue is full of Dom0, guest domains get such high waiting time percentages. For (1M, 1M), it exhibits the highest block time percentage, higher than 80% CPU time during the whole process, and lowest waiting time percentage in CPU run queue, stabilized around 10% waiting time after 70% load rate and higher. The CPU run queue of (1M, 1M) is not crowed. Virtualized system is busy to do the VM switch and event processing, which also proves our previous conclusions for network-intensive workloads combination. In summary, the overall block times of three workload combinations are all above 50% in Figure 22, which are higher than the cases on Platform I, this is because platform II has only one physical CPU and thus is easier to incur congestion in CPU run queue.

Although Platform I and Platform II are heterogeneous, we observe performance interference between workloads with high resource contentions due to running in a shared physical host through running in separate VMs. Moreover, we would like to draw two additional concluding remarks:

1. With lower L2 cache misses rate due to increased L2 cache size, even with poor CPU frequency, CPU contention will be alleviated and interference performance will be improved for both the combination of two CPU-bound workloads (i.e., (1K, 1K)) and the combination of one CPU-bound and one network-bound workload (i.e., (1K, 1M)).
2. With higher hardware capacities, especially larger L2 cache size, interference is still sensitive to the I/O efficiency in the privileged driver domain for the combination of two CPU-bound workloads. The grant table size is another potential bottleneck in addition to CPU resource.

4.2 Net I/O Performance Interference

From our throughput performance interference analysis in previous section, we know that the combination of both CPU-bound workloads (i.e., (1K, 1K) for Platform I and II) causes throughput interference mainly due to Dom0 demanding for fast I/O processing and grant mechanism management, while the combination of both network-intensive workloads (i.e., (100K, 100K) for Platform I and (1M, 1M) for Platform II) incurs the highest VMM switch overhead. Moreover, diverse workloads combinations (i.e., (1K, 100K) for Platform I and (1K, 1M) for Platform II) reach better performance by alleviating the stresses in driver domain. In this section we study the net I/O interference based on these three types of combinations.

Figure 24 presents the net I/O measurements for (1K, 1K), (1K, 100K) and (100K, 100K) on Platform I. These curves are highly correlated with trends of request throughput in Figure 4, based on the fact that (workload size) × (request rate) ≈ Net I/O. Both 1 KB and 100 KB workloads in (1K, 100K) achieve higher maximal net I/O than others under high workload rates, i.e., 1 KB × 1560 req/sec ≈ 1634 KB/sec and 100 KB × 85 req/sec ≈ 8550 KB/sec respectively. It is interesting to note that when the load rate is approximately less than 70%, (100K, 100K) gains better net I/O performance compared to (1K, 100K) combination, while from 30% to 70% load (i.e., the range ***II*** in Figure 24) the net I/O of 100 KB workload in (1K, 100K)

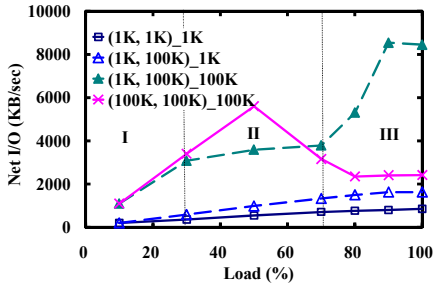


Fig. 24. Net I/O performance on Platform I. (Within range II, (1K, 100K)_100K gets poor performance than (100K, 100K)_100K).

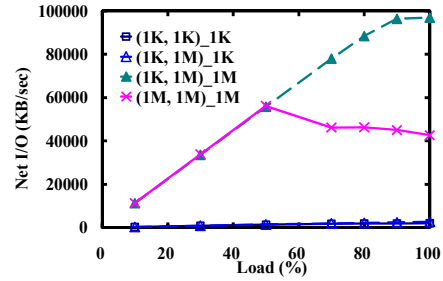


Fig. 25. Net I/O performance on Platform II (1 MB workloads goes more smoothly during the whole test).

combination (i.e., (1K, 100K)_100K) remains flat, around 3600 KB/sec, which is close to the current ***window size*** (i.e., 4 KB) in XenoLinux. It seems that 100 KB workload is "paused" during range ***II***. We can understand this phenomenon better by examining the guest domain details of 100 KB curves presented in Figure 12, Figure 13 and Figure 14. Within the load range 30% to 70%, (1K, 100K)_100K has average 5% higher block time than (100K, 100K)_100K. Meanwhile, the waiting time of (1K, 100K)_100K in Figure 14 (enhanced graph) is keeping around 2%. Thought, the waiting time of (100K, 100K)_100K increases three times from 0.5% to 1.7%, it is still the lowest one. These result in (1K, 100K)_100K utilizes CPU around 4%, while (100K, 100K)_100K consumes CPU continuously until saturating the NIC.

