Performance Measurements and Analysis of Network I/O Applications in Virtualized Cloud

Yiduo Mei[1,2], Ling Liu[1], Xing Pu[1,3], Sankaran Sivathanu[1]

[1] Georgia Institute of Technology, Atlanta, Georgia, USA, 30332
[2] Xi'an Jiaotong University, Xi'an, ShaanXi, China, 710049
[3] Beijing Institute of Technology, Beijing, China, 100081

Abstract*—Virtualization is a key technology for cloud based data centers to implement the vision of infrastructure as a service (IaaS) and to promote effective server consolidation and application consolidation. However, current implementation of virtual machine monitor does not provide sufficient performance isolation to guarantee the effectiveness of resource sharing, especially when the applications running on multiple virtual machines of the same physical machine are competing for computing and communication resources. In this paper, we present our performance measurement study of network I/O applications in virtualized cloud. We focus our measurement based analysis on performance impact of co-locating applications in a virtualized cloud in terms of throughput and resource sharing effectiveness, including the impact of idle instances on applications that are running concurrently on the same physical host. Our results show that by strategically co-locating network I/O applications, performance improvement for cloud consumers can be as high as 34%, and the cloud providers can achieve over 40% performance gain.*

Keywords: Cloud computing; Virtualization; Performance measurement.

I. INTRODUCTION

Cloud computing is changing the way that IT industry operates and its corresponding profit model by providing the IT infrastructure and software as chargeable services delivered over the Internet. Virtual machine monitors (VMMs) are popular in cloud environment within the service-delivery industry. The promise of virtualization technology for server consolidation and application consolidation is to run multiple services on a single physical machine (host) while allowing independent configuration of operating systems, software, and device drivers. Virtualization helps to achieve greater system utilization, and at the same time lowering total cost of ownership, responding more effectively to changing business conditions in enterprise, government and organizations.

However, current implementation of VMMs does not provide sufficient performance isolation to guarantee the effectiveness of resource sharing, especially when the applications running on multiple virtual machines (VMs) of the same physical machine are competing for computing and communication resources [11, 14, 16, 20]. 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.

In this paper we focus on performance measurement and analysis of network I/O applications (network-intensive applications) in a virtualized cloud. To maximize the benefit and effectiveness of server consolidation and application consolidation, we argue that it is important to conduct in-depth performance measurements for applications running on multiple VMs hosted on a single physical machine. Such measurements can offer deeper understanding of the key factors for effective resource sharing among applications running in virtualized cloud environments. We focus on network I/O applications in the measurement study because network intensive applications are known to be the dominating workloads in cloud-based data centers today, as evidenced by Amazon EC2 [22], Google AppEngine [23].

We concentrate our measurement study to two categories of server and application scheduling problems. The first category aims at addressing issues related to managing idle instances. When a domain is said to be idle, it means that there is no other runnable processes and the OS is executing idle-loop. Through this group of experimental study, we show the impact of running idle guest domains on system performance. We believe that the findings from this experimental study can help cloud service providers to effectively manage virtual machines to better meet consumers' demand, and at the same time, it can also provide useful insights to cloud consumers to manage idle instances more effectively for seamlessly scaling their applications.

The second focus of our performance study is to understand the performance impact of co-locating applications in a virtualized cloud in terms of throughput performance and resource sharing effectiveness. Through in-depth measurement analysis, we can better understand the set of key factors that can maximize the physical host capacity and the application performance running on individual VMs.

In summary, through this measurement study we show that it is important to understand the fundamental resource usage model and performance issues for concurrent I/O applications running in virtualized cloud. In addition, we show that the ability of exploring and quantifying the performance gains and losses relative to different configurations in guest domains and applications can provide valuable insights for cloud service providers and could consumers. Furthermore, our measurement analysis also reveals that applications should be arranged carefully and smartly to minimize unexpected performance degradation and maximize desired performance gains.

The remaining of this paper is structured as follows. We describe research related to our study in Section 2. In Section 3 and Section 4, we present our experiments and analyze the results. Section 5 summarizes our results and presents conclusions.

II. OVERVIEW AND BACKGROUD

In this section we first briefly review Xen [1], especially some features and implementation details of Xen, which are important backgrounds for our performance analysis and

measurement study. Then we briefly review the related work in the literature and outline our basic methodology for conducting performance measurement and analysis of network I/O applications in virtualized cloud environments.

A. Xen I/O Mechanism

Xen is an x86 VMM (hypervisor) developed based on paravirtualization [1]. VMM interfaces between the virtual machine tier and the underlying physical machine resources. At boot time, an initial domain, called Domain0, is created and serves as the privileged management domain, which uses the control interface to create and terminate other unprivileged domains (guest domains), and manages the CPU scheduling parameters and resource allocation policies.

In Xen [1], Domain0 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 Dom_1, Dom_2, …, Dom_n (n>1). A guest domain implements a virtual network interface controller driver called netfront to communicate with corresponding netback driver in Domain0. 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, 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. The procedure is reversed for sending packets from the guest domains via the driver domain [1, 4, 9, 10, 16, 19].

B. Credit Scheduler

Xen [1] employs the credit scheduler to facilitate load balancing on symmetric multiprocessing (SMP) host. The non-zero 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. 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. 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 [3, 6, 10, 16].

C. Related Work

Over the last few years, a fair number of research and development efforts have been dedicated to the enhancement of virtualization technology. Most of the efforts to date can be classified into two categories: (i) performance monitoring and enhancement of VMs on a single physical machine, and (ii) performance evaluation, enhancement, and migration of VMs running on multiple physical hosts. We below provide a brief summary about the research conducted. We can characterize this line of research in two directions. On one hand, a number of research projects have been devoted to performance monitoring tools for VMM and VMs, represented by the monitoring tools [3, 4, 7, 12, 15] for Xen [1, 5]. On the other hand, a fair amount of work has been conducted on varying CPU scheduler configurations [3, 9, 10, 15] or network I/O related parameter tuning [12, 13, 19], such as network bridging, TCP Segmentation Offload (TSO).

