Performance Analysis of Network I/O Workloads in Virtualized Data Centers

Yiduo Mei, Ling Liu, *Senior Member*, *IEEE*, Xing Pu, Sankaran Sivathanu, and Xiaoshe Dong

Abstract—Server consolidation and application consolidation through virtualization are key performance optimizations in cloud-based service delivery industry. In this paper, we argue that it is important for both cloud consumers and cloud providers to understand the various factors that may have significant impact on the performance of applications running in a virtualized cloud. This paper presents an extensive performance study of network I/O workloads in a virtualized cloud environment. We first show that current implementation of virtual machine monitor (VMM) does not provide sufficient performance isolation to guarantee the effectiveness of resource sharing across multiple virtual machine instances (VMs) running on a single physical host machine, especially when applications running on neighboring VMs are competing for computing and communication resources. Then we study a set of representative workloads in cloud-based data centers, which compete for either CPU or network I/O resources, and present the detailed analysis on different factors that can impact the throughput performance and resource sharing effectiveness. For example, we analyze the cost and the benefit of running idle VM instances on a physical host where some applications are hosted concurrently. We also present an in-depth discussion on the performance impact of colocating applications that compete for either CPU or network I/O resources. Finally, we analyze the impact of different CPU resource scheduling strategies and different workload rates on the performance of applications running on different VMs hosted by the same physical machine.

Index Terms—Cloud computing, performance measurement, virtualization, resource scheduling

1 Introduction

We view a virtualized cloud computing environment as a virtualized data center with a network of physical computing nodes (machines), and each physical machine is hosting multiple virtual machines (VMs). The cloud service providers offer the infrastructure as a service (IaaS) by allowing cloud consumers to reserve virtual machines of their desired capacity and pay only when the applications are actually consuming resources. The promise of virtual machine monitors for server consolidation and application consolidation is to run multiple services on a single physical machine (host) while allowing independent configuration of operating systems (OSs), software, and device drivers on each VM. By allowing multiplexing hardware resources among virtual machines running commodity OSs, virtualization helps to achieve greater system utilization, while lowering total cost of ownership.

However, several studies have documented that current implementation of VMMs does not provide sufficient performance isolation to guarantee the effectiveness of resource sharing, especially when the applications running on multiple VMs of the same physical machine are competing for computing and communication resources [2], [19], [23], [36], [37]. As a result, both cloud consumers and cloud providers may suffer from unexpected performance degradation in terms of efficiency and effectiveness of application execution or server consolidation.

We argue that in-depth performance analysis of different applications running on multiple VMs hosted by the same physical machine is an important step toward maximizing the benefit and effectiveness of server consolidation and application consolidation in virtualized data centers. Quantitative and qualitative analysis of performance bottlenecks that are specific to virtualized environments also provide deeper insights on the key factors for effective resource sharing among applications running in virtualized cloud environments.

In this paper, we present our performance analysis of network I/O workloads hosted in different VMs of a single physical machine. We focus on workloads that are either CPU bound or network I/O bound, because network intensive applications are known to be the dominating workloads in pay-as-you-go cloud-based data centers, exemplified by Amazon EC2, Google AppEngine, and Saleforce.com. We report our measurement study on three categories of resource scheduling problems. First, we study the impact of running idle guest domains on the overall system performance, addressing the cost, and benefit of managing idle VM instances in virtualized data centers. When a domain is said to be idle, it means that there is no other runnable processes and the OS is executing idle-loop.

- *Y. Mei is with the China Center for Industrial Security Research, Beijing Jiaotong University, Beijing, P.R. China. E-mail: meiyiduo@gmail.com.*
- *L. Liu and S. Sivathanu are with the College of Computing, Georgia Institute of Technology, 266 Ferst Drive, Atlanta, GA 30332-0765. E-mail: {lingliu, sankaran}@cc.gatech.edu.*
- *P. Xing is with State Radio Regulation of China, Beijing, P.R. China. E-mail: abenpu@gmail.com.*
- *X. Dong is with the Department of Computer Science and Technology, School of Electronic and Information Engineering, Xi'an Jiaotong University, 28 XianNing West Road, Xi'an 710049, China. E-mail: xsdong@mail.xjtu.edu.cn.*

Manuscript received 2 Nov. 2010; revised 29 Mar. 2011; accepted 4 May 2011; published online 14 June 2011.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSC-2010-11-0132.
Digital Object Identifier no. 10.1109/TSC.2011.36.

We conjecture that this experimental study not only helps cloud service providers to effectively manage virtual machines to better meet consumers' demands, but also provides useful insights for cloud consumers to manage idle instances more effectively for seamlessly scaling out their applications on demand. Second, we conduct an in-depth measurement analysis of concurrent network I/O applications colocated in a virtualized cloud in terms of throughput performance and resource sharing effectiveness. We focus on the set of key factors that can maximize the physical host capacity and the performance of independent applications running on individual VMs. Finally, we study how different CPU resource sharing strategies among VMs hosted on a single physical machine under different workload rates may impact the overall performance of a virtualized system. Our experimental study shows that quantifying the performance gains and losses relative to different VM configurations provides valuable insights for both cloud service providers and cloud consumers to better manage their virtualized infrastructure and to discover more opportunities for performance optimization of their applications. Our measurement analysis also reveals that applications running in virtualized data centers should be managed more carefully to minimize unexpected performance degradation and maximize desired performance gains.

The rest of the paper is organized as follows: We briefly review the related work in Section 2. Section 3 gives an overview of the basic methodology and measurement metrics used in this study. Section 4 presents our performance analysis on the cost and benefit of managing idle VM instances. Section 5 describes our measurement study of colocating applications that compete for either CPU or network I/O resources in a virtualized cloud. Section 6 studies the impact of tuning CPU credit scheduler on application performance. We conclude the paper in Section 7 with a summary and an outline of our ongoing research.

2 Related Work

A fair number of research and development efforts have been dedicated to the enhancement of virtualization technology in the past few years. Most of the efforts to date can be classified into three main categories: 1) performance monitoring and enhancement of VMs hosted on a single physical machine [5], [6], [13], [17], [21], [25], [26], [32]; 2) performance evaluation, enhancement, and migration of VMs running on multiple physical hosts [8], [16], [30], [34], [35], [41]; 3) performance comparison conducted with different platforms or different implementations of VMMs [10], [22], [40], such as Xen [3] and KVM [20], as well as the efforts on developing benchmarks [1], [22]. Given that the focus of this paper is on performance measurement and analysis of network I/O applications in a virtualized single host, in this section, we provide a brief discussion on the state of art in literature to date on this topic. Most of the research on virtualization in a single host has been focused on either developing the performance monitoring or profiling tools for VMM and VMs, represented by [14], [15], or conducting performance evaluation work by varying VM configurations on host capacity utilization or by varying CPU scheduler configurations [6], [12], [18], [21], [28], especially for I/O related performance measurements [3], [5], [7], [23]. For example, some work has focused on I/O performance improvement by tuning I/O related parameter [11], [13], [17], [25], [26], [32], [39], such as TCP Segmentation Offload (TSO), network bridging. Recently, some study showed that performance interference exists among multiple virtual machines running on the same physical host due to the shared use of computing resources [5], [23], [36], [37] and the implicit resource scheduling of different virtual machines done by VMM in the privileged driver domain [19]. For example, in current Xen implementation, all the I/O requests have to be processed by the driver domain, and Xen does not explicitly differentiate the Dom_0 CPU usage caused by I/O operations for each guest domain. The lacking of mechanism for Domain_0 (Dom_0) to explicitly separate its usage for different VMs is, to some degree, contributing to the unpredictable performance interference among guest domains (VMs) [14]. However, none of the existing work has provided in-depth performance analysis in the context of performance interference and application colocations.

To the best of our knowledge, this paper is the first one that provides an in-depth performance measurement and analysis on a number of important issues, including the cost and benefits of running idle VM instances concurrently with some applications in a virtualized cloud, the impact of different CPU resource scheduling strategies on the performance of network I/O applications under varying workload rates, and the impact of resource competition and performance interference on effectiveness of colocating applications. The measurement study presented in this paper offers new insights on the set of factors important for efficient scheduling and tuning of network I/O applications. Although this paper reports our measurement analysis on the open source version of Xen VMM and EC2 style cloud platform, we believe that the results and observations obtained by this study are instructmental to evaluate other implementations of VMM such as KVM [20] and VMware [38].

3 Overview

In this section we first briefly review Xen [3], especially some features and implementation details of Xen, which are important backgrounds for our performance analysis and measurement study. Then we briefly outline our basic methodology for conducting performance measurement and analysis of network I/O applications in virtualized cloud environments.

3.1 Xen I/O Mechanism

Xen is an x86 VMM (hypervisor) developed based on paravirtualization. VMM interfaces between the virtual machine tier and the underlying physical machine resources. At boot time, an initial domain, called Dom_0, is created and serves as the privileged management domain, which uses the control interface to create and terminate other unprivileged domains, called guest domains, and manages the CPU scheduling parameters and resource allocation policies.

In Xen, Dom_0 also serves as a driver domain by containing: 1) unmodified Linux drivers for I/O devices, 2) network bridge for forwarding packets to guest domains, and 3) *netback* interface to each guest domain. Devices can be shared among guest operating systems running in guest domains, denoted as $\mathrm{Dom}_1, \mathrm{Dom}_2, \ldots, \mathrm{Dom}_n (n > 1)$. A guest domain implements a virtual network interface controller driver called *netfront* to communicate with corresponding *netback* driver in Dom_0. Xen processes the network I/O requests through the event channel mechanism and the page flipping technique. For example, consider the guest domain which is receiving a network packet, whenever a network packet arrives, and the hardware raises an interrupt. The hypervisor intercepts the interrupt and then initializes a virtual interrupt through the event channel to inform the driver domain of the arrival of the packet. When the driver domain is scheduled to run, it sees the I/O event notification. The device driver in the driver domain fetches the packet and delivers it to the network bridge. The network bridge inspects the header of the packet to determine which *netback* to send to. After the network packet is put into proper *netback* driver, the network driver notifies the destination guest domain with a virtual interrupt through the event channel, and it encapsulates the network packet data into the form of memory pages. Next time when the guest domain is scheduled to run, the guest domain sees the notification. Then the memory page containing the network packet data in the *netback* driver is exchanged with an unused page provided by the destination guest OS through the network I/O channel. This process is called memory page flipping, which is designed to reduce the overhead caused by copying I/O data across domains (VMs). The procedure is reversed for sending packets from the guest domains via the driver domain. Previous studies also showed the grant mechanism as a significant source of network I/O overhead in Xen [3], [5], [12], [18], [29], [32], [33].