Despite achieving better request throughput and net I/O in the combination of (1K, 100K), we notice that 1 KB workload in this combination gets "priority" treatment while leaving 100 KB workload blocked more often and waiting longer. To understand this phenomenon, it is worthwhile to discuss the details of CPU run queues under ***Credit scheduler***. Under normal circumstances, all vCPUs in the CPU queues are served in the FIFO manner. However, when a VM receives an interrupt while it is idle, the VM enters the particular ***Boost*** state which has a higher priority to be inserted into the head of the run queue for the first CPU execution. This mechanism prevents long waiting time for the latest active VM by preempting the current running VM. However, the even priority shares of CPU usage remains a problem for network-intensive workloads here. Considering (1K, 100K) combination, because of 1 KB file is the shortest size, it could finish each request and enters the idle state faster (most infrequently blocked in Figure 13). Finally, VMM makes the decision to put VM1 in the head of run queue frequently. This makes 1 KB file have higher priority. Thus, the effects of ***Credit scheduler*** should be considered in virtualized cloud environments as it is making positive contributions to CPU-intensive workload while treating network-intensive workload unfairly with higher processing latency, bringing poor I/O performance.

In the experimental setups, we configured Platform II with one CPU run queue with ***Credit scheduler*** turned on. Figure 25 shows the net I/O performance results of three combinations on Platform II, i.e., (1K, 1K), (1K, 1M) and (1M, 1M). Herein 1 MB file is the representative network-intensive workload. It is observed that (1K, 1M) achieves maximum net I/O 1 MB × 97 req/sec ≈ 96900 KB/sec at 100% load and (1M, 1M) achieves maximum net I/O at 50% load with 1 MB × 55 req/sec ≈ 56100 KB/sec. In contrast to the network-intensive workload for Platform I in Figure 24, 1 MB file goes smoothly in both (1K, 1M) and (1M, 1M) during the entire tested range of workload rates. The "paused" net I/O phenomenon in Platform I for network-intensive workload does not occur for Platform II in Figure 25. According to the latest Xen documents, the ***Credit scheduler*** is designed for multi-core processors. With single CPU run queue, the scheduling policy of ***Credit scheduler*** is limited and CPU run queue is easy to get saturation. Under such situation, ***Credit scheduler*** works as the former CPU scheduler BVT [5], leading to a relatively fair CPU execution treatment.

4.3 Multi-core Performance Interference

We dedicate this section to analyze the impact of multi-core and larger L2 cache size on performance interference of co-locating CPU-bound or network-bound workloads in a virtualized cloud environment. First, the experimental setup is introduced. Then we study the performance impact of increasing the number of cores on Platform III in Section 4.3.1 and analyze different strategies for allocating cores to driver domain and guest domains in Section 4.3.2.

Table 4 displays the CPU configurations we used for multi-core interference studies. The left three groups are the total number of CPUs shared by all VMs without dedicated allocation of certain cores to specific VMs. We use these three configurations to further understand the performance interference of (1K, 1K) combination under increasing shared number of cores. The right groups of CPU configurations represent six specific configuration of assigning individual CPUs to each domain. Herein, we use the notation of (***Dom0***, [VM1, VM2]) to denote CPUs pinned on VMs in a specified order of Dom0, VM1 and VM2, with [VM1, VM2] indicating that VM1 and VM2 are sharing same number of CPUs during the measurements. For example, the expression (1, [2, 2]) means that Dom0 is allocated one exclusive CPU, while VM1 and VM2 are configured to share two CPUs. The expression (1, 1, 1) means each domain owns one CPU exclusively.

The ***basecase*** measurement for CPU-bound 1 KB workload on Platform III is 10000 req/sec. In fact, for all measurements, only when Dom0 is configured with one dedicated individual core, the ***basecase*** performance of 1 KB workload is limited by CPU resource at 10000 req/sec, because the achievable maximal request throughput of 1 KB workload for diverse CPU configurations depend on concrete allocations of CPUs to VMs (especially, in driver

TABLE 4
CPU CONFIGURATIONS FOR MULTI-CORE INTERFERENCE

Total CPUs	CPU Sharing (Shared by all VMs)		Total CPUs	Configuration (Dom0, VM1, VM2)
1	1		3	(1, 1, 1)
3	3		3	(1, [2, 2])
4	4		4	(1, [3, 3])
N/A	N/A		4	(2, [2, 2])
N/A	N/A		3	(2, [1, 1])
N/A	N/A		4	(3, [1, 1])

domain). Also guest domains may run into the risk of insufficient TCP/IP ports with higher request rate than 10000 req/sec[1]. We believe that using the request rate of 10000 req/sec is sufficient enough to understand multi-core related performance interference in Table 4. Similarly, the ***basecase*** throughput of network-bound 1 MB workload is 110 req/ses, limited by NIC used in the experiments.

4.3.1 Performance Impact of Varing Shared Cores

This subsection shows performance interference of left three CPU configurations in Table 4, where 1, 3 and 4 cores are shared by all VMs respectively. Obviously, the peak performance points are key entries to understand performance impacts of co-locating workloads in neighbor VMs. We concentrate on analyzing these peak points.

Figure 26 displays the achieved maximal load. To get an overview of performance on all three configurations, the maximal throughputs are converted into the same scales in term of the percentage of maximal load rates in ***basecase***. For (1K, 1K)_1K, 1 KB workload achieves 100% load with 1 shared core, while dropping to maximal loads of 40% and 45% for the cases of 3 cores and 4 cores respectively. In (1K, 1M)_1K, 1 KB workload is also observed 40% decrement for 3 cores and 35% decrement for 4 cores. In contrast, 1 MB workload maintains quit stable request rates in all measured combinations when we vary the number of shared CPUs, which are 100% in (1K, 1M)_1M and 50% in (1M, 1M)_1M.

Figure 27 measures the total CPU utilization under varying CPUs for three combinations of workloads. We observe that only (1K, 1K) and (1K, 1M) sharing 1 core consume more than 94% CPU resource and CPUs are under utilized in all the other cases. More cores do not improve the overall throughput. It is easy to understand that (1M, 1M) demands NIC capacity and thus network I/O becomes the primary resource bottleneck, thus it leads to low CPU utilization. However, with 1 KB workload reaching at least 10000 req/sec in the ***basecase***, one would expect that (1K, 1K)_1K should achieve more than 50% load and (1K, 1M)_1K is supposed to utilize CPU adequately to achieve 100% load. Thus, an immediate question following the observations in Figure 26 and Figure 27 is what the key factors are, which lead to potential bottleneck and interference for CPU-intensive 1 KB workload running concurrently among multiple VMs. Recall the performance interference study on Platform II in the previous sections, comparing with other workload co-location combinations, (1K, 1K) exhibits the lowest I/O efficiency on Platform II and it only has about 3% CPU not utilized. With Platform III offering 3 cores or 4 cores shared across VMs, the combination of (1K, 1K) still exhibits the lowest I/O efficiency and, furthermore, total CPU usages are no more than 57%. This result shows that enhancing CPU capacity by adding more number of cores or increasing on-chip L2 cache size do not necessarily guarantee better performance for running CPU-intensive workloads concurrently among multiple VMs. To be more specific, CPU resource is not the bottleneck for CPU bound 1 KB workload. Furthermore, even though TCP/IP request rate is high with 1KB workloads running in neighboring VMs on the same host, the monitored NIC usages and memory are far from the configured maximal values for neither driver domain nor guest domains. Thus to answer the question of where the potential bottleneck and interference factors are for CPU-intensive 1 KB workload running in multiple VMs concurrently, we next examine closely on the switches (or events) per second as well as CPU utilization and block time of each VM.

Figure 28 and Figure 29 illustrate switches per second in hypervisor and events per second in event channel respectively. These two numbers are normalized, using switches (or events) per second of one shared core as the ***baseline*** reference value for each combination. For the combination of (1M, 1M), the switch and event numbers are quit similar for three CPU configurations with less than 10% discrepancy. Intuitively, the small increments of switches/sec and events/sec are incurred by additional scheduling costs for managing more CPUs. Recall our performance interference analysis on Platform II, the leading interference factor for co-locating network-intensive workloads comes from switch and event costs residing in VMM and driver domain, and the resource bottleneck is the NIC capacity. Important to note is that these observations remain the same for shared multi-core environments. Unfortunately, for the combinations of (1K, 1K) and (1K, 1M), we observe a significant increase of the switch overhead and moderately higher event cost compared to one core case: switch costs are more than 3.7 times; event values are 2.4 times larger. These overhead increments are way beyond the overheads incurred by additional scheduling costs due to managing more cores.

Figure 30 to Figure 33 report CPU utilization and block time in driver domain and guest domains. Figure 30 shows the descending trends of CPU utilization in Dom0 as the number of shared cores increases. This is consistent with the increasing switch and event overheads shown in Figure 28 and Figure 29. Figure 31 shows the increasing block time of Dom0 as the number of cores increases for each workload combination. These figures indicate that, with increasing cores, Dom0 consumes less and less CPU and is less and less frequently getting into CPU run queues. In contrast, Figure 32 and Figure 33 show some opposite trends of CPU usage and block time in guest domains. For (1K, 1K)_1K and (1K, 1M)_1K, as the number of cores increases, CPU usage are increasing gradually and block time are dropping gradually in guest domains. However, for (1K, 1M)_1M and (1M, 1M)_1M, as

[1]We extended TCP ports to 65536. The lower 1024 ports are reserved for Linux OS, so about 64000 ports are remaining. Httperf timeout is 3 sec [13].