3.2 Credit Scheduler

Xen employs the credit scheduler to facilitate load balancing on symmetric multiprocessing (SMP) host. The nonzero cap parameter specifies the maximum percentage of CPU resources that a virtual machine can get. The weight parameter determines the credit associated with the VM. The scheduler will charge the running virtual CPU (VCPU) 100 credits every tick cycle, which is 10 milliseconds (ms) by default. According to the remaining amount of credits of each VCPU, its states can be: under (-1) and over (-2). If the credit is no less than zero, then the VCPU is in the under state, otherwise, it's in the over state. Each physical CPU checks VCPUs in the following steps before it goes into idle: first, it checks its running queue to find out the ready VCPU which is in the under state, then it will check other physical CPU's running queue to fetch VCPU that is in the under state. After that the scheduler will execute the VCPU in the over state in its own running queue from beginning. It will never go to idle states before it finally checks other physical CPU's running queue to see whether there exists runnable VCPU in the over state.

A VCPU can be scheduled to run on physical CPU for 30 ms by default before preemption as long as it has sufficient credit. To alleviate the high I/O response latency, the credit scheduler introduces the boost state to prompt the I/O processing priority. An idle domain can enter the boost state when it receives a notification over the event channel and it is previously in the under state, resulting in high scheduling priority [6], [9], [18], [29].

TABLE 1
Experimental Platform Setup

Specification	Platform I	Platform II	Client
CPU Family	**Xeon(TM)**	**Pentium(R) 4**	**Core(TM) 2**
# of Processor	2	1	1
Core	1	1	2
Frequency (GHz)	1.7	2.6	2.66
L2 Cache (KB)	256	512	4096
FSB (MHz)	400	533	1333
RAM (MB)	1024	512	2048
DISK (GB)	20	40	250
NIC (Mbit/sec)	100	1024	1024
OS	Ubuntu 8.0.4	Ubuntu 8.0.4	Debian 5.0
Xen Hypervisor	3.2	3.2	NA

3.3 Objectives and Experimental Setup

This section outlines the objectives of this measurement study and the experimental setup, including the representative network I/O workloads, and the virtual machine configuration.

In a virtualized cloud environment, cloud providers implement server consolidation by slicing each physical machine into multiple virtual machines based on server capacity provisioning demands. Cloud consumers may reserve computing resources through renting VMs from cloud providers. However, there is lacking of in-depth study on performance implications of running applications on multiple VMs hosted by the same physical machine.

In this paper, we design our measurement study, focusing on analyzing the following three important issues: 1) understanding the cost and benefit of maintaining idle VMs on application performance; 2) understanding the performance implication of colocating CPU bound and network I/O bound applications over separate VMs running on a shared physical host in a virtualized cloud, especially in terms of throughput performance and resource sharing effectiveness; and 3) understanding how different CPU resource sharing strategies among guest domains hosted on a physical machine may impact the overall system performance. By conducting an in-depth measurement analysis of these issues, we will gain better understanding of the key factors that can maximize the physical host capacity and the application performance. Furthermore, cloud service providers can provide more effective management of virtual machines to better meet consumers' demand. Cloud service consumers can learn from the insights of this study to better manage and scale their applications.

All experiments were conducted using two slightly different platforms to ensure the consistency of experimental results (see Table 1). The Platform I is a DELL Precision Workstation 530 MT with dual 1.7 GHz Intel Xeon processors, 1 GB ECC ram, Maxtor 20 GB 7200 RPM IDE disk, and 100 Mbps network connection. Platform II is Pentium 4-based server with one single 2.6 GHz processor, 512 MB ram, 40 GB IDE disk, and 1 Gbps network connection. We installed on both of the platforms the Ubuntu 8.0.4 distribution and Xen 3.2 with the default credit scheduler. Each of the two physical machines hosts multiple virtual machines. Each VM is running Apache web server to process web requests from remote clients. Each client generates file retrieval requests for a particular virtual machine such that the clients will not become the bottlenecks in our experiments. Each connection issues one file request by default. A control node coordinates individual clients and collects profiling data. The industry standard benchmarks for evaluating web server performance are the SPECweb '96, SPECweb '99, SPECweb '06, and SPECweb '09. The web server performance is measured as a maximum achievable number of connections per second when retrieving files of various sizes. We use httperf [27] to send client requests for web documents of size 1, 10, 30, 50, 70, 100, or 200 kB hosted in the Apache web servers running in guest VMs. In all experiments, we aim at measuring virtual server performance, thus the client machines we used are sufficiently powerful to ensure that the clients do not incur any performance bottleneck.

Finally, we would like to note the importance of using two slightly different hardware platforms to ensure the consistency of the measurement study. It is known that web server performance is CPU bound under a mix of small size files, and is network bound under a mix of large files [5] [31]. However, the criteria for small or large files depend primarily on the capacity of the machine used in the experiments. For our experimental setup, files with size larger than 10 kB are considered long file in Platform I since applications with file size of 30, 50, and 70 kB will saturate the network interface card of Platform I, but we had to use the files of size 100 or 200 kB to stress the network interface card of Platform II. This is because the network interface card of Platform II has much bigger capacity than the NIC in Platform I, though the physical server of Platform I is dual processors with bigger RAM. This further demonstrates the importance of employing two alternative platforms to conduct our measurement study. Concretely, all the experiments conducted on one platform are repeated on the other platform. We show that the impact of hardware capacity difference between the two physical servers (platforms) may have on our understanding and analysis of the cloud performance, though it is also possible that such impact is insignificant at times. For example, on both platforms, each idle guest domain can get about 250 microseconds (μs) for each run as reported in Section 4.1. However, for Platform I, the peak throughput for the 1 kB application is around 1,100 req/sec, while the peak throughput is reduced to about 980 req/sec for platform II.

3.4 Performance Metrics

In this section, we present the core set of metrics we use to conduct the performance analysis and measurements reported in this paper.

Different types of resource contention occur in a virtualized environment. Such contentions often are due to varying network I/O workload sizes being requested, the variation of applied workload rates, the varying number of VMs (guest domains), and the types of applications that are colocated in different VMs hosted on a single physical machine. Thus, in order to analyze the web server performance in a virtualized environment, the following metrics are used in our measurement study. They are collected using Xenmon [15] and Xentop monitoring tools.

Server throughput (req/sec). One way to measure web server performance is to measure the server throughput in each VM at different workload rates, namely the maximum number of successful requests served per second when retrieving web documents.

Normalized throughput. In order to compare throughputs of different VM configurations and different number of VMs hosted on a single physical machine, we typically choose one measured throughput as our baseline reference throughput and normalize the throughputs of different configuration settings by using the ratio of the throughput in a given setting over the baseline reference throughput in order to make adequate comparison.

Aggregated throughput (req/sec). We use aggregated throughput as a metric to compare and measure the impact of using different number of VMs on the aggregated throughput performance of a physical host. This metric also helps us to understand other factors that may influence the aggregated performance.

Reply time (millisecond). This metric measures the time elapsed between the client sent the first byte of the request and received the first byte of the reply.

Transfer time (millisecond). Different from the reply time, the transfer time measures the time took for the client to receive the entire reply.

CPU time per execution (μs/exe). The CPU time per execution (μs/exe) is a performance indicator that shows the average obtained CPU time in microseconds (μs) during each running of the domain.

Execution per second (exe/sec). The execution per second metric measures the counts of domain for being scheduled to run on a physical CPU over the unit time duration.

CPU utilization (percent). To understand the CPU resource sharing across VMs running on a single physical machine, we measure the average CPU utilization of each VM, including Dom_0 and guest domain CPU usage, respectively. The summation of all VM CPU utilizations represents the total CPU consumption.

Net I/O per second (kB/sec). We measure the amount of network I/O traffic in kB per second, transferred to/from a web server during the corresponding workload.

Memory pages exchange per second (pages/sec). We measure the number of exchanged memory pages per second in I/O channel. This metric indicates how efficient the I/O processing is.

Memory pages exchange per execution (pages/exe). This metric is a performance indicator that shows the average memory pages exchange during each running of the domain.

TABLE 2
Maximum throughput for Dom_1 on Platform I [Req/Sec]

App.	(# of guest domains, # of idle domains)				Worst Degradation
	(1,0)	(2,1)	(3,2)	(4,3)	
1kB	1070	1067	1040	999	6.6%
10kB	720	717	714	711	1.3%
30kB	380	380	380	380	0
50kB	230	230	230	230	0
70kB	165	165	165	165	0

4 Running Idle Instances: Impact Analysis

In this section, we provide a detailed performance analysis of maintaining idle VM instances, focusing on the cost and benefit of maintaining idle guest domains in the presence of network I/O workloads on a separate VM sharing the same physical host. Concretely, we focus our measurement study on addressing the following two questions: first, we want to understand the advantages and drawbacks of keeping idle instances from the perspectives of both cloud providers and cloud consumers. Second, we want to measure and understand the start-up time of creating one or more new guest domains on a physical host, and its impact on existing applications.

Consider a set of n $(n > 0)$ VMs hosted on a physical machine, at any given point of time, a guest domain (VM) can be in one of the following three states: 1) *execution state*, namely the guest domain is currently using CPU; 2) *runnable state*, namely the guest domain is on the run queue, waiting to be scheduled for execution on the CPU; and 3) *blocked state*, namely the guest domain is blocked and is not on the run queue. A guest domain is called idle when the guest OS is executing idle-loop.

4.1 Cost of Maintaining Idle Guest Domains

Both cloud providers and cloud consumers are interested in understanding the cost and benefit of maintaining idle guest domains in comparison to acquiring guest domains on demand. We attempt to answer this question by conducting a series of experiments. First, we use our test Platform I, on which we started one single guest domain, denoted as Dom_1. Dom_1 serves all http requests. We stress Dom_1 with as high workload as possible to find out its service limit. Then, we started the next three sets of experiments. We setup m guest domains on the Platform I machine with Dom_1 serving http requests and the remaining $m - 1$ guest domains being idle where $m = 2, 3, 4$. In order to measure and compare the cost of starting up an application on an idle domain versus the cost of acquiring a new guest domain from scratch, we start the Apache web server automatically in an idle domain to simulate the situation that an instance which has been booted up can respond to http requests immediately.