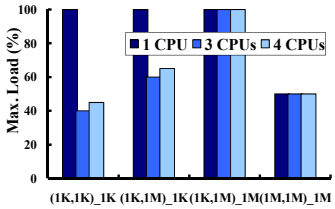


Fig. 26. Achieved maximum load.

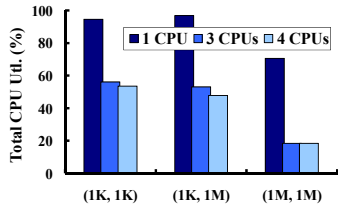


Fig. 27. Total CPU utilization.

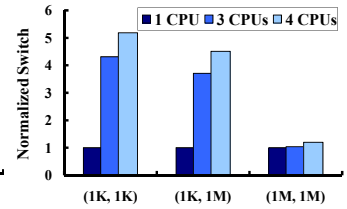


Fig. 28. Normalized switch cost.

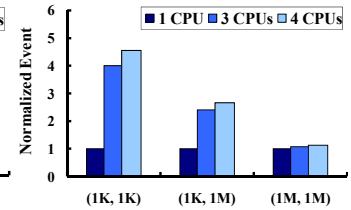


Fig. 29. Normalized event cost.

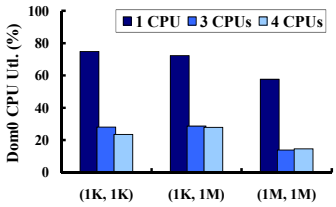


Fig. 30. CPU utilization of *Dom0*.

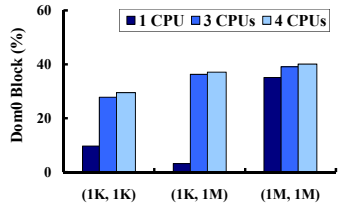


Fig. 31. Block time of *Dom0*.

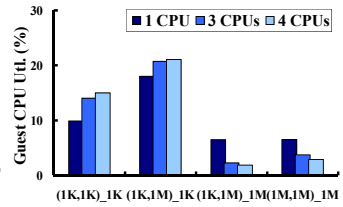


Fig. 32. CPU utilization of guests.

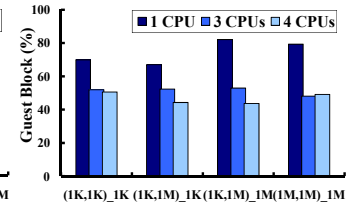


Fig. 33. Block time of guests.

the number of cores increases, CPU utilizations of guest domains are decreasing sharply and the block time of guest domains are dropped by more than 30%. But the difference in CPU utilization and block time of guest domains are very small when the number of cores changes from 3 to 4. Combined with Figure 30 and Figure 31, we can conclude that guest domains are getting much busier than Dom0 when serving CPU-intensive 1 KB workload under multi-core settings. Furthermore, for network-intensive 1MB workload, when the number of cores changes from single to multi cores, CPU utilization also decreases in both Dom0 and guest domains but the block time decreases only in guest domains and increases in Dom0. This implies that there are some other cost factors in addition to multi-core management in guest domains.

To understand other potential interference factors for CPU-intensive workload running on multi-core system, we take the combination of (1K, 1K) running with 3 shared cores as a representative instance below. We investigate some other system-level metrics in these two guest domains, such as memory usage, disk I/O, network details, fragmentations, etc [31], when they are serving 1 KB workload at 40% load rate (i.e., maximal achievable load in Figure 26). During the entire experimental time suggested by httperf, most metrics are varying within an acceptable range, no more than 5% discrepancy. The only exception is the number of SLAB allocators: for each guest domain, allocated SLAB number goes up from 21 to and sticks around 148, almost a factor of five. SLAB allocation [32] is a dedicated memory management mechanism adopted by Linux system to eliminate fragmentations. The number of SLAB allocator indicates cached data objects in guest domains. In practice, this number is associated with grant mechanism in virtualized Xen system.