Table 2 shows the results of the four sets of experiments. Dom_1 is running I/O application in high workload rate, with zero, one, two, or three other VMs in idle execution state. Each of the four sets of experiments records the maximum achievable throughputs for the five different network I/O applications (1, 10, 30, 50, and 70 kB). We observe a number of interesting facts from Table 2. First, there is no visible performance penalty in terms of the cost of keeping the idle domain(s) running when Dom_1, the running domain, is serving 30, 50, and 70 kB applications, because these applications are network bounded on Platform I. Second, the 1 kB application, in contrast, shows the worst performance degradation and is CPU bound. Compared with the single VM case where no idle domain is maintained, the highest throughput value achieved is 1,070 req/sec, and we see 6.6 percent performance degradation $(1 - 999/1{,}070)$ when the number of guest domains is four (999 req/sec). From these four sets of experiments, we observe that adding idle guest domains can impact the performance of CPU intensive applications in the running domain. Such overhead increases as the number of idle VMs grows.

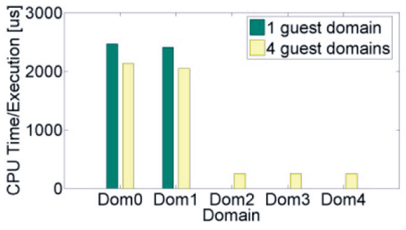

Fig. 1. CPU time per execution [μs/exe] for 1 kB application with 0 and 3 idle guest domains on Platform I.

We conduct more detailed measurements on the performance impact of maintaining idle instances in order to quantitatively characterize the overhead occurred for 1 kB application running on Platform I. Figs. 1 and 2 present the results. We run 1 kB application in Dom_1 under two scenarios, and measured the *CPU time per execution* and the *number of execution counts per second*, respectively. The first scenario is one VM with no idle domain and the second scenario is four VMs with three idle. We make two observations from Figs. 1 and 2. First, each of the three idle guest domains on average can get about 250 μs CPU time per execution, which is about 10 percent of the CPU time of Dom_0 for each execution. Second, comparing the scenario of 4 VMs with 3 idle with the single VM scenario, we see that the CPU time for each execution is dropped from 2,464 μs to 2,130 μsin Dom_0 and from 2,407 to 2,046 μs in Dom_1 (Fig. 1). Similarly, the execution count per second is dropped from 400,000 to 300,000 in Dom_1, though the execution count per second in Dom_0 shows a slight increase (Fig. 2). The drop in CPU time per execution and the number of executions per second is primarily due to two factors: 1) the overhead incurred by the execution of timer tick for the idle guest domains and the overhead of

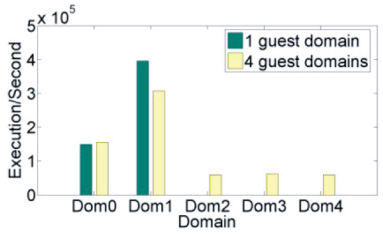

Fig. 2. Execution counts per second for 1 kB application with 0 and 3 idle guest domains on Platform I.

TABLE 3
Performance Degradation on Platform II

Req Rate (req/sec) w. r. t. [App.]	Throughput		Thr. Deg.	Reply Time milliseconds		RT Deg	Transfer Time milliseconds		TT Deg
	(1,0)	(4,3)		(1,0)	(4,3)		(1,0)	(4,3)	
980 [1k]	958	927	3.2%	53	77	45%	0	0	0
900 [1k]	897	887	1.1%	28	51	82%	0	0	0
500 [1k]	500	500	0	0.9	0.91	1.1%	0	0	0
200 [1k]	200	200	0	0.9	0.9	0	0	0	0
200[50kB]	200	200	0	0.9	0.9	0	2.2	2.2	0
200[100kB]	200	200	0	2.0	4.5	125%	5.8	11.9	105%
200[200kB]	200	200	0	3.6	10.9	200%	9.4	30.0	219%
100[200kB]	100	100	0	0.9	1.0	11.1%	5.1	5.2	2%
50[200kB]	50	50	0	0.9	1.0	11.1%	4.9	5.1	4%

associated context switch, and 2) the overhead of processing network packets, such as address resolution protocol (ARP) packets, which causes I/O processing in guest domain.

To ensure the consistency of our observations from experiments with Platform I physical machine, we conducted similar experiments on Platform II physical machine to gain more insights on impact of running idle guest domains. Similarly, we consider the two scenarios: 1 VM and 4 VM with 3 idle. In order to learn about other possible factors that may impact the performance of idle domains, in addition to throughput and throughput degration, we also measure reply time, transfer time, and their degradations, by varying the workload rates for the 1 kB application. Table 2 confirms that keeping idle guest domains may lead to some degree of performance degradation and such degradation differs from network I/O bounded applications to CPU bounded applications. In addition, there are multiple factors that influence the degree of performance degradation. We observe from Table 3 that when we run single application on a shared cloud hardware platform with throughput as the performance metric for the CPU bounded application, workload rate determines the extent of the throughput degradation. When the workload rate is at 900 or 980 req/sec, we experience some throughput degradation from 1.1 percent to 3.2 percent. However, changing workload rates has neither explicit impact on the throughput performance for network I/O bounded applications (e.g., 30, 50, or 70 kB) nor performance impact on CPU bounded applications (e.g., 1 kB) serving at lower workload rates (e.g., 200 or 500 req/sec). But if we consider the reply time or the transfer time, we could observe some obvious performance degradation in terms of the reply time and transfer time for applications of larger file sizes (e.g., 100, 200 kB) running on Platform II. Also the RT degradation column in Table 3 shows that 100 and 200 kB applications suffer more serious performance degradation at the high workload rate of 200 req/sec. Similar observations can be found with respect to the transfer time degradation (TT Deg column). Given that the reply for 1 kB application was less than one millisecond, too short to be measured at μs unit, and thus transfer time was zero.

4.2 In-Depth Performance Analysis

As shown in the previous section, there are more factors that impact the cost and benefit of maintaining idle guest domains in addition to throughput. In this section, we will provide an in-depth measurement on the various performance metrics in order to better manage the idle guest domains.

Figs. 3 and 4 measure the switches, blocks, or waits for five scenarios on Platform II: they are 1 kB at 200 req/sec, 1 kB at 500 req/sec, 1 kB at 900 req/sec, 50 kB at 200 req/sec, and 100 kB at 200 req/sec. Fig. 3 shows when Dom_1 is serving the CPU bound 1 kB application, the context switches with running 0 and 3 idle guest domains are slightly different for all five scenarios on Platform II (1 kB at 200 req/sec, 1 kB at 500 req/sec, 1 kB at 900 req/sec, 50 kB at 200 req/sec, and 100 kB at 200 req/sec). Take the 1 kB with 200 req/sec as an example, when running Dom_1 alone, Xenmon [15] recorded only 872,614 switches, compared with 990,169 switches occurred when running 3 idle guest domains. Fig. 4 show similar results when we considered

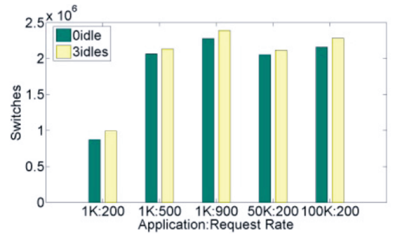

Fig. 3. Switches with different configurations on Platform II.

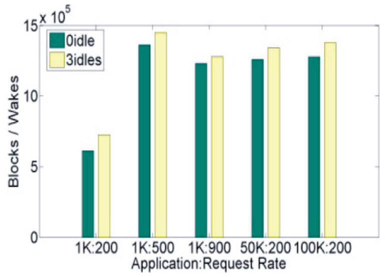

Fig. 4. Blocks/Wakes differences between tests running with 0 and 3 idle guest domains on Platform II.

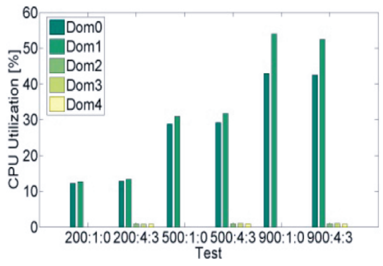

Fig. 5. CPU utilization [percent] for 1 kB application under 200, 500, and 900 workload rates with 0 and 3 idle guest domains on Platform II.

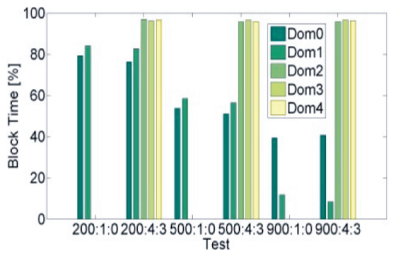

Fig. 6. Block time [percent] for 1 kB application under 200, 500, and 900 workload rates with 0 and 3 idle guest domains on Platform II.

the Blocks or Wakes as metrics. As we increase the idle guest domains from 0 to 3, there is an increase in terms of switches, blocks, or wakes due to the introduction of the idle guest domains which are executing idle loops.

Fig. 5 shows CPU utilization for 1 kB application with varying workload rate and varying number of idle guest domains. Two observations are made from Fig. 5. First, each of the idle guest domains takes about 1 percent CPU resource on Platform II. Second, when Dom_1 is serving low and medium workload rate (e.g., 200 req/sec and 500 req/sec), Dom_1 consumes a slightly more CPU than Dom_0. However, the demand for CPU resource by Dom_1 grows as the workload rate increases. At 900 req/sec, the Dom_1 CPU capacity is almost saturated. It is observed that the CPU share for Dom_1 drops from 55 to 52 percent when we switched the number of idle guest domains from 0 to 3 with the workload rate at 900 req/sec. This further explains the performance degradation in terms of reply time and transfer time shown in Table 3.

Fig. 6 measures the percentage of block time for 1 kB application with varying workload rate and varying number of idle guest domains. Table 4 lists the same result in concrete values. From this table and Fig. 6, we observe that Dom_1 block time drops dramatically compared with Dom_0 at the workload rate of 900 req/sec regardless whether the 1 kB application is running on Dom_1 with zero idle instance or three idle instances. This observation appears to be counter intuitive since with the same workload and the same application (1 kB) one would think that both Dom_0 and Dom_1 should get similar block time because they have almost the same amounts of network packets to process. By taking a closer look at the experimental results, we see that two important facts. First, at high workload rate of 900 req/sec and with the default behavior of the credit scheduler (which is trying to equally share CPU resources among multiple working domains), Dom_0 works efficiently to respond to Dom_1's I/O events. Thus, Dom_1 needs more CPU time to quickly respond to I/O event issued from Dom_0, leading to low percentage of block time at Dom_1. Second, at 900 req/sec rate, the CPU share for Dom_1 from 55 to 52 percent when the number of idle guest domains is changed from 0 to 3. We see that the highest block time is for the idle guest domains, and also comparing to the block time of Dom_1 with no idle instance, the block time dropped more for Dom_1 with 3 idle instances running next door. We can illustrate the block time differences in two steps. First, our experiments show that processing a 1 kB file request requires to transfer at least 5 TCP/IP network packets between the client and a web server. Thus, when Dom_1 and Dom_0 are working hard at each CPU cycle in order to cope with the 900 req/sec workload rate, the idle domains are consequently blocked longer because idle domainshave to wait until Dom_0 either finishes all the other packets that have higher priorities or finishes its current CPU cycle before being scheduled by the local I/O scheduler in Dom_0. The more idle instances are kept, the higher block time Dom_1 will experience as a consequence for CPU intensive workloads.

In addition to the block time, the wait time, and the CPU utilization are also important performance measures for understanding the cost of maintaining idle instances. Fig. 7 shows the waiting time for 1 kB application under 200, 500, and 900 workload rates. We have two observations: 1) the waiting time is very low for all idle guest domains at all workload rates for the two configuration scenarios, because once the I/O event reaches the idle guest domain, the idle guest domain is promoted into the BOOST status, then it can get to the head of running queue. 2) Compared with 1 VM with no idle instance, when we run the configuration of 4 VMs with 3 idle instances, the waiting time for Dom_1 is always higher at the medium or high workload rate (500 or 900 req/sec). This is because with high workload rate, Dom_1 is overcharged with its default credit. Thus, the credit scheduler will mark it as OVER state, then it will be put to the end of the

TABLE 4
Block Time for 1 KB Application with Varying Workload Rates

Domain	Request Rate (REQ/SEC) (# of guest domains, # of idle domains)					
	200		500		900	
	(1,0)	(4,3)	(1,0)	(4,3)	(1,0)	(4,3)
Dom_0	79.2	76.2	53.8	51.0	40.7	39.3
Dom_1	84.0	82.6	58.5	56.4	11.8	8.4
Dom_2	—	96.8	—	95.7	—	95.7
Dom_3	—	96.2	—	96.5	—	96.5
Dom_4	—	96.5	—	95.9	—	96.2

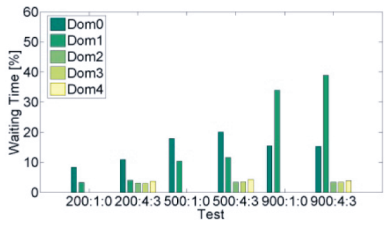

Fig. 7. Waiting time [percent] for 1 kB application under 200, 500, and 900 workload rates with 0 and 3 idle guest domains on Platform II.

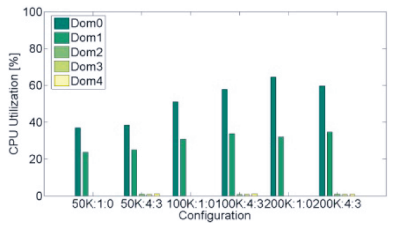

Fig. 8. CPU utilization [percent] for 50, 100, 200 kB applications under 200 workload rate with 0 and 3 idle guest domains on Platform II.

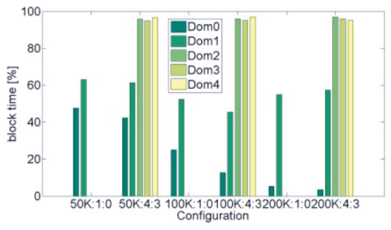

Fig. 10. Block time [percent] for 50, 100, 200 kB applications under 200 workload rate with 0 and 3 idle guest domains on Platform II.

running queue. The consequence of the high waiting time illustrates the reply time degradation shown in Table 3.

To further valid our observation above, we conduct the second set of experiments by setting the workload rate to 200 req/sec. We also vary the application from serving files with 50 to 200 kB. Figs. 8, 9, 10 recorded the CPU utilization, waiting time and block time for the three applications under 200 req/sec workload rate for both zero and three idle guest domain configurations.

Compared with the first set of experiments in Fig. 7, a noticeable difference is observed from Figs. 8 and 9: Dom_0 demands more CPU resources than Dom_1 for network I/O bound applications. This is because the data processing incurs memory page management interferences, such as packets lost for high latency, fragmentations, and increased data copying overhead. Also the imbalance in terms of demand for CPU resources by Dom_0 and Dom_1 will influence the allocation of CPU resources by the credit scheduler. Dom_0 gets more than its equal part of CPU share compared to Dom_1 (Fig. 8). Thus, Dom_0 will have to wait longer in the running queue of the physical CPU (Fig. 9), thus Dom_0 cannot always get its desired CPU on time (Dom_0 has longer waiting time), and consequently it cannot process I/O event destined for Dom_1 or the idle guest domains, leading to high block time for Dom_1 and idle domains and short block time for Dom_0 (Fig. 10).

4.3 Stability of Performance Metrics

We have shown in the previous sections that different performance metrics, such as CPU utilization, block time, wait time, throughput, are all important performance indicators. However, collecting all these measures frequently in real time can be costly. Some research [24] has shown that the monitoring service will not report the occurrence of some event until when the accumulated level of certain violations exceeds a specified threshold. In this section, we discuss briefly our experiences in terms of measuring the Coefficient of Variation (CoV) for CPU utilization, block time, and waiting time, which captures the fluctuation of each measurement statistic.

Fig. 11 shows that when Dom_1 is serving 1 kB applications with fixed rate of 500 req/sec, the CoV for CPU utilization (UT), block time (BL), and waiting time (WA) with respect to zero and three idle guest domains. We observe that the CPU utilization is the most stable performance indicator to illustrate CPU allocation, whereas the waiting time is the most frequently fluctuated factor. This observation tells us that if the monitoring resource is limited, one needs to track on a smallest set of statistics to detect performance saturation in a virtualized cloud, then the CPU utilization is a good and stable performance indicator to use. However, if more comprehensive metrics may be needed to monitor and notify the arrival of some specific event, then using the tolerance degree or tolerance interval of waiting time can be more informative than the CPU utilization and block time.

Another interesting observation from Fig. 11 is that the waiting time has the highest CoV while the CPU utilization has the lowest CoV. The CPU utilization has the lowest CoV is because we use the default settings of Apache web server, for example, the KeepAlive time and MaxClients are set by the default values. When the request rate is fixed, the CPU utilization and the demand for memory resources are fixed [42]. This means that the credit scheduler allocates a certain amount of CPU cycles to specific domain at the initial stage and the credit scheduler will revise its allocation of the CPU cycles based on the credits consumed. Take Dom_1 as an example, it means at the beginning Dom_1 gets its fair amount of CPU cycles in under states, and Dom_1 will be put into the running queue in a FIFO way. Consequently, Dom_1 get the normal waiting time. However, after Dom_1 consumes its preassigned credits, it goes into the over state, that means once Dom_1 is ready to run, it will be put to the end of the running queue, which means longer waiting time. As a result, the waiting time appears in an unstable fashion.

In order to understand other factors that may impact the CoV, we conduct another set of experiments. Fig. 12 shows the CoV of the waiting time for the 1 kB application under changing workload rate. We see a dramatic increase in the

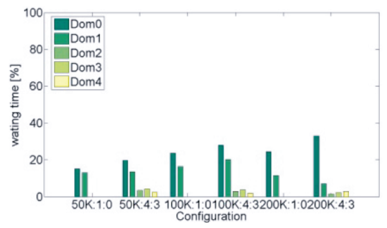

Fig. 9. Waiting time [percent] for 50, 100, 200 kB applications under 200 workload rate with 0 and 3 idle guest domains on Platform II.

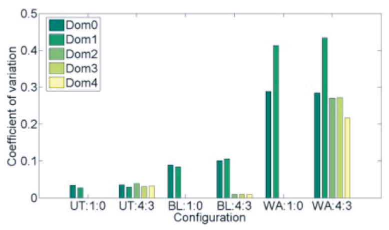

Fig. 11. Coefficient of variation for 1 kB application under 500 workload rate with 0 and 3 idle guest domains on Platform II.

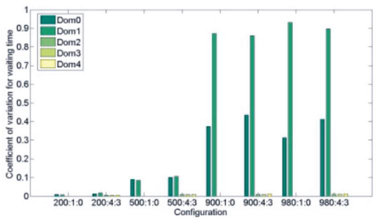

Fig. 12. Coefficient of variation for waiting time for 1 kB application under varying workload rates with 0 and 3 idle guest domains on Platform II.

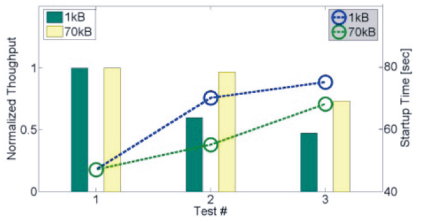

Fig. 14. Nomalized throughputs for 1 and 70 kB applications and startup time [sec] of one and two new guest domains on Platform I.

waiting time of Dom_1 and Dom_0 when the applied workload rate is high. This is because when the workload rate is high, there is a great opportunity for Dom_1 and Dom_0 going into the OVER state, leading the fluctuations, and high CoV in the waiting time for Dom_1 and Dom_0.

4.4 Starting up Guest Domains on Demand

We have studied the cost of keeping idle instances running in the previous sections. In this section, we study the cost of starting a virtual instance on demand. Concretely, we want to study how CPU intensive application and network I/O intensive application may impact the throughput performance of the running guest domain when new instances are started on demand next door. We also want to understand the startup time for creating one or more new guest domains on demand and the set of factors that may impact such start-up time.