Recall section 3.1, when a memory page in I/O buffer is granted to driver domain, one free grant entry in grant table is occupied and one new SLAB allocator is created. The increased absolute number of SLAB allocator equals to the number of occupied grant entries in grant table. In (1K, 1K) for 3 shared cores, SLAB allocator gains an increment of 148 - 21 ≈ 128, which means that each guest domain is holding approximate 128 table entries during the experimental period. In the current Xen version, the maximum number of grant entries is co-defined by parameters of ***MAX_MAPTRACK_TO_GRANTS_RATIO*** and ***gnttab_max_nr_frames***, which limit the maximum entries to 128. The purpose of this limitation is to provide a meaningful number of data copies without occupying too much kernel memory. When guest domains are performing high rate I/O processing on shared multi-core host, applications co-locating with their neighbors interfere with each other causing hypervisor to be busy with scheduling and communication. The increased switch and event overheads lead to the delay of the operations of grant invoke and revoke. As a consequence, the grant table easily becomes a bottleneck causing lower I/O count per execution. In our experiments, for all virtual instances serving 1 KB workload (i.e., (1K, 1K)_1K and (1K, 1M)_1K) with all CPU configurations in Table 4, we consistently observed the grant table bottleneck at the peak of throughput. Although tuning the parameters of ***MAX_MAPTRACK_TO_GRANTS_RATIO*** and ***gnttab_max_nr_frames*** may alleviate the size limit of the grant table at the cost of occupying more kernel memory of guest domains. A balance is worth of study in the future.

In short, the experiments in this section reveal that increasing number of shared cores among VMs does not guarantee performance improvement for workloads running in separate VMs on the same physical host:

1. The performance of CPU-intensive workload (i.e., 1 KB) is highly sensitive to the number of cores shared by all VMs. In general, throughput descends gradually when the number of shared cores increasing
2. The performance of network-intensive workload (i.e., 1 MB) is relatively insensitive to the number of cores shared by all VMs.
3. As a result of lower I/O execution, the limitation of grant table entries can be a potential bottleneck for guest domains serving CPU-intensive workload.

4.3.2 Specific CPU Configuration Studies

This section is dedicated to understand the performance impact of specific CPU assignments to each VM, especially, can we decrease potential communication and scheduling overheads in VMM and Dom0 through smart assignment of cores to VMs. We use the right six configurations given in Table 4 to study the effects of varying pinned CPUs on

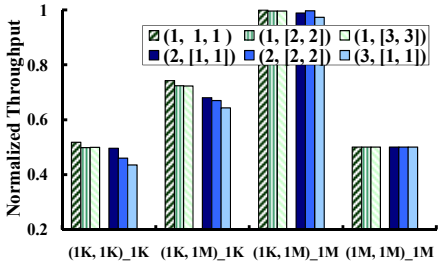


Fig. 34. Normalized throughput for specific CPU configurations in *Dom0* on Platform III.

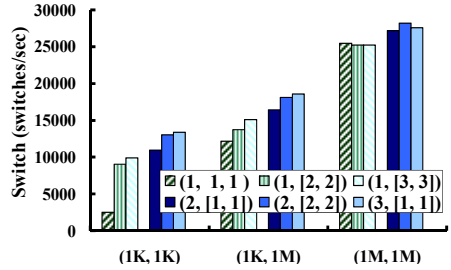


Fig. 35. Switch number per second for specific CPU configurations in *Dom0* on Platform III.

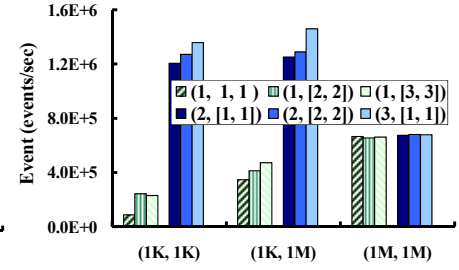


Fig. 36. Event number per second for specific CPU configurations in *Dom0* on Platform III.

each VM by measuring the peak performance in terms of normalized throughput, switch and event respectively for the three representative combinations of workloads.

Figure 34 shows achieved maximal request throughput of each combination under the six CPU configurations. Throughput results are normalized basing on the values of ***basecase***, namely 10000 req/sec for 1 KB workload and 110 req/sec for 1 MB workload. First, the impacts on CPU-bound 1 KB workload are studied. In the group of (1K, 1K)_1K, 1 KB workload gains highest score around 5181 req/sec under (1, 1, 1). With pinned CPUs increasing, throughput of 1 KB workload starts to drop gradually. The worst assignment setting is (3, [1, 1]), which reaches maximal performance around 4351 req/sec. Comparing with (1, 1, 1), it is 16% throughput degradation. In the group of (1K, 1M)_1K, throughput drop is also observed. The best assignment setting is (1, 1, 1) around 7425 req/sec, while the worst is (3, [1, 1]) around 6434 req/sec. Dom0 is a critical place related to interference factors when neighbors interact with each other on the same host. Every I/O data and most messages go through or involve Dom0. If configured with CPUs exclusively, Dom0 serves other VMs more efficiently. In Figure 35 and Figure 36, for 1 KB workload, switch and event numbers grows gradually with throughput drops gradually.

We conclude that for 1 KB workload: 1) less CPUs pinned to Dom0, better performance obtained; 2) given the same CPUs pinned on Dom0, more CPUs assigned to guest domains poorer throughput achieved. Incidentally, grant table bottleneck is also observed in guest domain serving 1 KB workload. SLAB's increment is around 128 as previous case in 4.3.1.

No matter in which workload combination, network-intensive 1 MB workload utilizes NIC adequately and exhibits relative stable performance under all specific CPU assignments: In terms of throughput, we have roughly 110 req/sec in (1K, 1M) and 55 req/sec in (1M, 1M). In terms of response time, (1M, 1M)_1M keeps around 2 ms and (1K, 1M)_1M keeps around 10~15 ms. Switch and event overheads of (1M, 1M) in Figure 35 show tiny difference among different combinations. These results are consistent with the 1 MB workload observed in Section 4.3.1.

We summarize the results from Section 4.3.1 and Section 4.3.2 as follows:

1. Performance of guest domain serving CPU-bound workload (i.e., 1 KB) is highly sensitive to CPU assignment among neighbor VMs on the same host, especially driver domain for three reasons: i) As more CPUs are assigned to the driver domain exclusively, throughput performance suffers more; ii) With the same CPUs pinned on the driver domain, less CPUs will be shared by guest domains, which improves the performance by alleviating switch/event costs; iii) Limited size of grant entries exists as a potential bottleneck for high request rate services.
2. For lower request rate and larger data transfer, the performance of guest domains serving network-bound workload (i.e., 1 MB) is insensitive to the CPU assignments among neighbor VMs on the same host.

In summary, the basis of our measurement interference analysis can be summarized as a three-step process. First, we propose to use eight system-level metrics as outlined in Section 3 to perform the analysis of performance interference and we argue that they are suitable for understanding the communication interference, VMM control interference and I/O processing costs. Second, we choose to use a ***basecase*** on Platform I as a reference and at the same time to illustrate the usage of some VM-specific basic metrics, such as block time and waiting time. Finally, we present a detailed measurement study and analytical discussion on the most important results and observations.

5 Conclusions

We have presented an extensive experimental study of performance interference in running CPU-bound and network-bound workloads on separate VMs hosted by the same physical infrastructure, enabled by Xen Virtual Machine Monitor (VMM). Eight metrics are used to analyze the performance interference among VMs serving net I/O workloads that are either CPU bound or network bound. Based on measurements and observations, we conclude with five key findings which are critical to effective management of virtualized cloud for both cloud service providers and consumers. First, running network-bound workloads in isolated VMs on a shared hardware platform leads to high overheads due to extensive context switches and events in driver domain and VMM. Second, co-locating CPU-bound workloads in isolated VMs on a shared hardware platform incurs high CPU contention due to the demand for fast memory page exchanges in I/O channel. Third, running CPU-bound and network-bound workloads in conjunction incurs the least resource contention, delivering higher aggregate performance. However, default ***Credit scheduler*** treats network-bound workload unfairly under SMP system. Fourth, the

performance of network-bound workload is not sensitive to CPU core assignment among VMs. In contrast, more cores pinned on driver domain delivers worse performance for CPU-bound workload. Finally, due to fast I/O processing in I/O channel, the limited size on grant table can be a potential bottleneck in current Xen system. Identifying factors that impact total demand of exchanged memory pages is critical to in-depth understanding of interference overheads in driver domain and VMM layer.

ACKNOWLEDGMENT

This work is partially supported by grants from NSF CISE NetSE program, CyberTrust program, and grants from IBM faculty award, IBM SUR, and Intel Research Council. The first author conducted this research as a visiting PhD student at Distributed Data Intensive Systems Lab (DiSL), College of Computing, Georgia Institute of Technology, supported by China Scholarship Council (CSC) and School of Computer Science and Technology, Beijing Institute of Technology.

REFERENCES

[1] P. Apparao, S. Makineni, and D. Newell, "Characterization of Network Processing Overheads in Xen," ***Proc. 2nd Int'l Workshop on Virtualization Technology in Distributed Computing (VTDC '06)***, pp. 2-9, Nov. 2006.

[2] P. Barham, B. Dragovic, K. A. Fraser, S. Hand, T. Harris, A. Ho, E. Kotsovinos, A. Madhavapeddy, R. Neugebauer, I. Pratt, and A. Warfield, "Xen 2002," Technical Report UCAM-CL-TR-553, University of Cambridge, Cambridge, Jan. 2003.

[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. the 9th ACM Symp. Operating Systems Principles (SOSP '03)***, pp. 164-177, Oct. 2003.

[4] L. Cherkasova, R.Gardner, "Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor," ***Proc. USENIX Annual Technical Conference (ATC '05)***, pp. 387-390, Apr. 2005.

[5] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the Three CPU Schedulers in Xen," ***ACM SIGMETRICS Performance Evaluation Review (PER '2007)***, pp. 42-51, Sep. 2007.