There are two steps to start new service on demand: First, a cloud consumer needs to request new virtual machine instances from the cloud provider and the cloud service provider needs to find some existing idle instances or create new virtual machine instances in response to the consumer's request. Second, the cloud consumers need to set up the preinstalled or preconfigured instances that are ready to respond to client requests. The creation of new virtual machine is time consuming. We measured the creation time for new virtual machine with 1, 2, or 4 GB disk on Platform I. Fig. 13 shows that the creation time increases as the disk size that the instance posses increases. We also measured the startup time of new instances on Platform I. The instances are 1, 2, or 4 GB, respectively. The results show that the difference among the startup time of these instances is insignificant.

In order to study the startup time of new instances, we set up one guest domain on Platform I where Dom_1 is serving the 1 or 70 kB applications. Then we create one or two idle instances on demand. Fig. 14 shows the fluctuations in Dom_1's throughput and startup time for the idle guest domains for three sets of experiments: 1) running

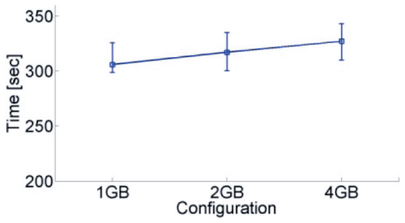

Fig. 13. Create time [sec] for virtual machine with size of 1, 2, and 4 GB on Platform I.

Dom_1 alone (Exp1), 2) running Dom_1 with startup one VM on demand (Exp2), and 3) running Dom_1 with startup two VMs on demand (Exp3). In this set of experiments, the request rate is fixed at 900 req/sec for the 1 kB application or 150 req/sec for the 70 kB application, both of which are approaching 90 percent of maximum throughput values given in Table 2. The primary y-axis (left) is the normalized throughput with 900 req/sec for 1 kB application or 150 req/sec for 70 kB application as the baseline, The second y-axis (right) denotes the start-up time (sec) for starting one (alone), one with Dom_1 running, or two VMs on demand with Dom_1 running. Note that the circles in Exp1 denote the startup time for one single instance without running Dom_1.

Fig. 14 presents us with three interesting observations. First, on demand start-up of guest domains has severe short term impact on the performance of running domain no matter what type of application is hosted by it. This is because starting up a VM instance is I/O intensive. In our experiment, it means to create one 2 GB guest domain instance. Thus, the average CPU consumption for starting up a new VM is about 20 percent, and the peak CPU consumption can be as high as 75 percent, and the virtual block device read is about a total of 900 and the virtual block device write is about a total of more than 200. These I/O related activities due to start-up new domains cannot be finished without the presence of Dom_0, which also plays a critical role in processing Dom_1's web workloads in our experiments. The second observation is that the 70 kB application suffers relatively less in terms of start-up time than the 1 kB application. This is because the performance of the 70 kB application is network bounded, and it consumes less CPU, which alleviates the CPU and disk resource contention at Dom_0. Concretely, for 1 kB application running in Dom_1, it will consume about 90 percent CPU resources in addition to about 5,400 memory page exchanges per second between Dom_0 and Dom_1 to serve at 900 req/sec rate. In contrast, only 60 percent CPU resource is reserved to serve the 70 kB application at 150 req/sec rate. Furthermore, for the 1 kB application, the startup time for creating two guest domains in Exp3 grows from 47 sec in Exp1 to 75 sec, about 1.5 times bigger. In contrast, for 70 kB application, the difference in start-up time from creating two VMs to one VM is relatively smaller. This indicates that the start-up time for creating new VMs on demand is related to both the type of resource-bound applications in the running domain and the number of new VMs being created. Given the CPU and disk I/O demands involved in creating new domains, both CPU intensive or disk I/O intensive applications in running domain will cost more

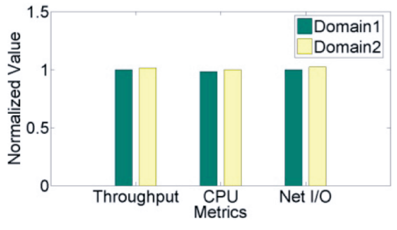

Fig. 15. Normalized throughput, CPU utilization and Net I/O between Dom_1 and Dom_2, both with identical 1 kB application at 50 percent workload rate.

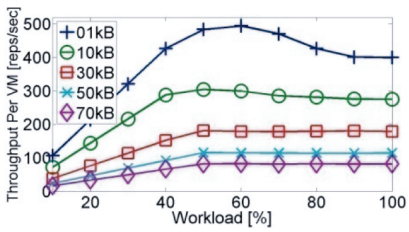

Fig. 16. Average throughput [req/sec] per guest domain, with both guest domains running identical application at the same workload rate.

start-up time than network I/O bounded applications. Third but not the least, we observe that the duration of performance degradation experienced due to creating new VMs on demand is typically bounded within 100 sec in our experiments, and is also related with the machine capacity, the workload rate supported in the running domain, and the number of new VM instances to be started. We argue that by understanding the cost of maintaining idle instances and the cost of starting up new instances on demand, both cloud providers and cloud consumers can make their respective decisions based on concrete scaling requirements of their services.

5 Impact of Neighboring Applications

In a virtualized cloud, some resources like CPU, memory are sliced across multiple VMs, whereas other resources like the network and the disk subsystem are shared among multiple VMs. We design three groups of experiments to perform an extensive measurement study on performance impact of co-locating applications with different resource usage patterns and different number of VMs. The first group and the second group of experiments focus on performance impact of running applications with different resource usage patterns. To isolate the number of factors that impact on the performance of co-locating patterns of applications, we choose the five I/O applications of 1, 10, 30, 50, and 70 kB in our experiments, but divide the experiments into three steps. In the first group, we run identical application on all VMs for all five applications. In the second step, we study the slightly more complex scenarios where different applications are running on different VMs. In the third group of experiments, we study the problem of distributing workloads among multiple VMs. Note that, all the results reported in this section are for experiments conducted on Platform I.

5.1 Colocating Identical Applications

In this group of experiments, we design two guest domains, Dom_1 and Dom_2, both serve identical web requests issuing at the same workload rates. In this simplified scenario, our experimental results show that when two identical I/O applications are running together, the credit scheduler can approximately guarantee their fairness in CPU slicing, network bandwidth consumption, and the resulting throughput.

Fig. 15 shows the experimental results for two VMs when both are serving 1 kB applications with 50 percent workload rate. We measured throughput, CPU utilization, Net I/O. For example, Dom_1 consumes 36.1 percent CPU resources while Dom_2 consumes 36.8 percent CPU resources. And the throughputs and network bandwidths for Dom_1 and Dom_2 are: 480 req/sec and 487 req/sec, 609 kB/sec and 622 kB/sec, respectively. We present these three metrics in normalized values to show their similarities. For each metrics pair, we use the value for Dom_1 as the comparative baseline. From Fig. 15, we see that the difference between the measurement in Dom_1 and the measurement in Dom_2 is trivial and can be ignored.

Fig. 16 measures the average throughput of Dom_1 and Dom_2 for all five I/O applications. We observe that 1) all the applications arriving at the peak performance under applied workload of 50 or 60 percent, 2) there is crucial difference between small-sized file application and large-sized file application. For small-sized file application such as 1 and 10 kB, obvious performance degradation can be observed at workload rates higher than 50 or 60 percent. However, this is not the case for large-sized file applications. The significant skew happened in the 1 kB application because: 1) its performance is bounded by the CPU resources, 2) the guest domain spends much more time to deal with the fast arrival of network packets when the workload rate is high, 3) compared with the single domain experiment for all five applications shown in Table 2, the overhead has increased due to the network bridging happened in Dom_0, and context switch between two guest domains.

Fig. 17 measures the CPU usages for 1 and 70 kB applications under varying workload rates. We add up CPU used by Dom_1 and Dom_2 together since the results in Fig. 15 indicate that Dom_1 and Dom_2 always get almost the same amount of CPU allocation. Fig. 17 presents an interest observation: under the same workload rate, the guest domain CPU usage for 1 kB file is much larger than that of the 70 kB application, despite the fact from our results that the memory page exchange rate for 1 kB file is much less than that of the 70 kB application. This verifies the CPU

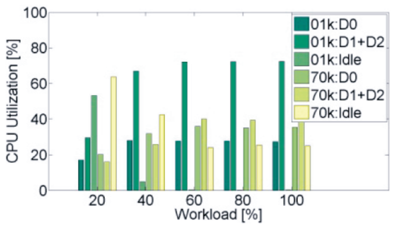

Fig. 17. CPU usage [percent] for Dom_0, aggregated CPU usage for guest domains, and percentage of idle CPU.

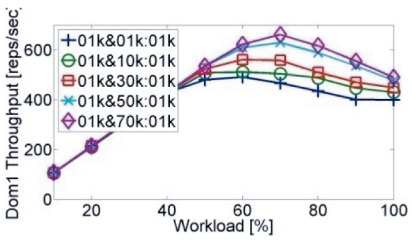

Fig. 18. Dom_1 throughput [req/sec] when Dom_1 is serving 1 kB appliaction and Dom_2 is serving 1 to 70 kB applications.

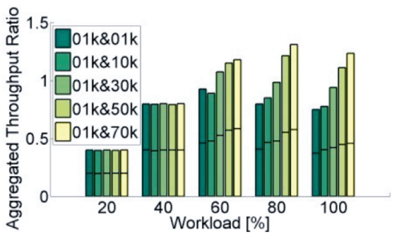

Fig. 20. Aggregated throughput ratio for Dom_1 and Dom_2 across five applied workload rates.

bounded nature of the 1 kB application. The CPU consumed to process network requests is mainly composed of two major components: the time spent in establishing TCP connections, and the time spent in transporting web file content. Furthermore, for the 1 kB application, the connection phase demands significantly more CPU resources than the transportation phase (refer to Table 3).

5.2 Colocating Different Applications

From experimental results in the previous section, we know that when two applications are identical, then approximate fairness can be obtained by using the default credit scheduler in Xen. Thus the main factors that impact the performance of applications colocated on the same physical host are applied workload rates and resource usage patterns of applications. In this section, we examine the performance for guest domains when they are serving different applications as this is more likely to happen in real world scenario. We simulate two cloud consumers, one is using Dom_1 and serving the 1 kB application, the other is using Dom_2, running the application, which is by design varying from 1 to 70 kB.

Figs. 18 and 19 measure the throughputs for Dom_1 and Dom_2 under the 70 percent workload, respectively. We observe two interesting facts: 1) although Dom_1 always serves the 1 kB file, its performance highly depends on the application running in its neighbor Dom_2. For example, in the 1 and 70 kB combination (661 req/sec for the highest 1 kB throughput in Dom_1) compared with in the 1 and 1 kB combination (494 req/sec for the highest 1 kB throughput in Dom_1), the performance difference can be 34 percent. 2) The highest throughput points occurring in Figs. 18 and 19 show considerably different tendency. Take the 1 and 70 kB application combination as an example, for the two guest domains, the highest throughput points come out under different applied workloads: the highest point for the 1 kB file appears at 70 percent workload rate, while it comes at 100 percent workload for the 70 kB application. Clearly, this phenomenon is due to the resource usage pattern of 1 and 70 kB applications, 1 kB is CPU bounded and 70 kB is network bounded.

Fig. 20 measures the aggregated throughput ratio as a function of workload rate. We use the maximum throughput of single VM for five applications in the first column of Table 2 as the baseline to get individual throughput ratio for each guest domain under each specific workload. For example, the throughput for Dom_1 is 661 req/sec at 70 percent workload rate, thus the throughput ratio is about 62 percent (661/1,070). Similarly, we have the throughput ratio of 68 percent for the 70 kB application. Thus the aggregated throughput ratio is 130 percent. From the results for five combinations of neighboring applications in Fig. 20, we observe that the best colocating case is the 1 and 70 kB combination with the highest aggregated throughput ratio of 1.3, and the worst case is the 1 and 1 kB combination with the highest aggregated throughout ratio of 0.92, The performance difference could be more than $40\%((1.3 - 0.92)/0.92 = 41\%)$.

5.3 Colocating Applications among Multiple VMs

We have studied the impact of colocating applications on two guest domains hosted on a single physical node. In this section, we dedicate our measurement study to examine the impact of multiple VMs on application colocation strategy.

Our first set of experiments is designed by varying the number of guest domains from one to six and each guest domain serves 10 percent applied workload. Total workload rate can be calculated by multiplying the number of guest domains with the applied workload rate. Using 10 percent workload applied to each guest domain guarantees no severe resource contention will occur. Fig. 21 shows when there are six guest domains running, the CPU time spent (9.8 percent) to process the same amount of I/O data (10 percent workload per guest domain) equals to 1.5 times of the CPU time spent (6.9 percent) in the single guest domain case. This group

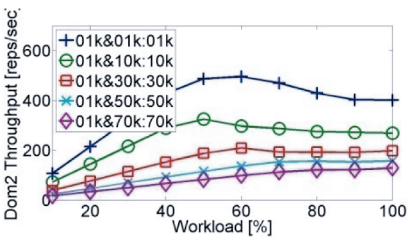

Fig. 19. Dom_2 throughput [req/sec] when Dom_1 is serving 1 kB application and Dom_2 is serving 1 to 70 kB applications.

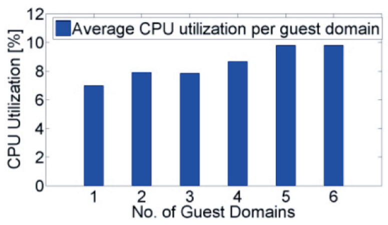

Fig. 21. Average CPU utilization [percent] for each guest domain when varying the number of guest domains from one to six, each is serving 10 percent workload rate.

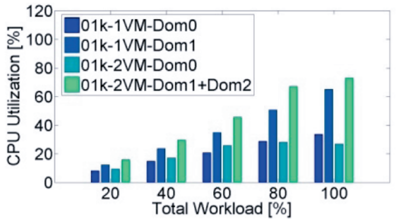

Fig. 22. CPU usage [percent] by one and two guest domains with varying workload rates.

of experiments intends to show that compared with single guest domain case, when multiple guest domains are running, the context switches among the guest domains will lead to more frequent cache miss and TLB miss [26], which will result in more CPU time consumption in serving the same data. The cost of VM context switches is typically proportional to the number of guest domains hosted on a physical machine.

For the second set of experiments, we fix the total workload rates to 20, 40, 60, 80, and 100 percent. The 1 and 70 kB application are chosen as they are the two representative applications. Fig. 22 shows the CPU utilization measured for both driver domain and guest domain under two types of virtual machine configurations: single VM and two VMs. For example, 01k-1VM-Dom0 denotes the measurement of Dom_0 CPU utilization for 1 kB application running on a single VM. 01k-2VM-Dom0 denotes the measurement of Dom_0 CPU utilization for 1 kB application running on two VMs. 01k-2VM-$\mathrm{Dom}_1 + \mathrm{Dom}_2$ measures the combined CPU usage of both Dom_1 and Dom_2 for 1 kB application. Two VCPUs are configured for each guest domain. When two guest domains are running, six VCPUs are waiting for being scheduled into the physical CPUs, compared with four VCPUs in single guest domain case. Frequent context switches incur undesirable cache miss and TLB miss. For the two guest domain experiments, Dom_0 has to deal with both the context switch and the scheduling overhead, also the network bridging overhead is raised due to transferring packets to individual guest domains. Thus Dom_0 gets larger fraction of CPU resources for the two guest domain setting. This set of experimental results also shows that the CPU usage in the guest domain increases sharply as the workload rate approaches 100 percent.

Fig. 23 shows the CPU usages under high contention situation. We varied the total workload rates to 120 percent. As seen from the figure, when the number of guest domains grows from one to six, the CPU share for Dom_0 reduces at a more gradual rate for the 70 kB application (32.5 to 31.3 percent). In contrast, when the number of guest

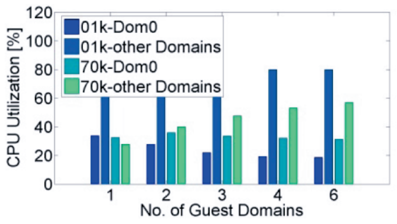

Fig. 23. CPU usage [percent] for one, two, three, four and six guest domains with 120 percent workload rate.

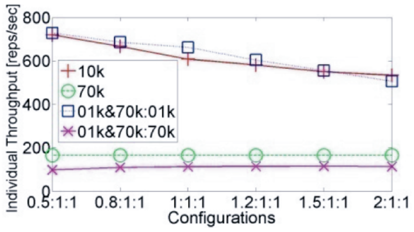

Fig. 24. Throughputs for three groups of experiments with varying the weightings among Dom_0 and guest domains.

domains is changed to six, the CPU utilization at Dom_0 for the 1 kB application is reduced from 33.8 to 18.6 percent. For the 1 kB application, the significant reduction in Dom_0 CPU utilization indicates the growing CPU contention due to the continuous growth in the guest domain CPU usages. The credit scheduler tries to fairly share CPU slices among domains including Dom_0.

6 Impact of Tuning Credit Scheduler

In all the previous experiments, Dom_0 is allocated equal amount of CPU shares as the rest of the guest domains in Xen. In this section, we conduct a group of experiments to measure the impact of tuning the credit scheduler on the overall performance of the applications. We show that significant opportunity exists in optimizing the overall system performance by simply tuning the weight parameter of the credit scheduler. In the cloud computing environment, if the cloud provider could carefully design and tune the scheduling parameters, considerable performance gain can be achieved.

We design this group of experiments by configuring each experiment with different amount of CPU allocated to Dom_0 relative to Dom_1 and Dom_2. For example, $0.5:1:1$ means that Dom_0 is assigned half of the weight of Dom_1 and Dom_2, and 2:1:1 means that Dom_0 obtains twice as much the CPU weight as that of Dom_1 and Dom_2. Note, Dom_1 and Dom_2 always obtain identical CPU weight. The settings for the three sets of experiments are as follows: In Experiment 1, each of the two clients is sending 10 kB application with the rate 360 req/sec for one virtual machine. In Experiment 2, two clients send 70 kB file retrieval requests to two different VMs, each at a fixed rate of 82 req/sec. In Experiment 3, one guest domain is processing 1 kB file requests at the rate of 800 req/sec, and the other guest domain is processing 70 kB file requests at the rate of 150 req/sec. Note that, all the experiments reported in this section were conducted on Platform I.

Fig. 24 summarizes throughputs for three experimental settings. For 10 and 70 kB case, we show the aggregated throughputs. For the 10 kB test, compared to the best case of assigned ratio of $0.5:1:1$ (720 req/sec), the performance of the worst case (534 req/sec), which appears at the ratio of 2:1:1, goes down by about 25 percent. For the 70 kB case, the flattened behavior of the curve is due to the network bound nature of the 70 kB applications. For the 1 and 70 kB application combination, for the 1 kB file, when the configuration switches from $0.5:1:1$ to $2:1:1$, about 30 percent performance degradation can be observed, while

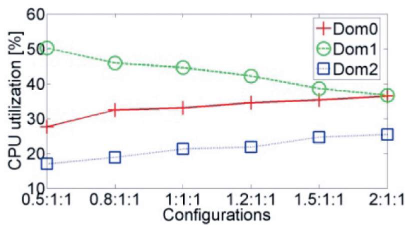

Fig. 25. CPU utilization [percent] for 1 kB appliacation at 800 req/sec and 70 KB at 150 req/sec with different weightings assigned to Dom_0 and guest domains.

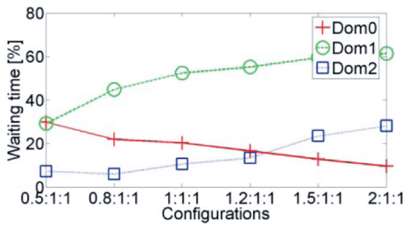

Fig. 26. Waiting time [percent] for 1 kB appliacation at 800 req/sec and 70 KB at 150 req/sec with different weightings assigned to Dom_0 and guest domains.

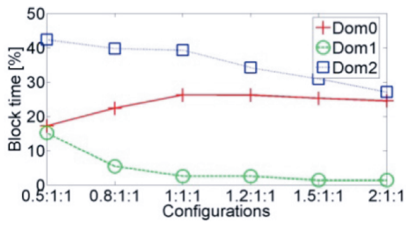

Fig. 27. Block time [percent] for 1 kB appliacation at 800 req/sec and 70 KB at 150 req/sec with different weightings assigned to Dom_0 and guest domains.