[6] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson, "Reconstructing I/O," Technical Report UCAM-CL-TR-596, University of Cambridge, Cambridge, Aug. 2004.

[7] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williams, "Safe Hardware Access with the Xen Virtual Machine Monitor," ***Proc. Operating System and Architectural Support for the on demand IT Infrastructure (OASIS'04)***, pp. 1-10, Oct. 2004.

[8] D. Gupta, R. Gardner, and L. Cherkasova, "XenMon: Qos monitoring and performance profiling tool," Technical Report HPL-2005-187, HP Laboratories, Palo Alto, Oct. 2005.

[9] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing Performance Isolation Across Virtual Machines in Xen," ***Proc. ACM/IFIP/USENIX 7th International Middleware Conference (Middleware '06)***, pp. 342-362, Nov. 2006.

[10] T. Hatori and H. Oi, "Implementation and Analysis of Large Receive Offload in a Virtualized System," ***Proc. Virtualization Performance: Analysis, Characterization, and Tools (VPACT '08)***, Apr. 2008.

[11] M. Kallahalla, M. Uysal, R. Swaminathan, D. Lowell, M. Wray, T. Christian, N. Edwards, C. Dalton, and F. Gittler, "SoftUDC: A Software-Based Data Center for Utility Computing," Computer, vol. 37, no. 11, pp. 38-46, Nov. 2004, doi:10.1109/MC.2004.221.

[12] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An Analysis of Performance Interference Effects in Virtual Environments," ***Proc. IEEE Symp. on Performance Analysis of Systems and Software (ISPASS '07)***, pp. 200-209, Apr. 2007.

[13] D. Mosberger and T. Jin, "Httperf—A Tool for Measuring Web Server Performance," ***In Performance Evaluation Review***, vol. 26, no. 3, pp. 31-37, Dec. 1998.

[14] A. Menon, J. Santos, Y. Turner, "Diagnosing Performance Overheads in the Xen Virtual Machine Environment," ***Proc. ACM/USENIX Conf. on Virtual Execution Environments (VEE '05)***, pp. 13-23, Jun. 2005.

[15] A. Menon, A. L. Cox, W. Zwaenepoel, "Optimizing Network Virtualization in Xen," ***Proc. USENIX Annual Technical Conference (ATC '06)***, pp. 15-28, Jun. 2006.

[16] K. Mansley, G. Law, D. Riddoch, G. Barzini, N. Turton, and S. Pope, "Getting 10 Gb/s from Xen: Safe and Fast Device Access from Unprivileged Domains," ***Proc. Euro-Par Conf. on Parallel Processing (Euro-Par)***, pp. 224-233, Dec. 2007.

[17] Y. Mei, L. Liu, X. Pu, and S. Sivathanu, "Performance Measurements and Analysis of Network I/O Applications in Virtualized Cloud," ***Proc. IEEE Int'l Conf. on Cloud Computing (CLOUD '10)***, pp. 59-66, Jul. 2010.

[18] H. Oi and F. Nakajima, "Performance Analysis of Large Receive Offload in a Xen Virtualized System," ***Proc. Computer Engineering and Technology (ICCET '09)***, pp. 475-480, Aug. 2009.

[19] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. Shin, "Performance Evaluation of Vrtualization Technologies for Server Consolidation," Technical Report HPL-2007-59R1, HP Laboratories, Palo Alto, Sep. 2008.

[20] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments," ***Proc. IEEE Int'l Conf. on Cloud Computing (CLOUD '10)***, pp. 51-58, Jul. 2010.

[21] R. Rose, "Survey of System Virtualization Techniques", Oregon State University, Mar. 2004.

[22] K. K. Ram, J. R. Santos, Y. Turner, A. L. Cox, and S. Rixner, "Achieving 10 Gb/s Using Safe and Transparent Network Interface Virtualization," ***Proc. ACM/SIGPLAN/SIGOPS Conf. on Virtual Execution Environments (VEE'09)***, pp. 61–70, Mar. 2009.

[23] K. K. Ram, J. R. Santos, and Y. Turner, "Redesigning Xen's Memory Sharing Mechanism for Safe and Efficient I/O Virtualization," ***Proc. the 2nd Int'l Workshop on I/O Virtualization (WIOV '10)***, pp. 2-9, Mar. 2010.

[24] J. R. Santos, G. J. Janakiraman, and Y. Turner, "Network optimizations for PV guests," ***In 3rd Xen Summit***, Sep. 2006.

[25] J. R. Santos, Y. Turner, G. Janakiraman, and I. Pratt. "Bridging the Gap between Software and Hardware Techniques for I/O Virtualization," ***Proc. USENIX Annual Technical Conference (ATC'08)***, pp. 29-42, Jun. 2008.