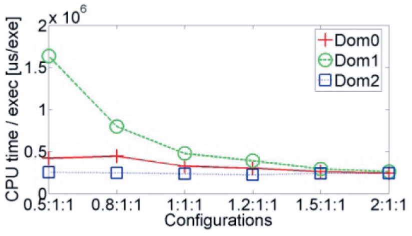

Fig. 28. Allocated CPU time per execution [μs/exe] for 1 kB appliacation at 800 req/sec and 70 KB at 150 req/sec with different weightings assigned to Dom_0 and guest domains.

for the 70 kB application about 16 percent performance gain can be obtained.

Figs. 25, 26, and 27 show CPU utilization, block time, and waiting time, respectively, for Dom_0, Dom_1, and Dom_2 with different configurations of the scheduler, where Dom_1 runs the 1 kB application at 800 req/sec, and Dom_2 runs the 70 kB application at 150 req/sec. From Fig. 25, we could see that as the weighting allocated to Dom_0 is increased, Dom_0 CPU utilization is also increased while Dom_1 CPU utilization is dropped as expected. A side effect of this tuning is the increase in Dom_2 CPU utilization. This also explained the 30 percent performance degradation for the 1 KB application and the 16 percent performance gain for the 70 kB application shown in Fig. 24. Fig. 26 shows that only Dom_0 waiting time is improved when we increase the weighting to Dom_0. This is because the weighting parameter determines how much CPU resources to be allocated when multiple domains are

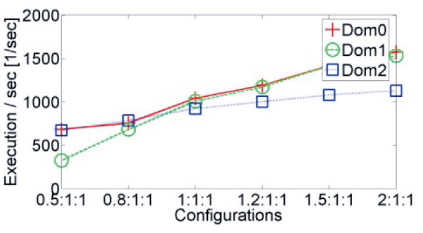

Fig. 29. Execution per second [1/sec] for 1 kB appliacation at 800 req/sec and 70 KB at 150 req/sec with different weightings assigned to Dom_0 and guest domains.

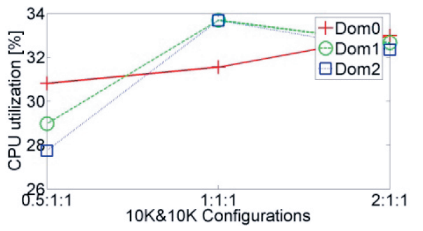

Fig. 30. CPU utilization [percent] for dual 10 kB at 360 req/sec with different weightings assigned to Dom_0 and guest domains.

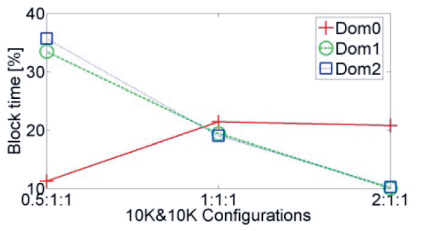

Fig. 31. Block time [percent] for dual 10 kB at 360 req/sec with different weightings assigned to Dom_0 and guest domains.

competing for CPU cycles. As a result, when we increase the weighting for Dom_0, the waiting time for Dom_0 is reduced, at the same time, the priority for processing guest domains reduced, thus the waiting time for both Dom_1 and Dom_2 are increased as shown in Fig. 26. Dom_1 has the highest waiting time because Dom_1 is demanding more CPU and it often over charges its credits, and is switched to the Over state, leading to the largest waiting time in all the weight configurations.

Fig. 27 shows that the block time for Dom_1 and Dom_2 is reduced due to the improvement of Dom_0 waiting time, which means Dom_0 can run faster and process I/O event destined for guest domains on time. However, as the waiting time for Dom_1 and Dom_2 is increased when we switch the configurations from 0.5:1:1 to 2:1:1, Dom_1 and Dom_2 can no longer process I/O event destined for Dom_0 quickly, resulting in some increase in Dom_0 block time.

To further illustrate the up and down in CPU utilization shown in Fig. 25, we design another set of experiments to measure the allocated CPU time per execution and the execution count per second. Fig. 28 shows that as we switch the Dom_0 weight configuration from 0.5:1:1 to 2:1:1, Dom_1 suffers the most, and its allocated CPU time per execution (2:1:1) is dropped to about 1/3 of the original setting (0.5:1:1). Compared with Dom_1, we could see that the allocated time per execution for Dom_0 and Dom_2 remain at almost the same level. However, the execution counts per second for Dom_0 and Dom_2 are increased. Combining Figs. 28 and 29, we observe that when we switch the configuration from 0.5:1:1 to 2:1:1, Dom_0 and

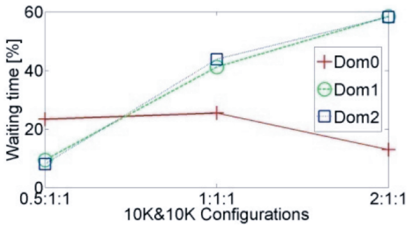

Fig. 32. Waiting time [percent] for dual 10 kB at 360 req/sec with different weightings assigned to Dom_0 and guest domains.

Dom_2 remain the same level of Allocated CPU time per execution, allowing them to benefit from the increased execution count per second. As a result, they are allocated more CPU resources. This illustrates the CPU utilization reallocation shown in Fig. 25.

Figs. 30, 31, 32 show the CPU utilization, waiting time and block time, respectively, for dual 10 kB at 360 req/sec with different weightings assigned to Dom_0, Dom_1, and Dom_2. Not surprisingly, as the weighting allocated to Dom_0 increases, we observe that: 1) Dom_0 CPU utilization is increased, 2) the waiting time for Dom_0 is reduced and the waiting times for Dom_1 and Dom_2 are increased, and 3) Dom_1 block time and Dom_2 block time are improved by making Dom_0 working more efficiently. An interesting observation is that when the weight configuration is 0.5:1:1, Dom_1 and Dom_2 can get more CPU than Dom_0 and at the same time Dom_0 can work more efficiently than any other weight configurations. This can be illustrated through measuring the number of memory page exchanges per execution for different weight configurations. As shown in Table 5, the *memory pages exchanged per execution* for Dom_0 drops dramatically as the weighting allocated to Dom_0 increases.

Table 5 summarizes performance statistics for Dom_0 over three configurations. The *execution count per second* increased with the weight allocated to Dom_0 grows, which means the context switches frequency increased. At the same time, *memory pages exchanged per second* for Dom_0 grows as the weight assigned to Dom_0 decreases, which means that the driver domain is capable of processing more network packets during the best case (0.5:1:1). The *memory pages exchanged per execution*, the *memory pages exchanged per second*, and the *memory pages exchanged per execution* describe the efficiency of processing network packets upon Dom_0 being scheduled to run on physical CPU.

7 Conclusion

We have presented an in-depth performance measurement study focused on network I/O application, one of the dominating workloads in cloud-based virtual data centers. We first show that current implementation of virtual machine monitor does not provide sufficient performance isolation to guarantee the effectiveness of resource sharing across multiple virtual machine instances running on a single physical host machine, especially when applications running on neighboring VMs are competing for computing and communication resources. Then we present the detailed analysis on different factors that can impact the throughput performance and resource sharing effectiveness. Concretely, we presented our measurement study and analysis along three dimensions: 1) the performance impact of idle instances on applications that are running concurrently on the same physical host; 2) the performance impact of colocating applications in a virtusalized data center; and 3) how different CPU resource scheduling and allocation strategies and different workload rates may impact the performance of a virtualized system. In the first dimension of study, our experimental results show that in general CPU utilization, block time, waiting time, throughput are all important performance indicators for running idle instances. Furthermore, disk size and resource usage patterns (CPU intensive or network IO intensive) may impact the creation time of new instances. In the second study, we show that both applied workload rates and the resource usage patterns are two important factors for performance of colocated applications. Also the cost of VM switches is typically proportional to the number of guest domains hosted on a physical machine. Such context switches will lead to more frequent cache miss and TLB miss, which results in more CPU time in serving the same data. In the third study, we show that significant opportunity exists in optimizing the overall system performance by simply tuning the weight parameters of the credit scheduler. Several factors impact the settings of such weights, including memory page exchanged per execution, memory page exchanged per second, and execution count per second. We argue that by exploiting optimizations for colocating different applications, performance improvement for cloud consumers can be as high as 34 percent, and at the same time, the cloud providers can achieve over 40 percent performance gain by strategically colocating network I/O applications together.

TABLE 5
Performance Metrics Related to Three Configurations for Dom_0 with Dual 10 kB Applications

Metrics		Configuration		
		0.5:1:1	1:1:1	2:1:1
Aggregated Throughput [req/sec]		720	608	534
Network I/O [kB/sec]		7147	6296	5300
Dom_0	Exe/sec [1/sec]	950	1102.7	1846
	CPU/exe [µs/exe]	308672	211771	184084
	Mem/sec [pages/sec]	11520	9728	8544
	Mem/exe [pages/exe]	12.1	8.8	4.6

Acknowledgments

This work was partially supported by grants from the US National Science Foundation (NSF) CISE NetSE program, the NSF CISE CyberTrust program, an IBM faculty award, an IBM SUR grant, a grant from the Intel Research Council and the Intel ISTC on Cloud Computing. Yiduo Mei performed this research as a visiting PhD student at the Distributed Data Intensive Systems Lab (DiSL) in the College of Computing, Georgia Institute of Technology, supported by the China Scholarship Council, the Department of Computer Science at Xi'An Jiaotong University, NSFC under grant no. 11101401, and the Ministry of Education of P.R. China under grant no. B09C1100020. Xiaoshe Dong was partially supported by NSFC under grant number 60773118.

REFERENCES

[1] P. Apparao, R. Iyer, X. Zhang, D. Newell, and T. Adelmeyer, "Characterization & Analysis of a Server Consolidation Benchmark," *Proc. ACM/USENIX Int'l Conf. Virtual Execution Environments,* pp. 21-29, 2008.

[2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," Technical Report UCB/EECS-2009-28, http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html, 2010.

[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. ACM Symp. Operating Systems Principles (SOSP),* pp. 164-177, 2003.