[26] VmWare: http://www.vmware.com, 2010

[27] The Xen™ Virtual Machine Moniter: http://www.xen.org, 2010.

[28] Citrix® XenServer: http://www.citrix.com, 2011.

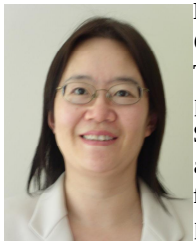
[29] KVM: http://www.linux-kvm.org, 2010

[30] Microsoft® Hyper-V: http://www.microsoft.com/hyper-v-server, 2011

[31] Collectl: http://collectl.sourceforge.net, 2011.

[32] SLAB: http://en.wikipedia.org/wiki/Slab_allocation, 2011.

Xing Pu received the BE degree in the School of Computer Science at Beijing Institute of Technology in 2004. During year 2008-2010, he was a visiting PhD student at Distributed Data Intensive Systems Lab (DiSL) in Georgia Institute of Technology. Currently, he is a PhD student in the School of Computer Science, Beijing Institute of Technology. His main research interests include cloud computing, mobile computing, system virtualization, etc.



Ling Liu is a full Professor in the School of Computer Science at Georgia Institute of Technology. There she directs the research programs in Distributed Data Intensive Systems Lab (DiSL), examining various aspects of data intensive systems with the focus on performance, availability, security, privacy, and energy efficiency. Prof. Liu and her students have released a number of open source software tools, including WebCQ, XWRAPElite, PeerCrawl, GTMobiSim. Prof. Liu has published over 250 International journal and conference articles in the areas of databases, distributed systems, and Internet Computing. She is a recipient of the best paper award of ICDCS 2003, WWW 2004, the 2005 Pat Goldberg Memorial Best Paper Award, and 2008 Int. conf. on Software Engineering and Data Engineering. Prof. Liu has served as general chair and PC chairs of numerous IEEE and ACM conferences in data engineering, distributed computing, service computing and cloud computing fields and is a co-editor-in-chief of the 5 volume Encyclopedia of Database Systems (Springer). She is currently on the editorial board of several international journals, such as Distributed and Parallel Databases (DAPD, Springer), Journal of Parallel and Distributed Computing (JPDC), ACM Transactions on Web, IEEE Transactions on Service Computing (TSC), and Wireless Network (WINET, Springer). Dr. Liu's current research is primarily sponsored by NSF, IBM, and Intel.

Yiduo Mei received the BE degree and the PhD degree in Computer Science and Technology from Xi'an Jiaotong University in 2004 and 2011, respectively. He was a visiting PhD student from 2008-2010 at the Distributed Data intensive Systems Lab (DiSL) in Georgia Institute of Technology, Atlanta, USA. His main research interests include cloud computing, grid computing, system virtualization, performance optimization and trust management in distributed and virtualized systems. He is currently a postdoc in the China Center for Industrial Security Research, where he works on systems and applications security and related research.

Sankaran Sivathanu is a member of the performance team at VMware Inc., Palo Alto, where he works on various storage performance issues in virtualized system stack. He received his PhD in Computer Science at the College of Computing in

Georgia Institute of Technology in 2011. Prior to that he obtained his Master's degree in Computer Science at Georgia Tech and Bachelor's degree in Anna University, India in 2007. His research interests include end-to-end I/O performance issues in large-scale virtualized deployments, workload characterization of emerging applications, faithful workload reproduction techniques, etc.

Younggyun Koh received his PhD degree from the School of Computer Science in the College of Computing at Georgia Institute of Technology in summer, 2010. His research interest lies in cloud computing, system virtualization, system performance optimization. After graduating from Georgia Tech, he joined Google, USA.

Calton Pu received his PhD from University of Washington in 1986 and he is holding the position of Professor and John P. Imlay, Jr. Chair in Software in the School of Computer Science, Georgia Institute of Technology. He is interested in and has published in operating systems, transaction processing, software dependability and information security, Internet data management, and service computing. He is currently working on two major projects: 1) automated system management in cloud environments; 2) document/information quality analysis, including spam and wiki vandalism detection. In addition, he is working on automated workflow, information integration, cyber physical systems, and service computing. He has published more than 70 journal papers and book chapters, 200 refereed conference and workshop papers. He served on more than 120 program committees, including PC chairs or co-chairs (7 times) and general chairs or co-general chairs (8 times), and conference steering committees. His research has been supported by NSF, DARPA, ONR, AFOSR, NIH, and industry grants from ATT, Fujitsu, HP, IBM, Intel, Wipro, among others.

Yuanda Cao is a full Professor in the School of Computer Science and the School of Software at Beijing Institute of Technology. He received BE degree from Beijing Institute of Technology in 1967. His main research interests include artificial intelligence, distributed computing, grid computing, etc.