[4] Z. Chen, D. Kaeli, and K. Murphy, "Performance Evaluation of Virtual Appliances," *Proc. First Int'l Workshop Virtualization Performance: Analysis, Characterization, and Tools,* Apr. 2008.

[5] L. Cherkasova and R. Gardner, "Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor," *Proc. USENIX Ann. Technical Conf. (ATC),* p. 24, 2005.

[6] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the Three CPU Schedulers in Xen," *ACM Sigmetrics Performance Evaluation Rev.,* vol. 35, no. 2, pp. 42-51, Sept. 2007.

[7] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J.N. Matthews, "Xen and the Art of Repeated Research," *Proc. USENIX Ann. Technical Conf. (ATC),* pp. 135-144, 2004.

[8] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfiel, "Live Migration of Virtual Machines," *Proc. USENIX Symp. Network Systems Design and Implementation (NSDI),* pp. 273-286, 2005.

[9] Credit Based CPU Scheduler, http://wiki.xensource.com/xenwiki/CreditScheduler, 2010.

[10] T. Deshane, Z. Shepherd, J.N. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao, "Quantitative Comparison of Xen and KVM," *Xen Summit Boston 2008,* http://xen.org/xensummit/xensummit_summer_2008.html, 2010.

[11] Y. Dong, X. Yang, X. Li, J. Li, K. Tian, and H. Guan, "High Performance Network Virtualization with SR-IOV," *Proc. IEEE 16th Int'l Symp. High Performance Computer Architecture (HPCA),* pp. 1-10, 2010.

[12] S. Govindan, A.R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, "Xen and Co.: Communication-Aware CPU Scheduling for Consolidated Xen-Based Hosting Platforms," *Proc. ACM/USENIX Int'l Conf. Virtual Execution Environments,* pp. 126-136, 2007.

[13] A. Gulati, A. Merchant, and P. Varman, "mClock: Handling throughput Variability for Hypervisor IO Scheduling," *Proc. Ninth USENIX Symp. Operating System Design and Implementation (OSDI),* 2010.

[14] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing Performance Isolation across Virtual Machines in Xen," *Proc. ACM/IFIP/USENIX Seventh Int'l Conf. Middleware (Middleware),* pp. 342-362, 2006.

[15] D. Gupta, R. Gardner, and L. Cherkasova, "XenMon: QoS Monitoring and Performance Profiling Tool," Technical Report HPL-2005-187, http://www.hpl.hp.com/techreports/2005/HPL-2005-187.html, 2010.

[16] M. Hines and K. Gopalan, "Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning," *Proc. ACM/USENIX Int'l Conf. Virtual Execution Environments,* pp. 51-60, 2009.

[17] M. Kesavan, A. Gavrilovska, and K. Schwan, "Differential Virtual Time (DVT): Rethinking I/O Service Differentiation for Virtual Machines," *Proc. ACM Symp. Cloud Computing (SOCC),* pp. 27-38, 2010.

[18] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee, "Task-Aware Virtual Machine Scheduling for I/O Performance," *Proc. ACM/USENIX Int'l Conf. Virtual Execution Environments,* pp. 101-110, 2009.

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

[20] KVM, http://www.linux-kvm.org/page/Main_Page, 2012.

[21] M. Lee, A.S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Supporting Soft Real-Time Tasks in the Xen Hypervisor," *Proc. ACM/USENIX Int'l Conf. Virtual Execution Environments,* pp. 97-108, 2010.

[22] J.N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, "Quantifying the Performance Isolation Properties of Virtualization Systems," *Proc. Workshop Experimental Computer Science,* article 6, 2007.

[23] 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. Cloud Computing,* pp. 59-66, 2010.

[24] S. Meng, T. Wang, and L. Liu, "Monitoring Continuous State Violation in Datacenters: Exploring the Time Dimension," *Proc. IEEE 26th Int'l Conf. Data Eng. (ICDE),* pp. 968-979, 2010.

[25] A. Menon, A.L. Cox, and W. Zwaenepoel, "Optimizing Network Virtualization in Xen," *Proc. USENIX Ann. Technical Conf. (ATC),* pp. 15-28, 2006.

[26] A. Menon, J.R. Santos, Y. Turner, G.J. Janakiraman, and W. Zwaenepoel, "Diagnosing Performance Overheads in the Xen Virtual Machine Environment," *Proc. ACM/USENIX Int'l Conf. Virtual Execution Environments (VEE),* pp. 13-23, 2005.

[27] D. Mosberger and T. Jin, "Httperf-A Tool for Measuring Web Server Performance," *ACM Sigmetrics Performance Evaluation Rev.,* vol. 26, no. 3, pp. 31-37, Dec. 1998.

[28] N. Nishiguchi, "Evaluation and Consideration of the Credit Scheduler for Client Virtualization," *Xen Summit Asia,* http://www.xen.org/xensummit/xensummit_fall_2008.html, 2010.

[29] D. Ongaro, A.L. Cox, and S. Rixner, "Scheduling I/O in Virtual Machine Monitors," *Proc. ACM/USENIX Int'l Conf. Virtual Execution Environments,* pp. 1-10, 2008.

[30] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K.G. Shin, "Performance Evaluation of Virtualization Technologies for Server Consolidation," Technical Report HPL-2007-59R1, http://www.hpl.hp.com/techreports/2007/HPL-2007-59R1.html, 2010.

[31] F. Prefect, L. Doan, S. Gold, and W. Wilcke, "Performance Limiting Factors in Http (Web) Server Operations," *Proc. IEEE 41st Int'l Computer Conf. (COMPCON '96),* pp. 267-273, 1996.

[32] 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/USENIX Int'l Conf. Virtual Execution Environments,* pp. 61-70, 2009.

[33] K.K. Ram, J.R. Santos, and Y. Turner, "Redesigning Xen's Memory Sharing Mechanism for Safe and Efficient I/O Virtualization," Technical Report HPL-2010-39, http://www.hpl.hp.com/techreports/2010/HPL-2010-39.html, 2010.

[34] A. Ranadive, M. Kesavan, A. Gavrilovska, and K. Schwan, "Performance Implications of Virtualizing Multicore Cluster Machines," *Proc. Workshop System-Level Virtualization for High Performance Computing,* pp. 1-8, 2008.

[35] C.P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M.S. Lam, and M. Rosenblum, "Optimizing the Migration of Virtual Computers," *Proc. Fifth USENIX Symp. Operating Systems Design and Implementation (OSDI),* pp. 377-390, 2002.

[36] S. Sivathanu, L. Liu, Y. Mei, and X. Pu, "Storage Management in Virtualized Cloud Environment," *Proc. IEEE Int'l Conf. Cloud Computing,* pp. 204-211, 2010.

[37] G. Somani and S. Chaudhary, "Application Performance Isolation in Virtualization," *Proc. IEEE Int'l Conf. Cloud Computing,* pp. 41-48, 2009.

[38] VMware, http://www.vmware.com, 2012.

[39] J. Wang, K. Wright, and K. Gopalan, "XenLoop: A Transparent High Performance Inter-VM Network Loopback," *Proc. 17th Int'l Symp. High Performance Distributed Computing (HPDC),* pp. 109-118, 2008.

[40] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy, "Profiling and Modeling Resource Usage of Virtualized Applications," *Proc. ACM/IFIP/USENIX Ninth Int'l Conf. Middleware (Middleware),* pp. 366-387, 2008.

[41] T. Wood, P.J. Shenoy, A. Venkataramani, and M.S. Yousif, "Black-Box and Gray-Box Strategies for Virtual Machine Migration," *Proc. USENIX Symp. Networked Systems Design and Implementation (NSDI),* pp. 229-242, 2007.

[42] D. Yixin, N. Gandhi, J.L. Hellerstein, S. Parekh, and D.M. Tilbury, "Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics with Application to the Apache Web Server," *Proc. IEEE/IFIP Network Operations and Management Symp. (NOMS),* pp. 219-234, 2002.

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 to 2010 at the Distributed Data intensive Systems Lab (DiSL) at the Georgia Institute of Technology. In 2011, he began working as a post doctoral researcher at the China Center for Industrial Security Research, Beijing Jiaotong University. His main research interests include cloud computing, grid computing, system virtualization, performance optimization, trust management, and big data.

Ling Liu is a full professor in the School of Computer Science at the Georgia Institute of Technology. There, she directs the research programs in the Distributed Data Intensive Systems Lab (DiSL), examining various aspects of data intensive systems with a focus on performance, availability, security, privacy, and energy efficiency. She was a recipient of the best paper award at ICDCS 2003, WWW 2004, and the 2008 International Conference on Software Engineering and Data Engineering, and also won the 2005 Pat Goldberg Memorial Best Paper Award. She has served as the general chair or PC chair for numerous IEEE and ACM conferences in the data engineering, distributed computing, service computing, and cloud computing fields and is a coeditor-in-chief of the five-volume *Encyclopedia of Database Systems* (Springer). Currently, she is on the editorial boards of several international journals, such as *Distributed and Parallel Databases* (DAPD, Springer), the *Journal of Parallel and Distributed Computing (JPDC)*, *ACM Transactions on Web*, *IEEE Transactions on Service Computing (TSC)*, and *Wireless Network* (WINET, Springer). Her current research is primarily sponsored by the US National Science Foundation, IBM, and Intel. She is a senior member of the IEEE.

Xing Pu received the BE degree and the PhD degree from the School of Computer Science and Technology at the Beijing Institute of Technology in 2004 and 2012, respectively. Currently, he is an engineer at State Radio Regulation of China. His main research interests include I/O models, memory management, security, and performance isolation in virtualized systems.

Sankaran Sivathanu received the bachelors degree from Anna University, India, in 2007, the masters degree in computer science from Georgia Tech, and the PhD degree in computer science from the College of Computing, Georgia Institute of Technology, in 2011. His research interests include storage systems and system virtualization.

Xiaoshe Dong received the BE and ME degrees in computer science from Xi'an Jiaotong University, China, in 1985 and 1990, respectively, and the PhD degree in computer science from Keio University, Japan, in 1999. He was a lecturer from 1987-1994 and an associate professor from 1999-2003 in the Department of Computer Science & Technology of Xi'an Jiaotong University, where he has been a full professor with the Department of Computer Science & Technology since 2003. His interests include computer architecture and high performance computing.