Concretely, some previous study has shown that performance interference exists among multiple virtual machines running on the same physical host due to the shared use of computing resources [15, 18, 20] and the implicit resource scheduling of different virtual machines done by VMM in privileged driver domain [11]. For example, in the current Xen implementation, all the I/O requests have to be processed by the driver domain, and Xen does not explicitly differentiate the Domain0 CPU usage caused by I/O operations for each guest domain. The lacking of mechanism for Domain0 to explicitly separate its usage contributes to the unpredictable performance interference among guest domains [8].

D. Basic Methodology and Performance Metrics

Experimental Setup. All experiments were conducted on an 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 100Mbps network connection. We used the Ubuntu 8.0.4 distribution and Xen 3.2 with the default credit scheduler. The physical machine 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. Each connection issues one file request by default. A control node coordinates individual clients and collects profiling data. The web server performance is measured as a maximum achievable number of connections per second when retrieving files of various sizes. We use httperf [14] to send client requests for web document of size 1kB, 10kB, 30kB, 50kB or 70kB. Authors of [4, 18] showed that web server performance is CPU bound under a mix of small size files, and is network bound under a mix of large files. The criteria for small or large files depend on the

capacity of the machine. For our experimental setup, files with size larger than 10K are network bounded.

Performance Metrics. The following metrics are used in our measurement study. They are collected using Xenmon [7] and Xentop [24].

- **Server throughput** (#req/sec). It quantitatively measures the maximum number of successful requests served per second when retrieving web documents.
- **Normalized throughput**. We typically choose one measured throughput as our baseline reference throughput and normalize the throughputs of different configuration settings in order to make adequate comparison.
- **Aggregated throughput** (#req/sec). We use aggregated throughput as a metric to measure the impact of using varying number of VMs on the aggregated throughput performance of a physical host.
- **CPU time per execution** (μs/exe). It is a performance indicator that shows the average obtained CPU time in microseconds (μs) during each run of the given domain.
- **Execution per second** (#exe/sec). It measures the number of guest domains being scheduled to run on a physical CPU during one unit time.
- **CPU utilization** (%). To understand the CPU resource sharing across VMs running on a single physical machine, we measure the average CPU utilization of each VM, including Domain0 CPU usage and guest domain CPU usage respectively.
- **Network I/O per second** (kByte/sec). We measure the amount of network I/O traffic in kB per second, transferred to/from a remote web server for the corresponding workload.
- **Memory pages exchange per second** (pages/sec). We measure the number of memory pages exchanged per second in the I/O channel. It 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 exchanged during each run of the given domain.

E. Measurement Study: Objectives and Methodology

In a virtualized cloud environment, cloud providers implement server consolidation by slicing each physical machine into multiple virtual machines (VMs) based on server capacity provisioning demands. Cloud consumers may reserve computing resources through renting VMs from cloud providers. However, there has not been much dedicated study on the ways that multiple VMs hosted on the same physical machine may impact on the performance of the applications running on them, including when some of these VMs are idle, some are running CPU intensive workloads or I/O intensive applications.

With this problem in mind, we design our measurement study on the following two important issues: (i) understanding the impact of idle instances on application performance; (ii) understanding the impact of co-locating applications in a virtualized cloud in terms of throughput performance and resource sharing effectiveness. Through in-depth measurement analysis of these issues, we can better understand the set of key factors that can maximize the physical host capacity and the application performance. In addition, cloud service providers can provide more effective management of virtual machines to better meet consumers' demand, and at the same time, cloud consumers can utilize the insights gained from this study to manage and scale their applications more effectively.

III. DEALING WITH IDLE INSTANCES

Consider a set of n (n>0) VMs hosted on a physical machine, at any point of time, a guest domain (VM) can be in one of the following three states: (i) execution state, namely the guest domain is currently using CPU; (ii) runnable state, namely the guest domain is on the run queue, waiting to be scheduled for execution on the CPU; and (iii) blocked state, namely the guest domain is blocked and is not on the run queue. A guest domain is called idle when the guest domain is executing idle-loop, i.e., there is no VM that is not blocked and not idle.

In this group of experiments, we intent to study the following two issues: First, we want to understand the advantage and drawbacks of keeping idle instances from the perspective 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, its impact on existing applications, and the key factors impact the application performance and new domain start-up time.

We setup the first set of experiments in two steps. First, we use one single guest domain, denoted as Domain1, to serve all http requests. We stress Domain1 with as high workload as possible to find out its service limit. Then, we varied the number of idle guest domains in addition to Domain1 from zero to three. Note that, the Apache web server starts automatically in the idle domain to simulate the situation that an instance which has been booted up can respond to requests immediately. This is a more practical way to simulate the real world scenario. Table I shows the results of four experimental setups. Domain1 is running I/O application in high workload rate, with zero, one, two, or three other VMs in idle execution state. Each setup records the maximum achievable throughputs for all five I/O applications (1kB, 10kB, 30kB, 50kB and 70kB).

TABLE I. MAXIMUM THROUGHPUT FOR DOMAIN1 [#REQ/SEC]

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

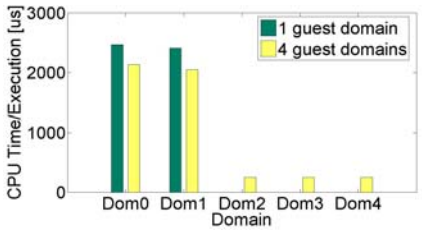

Figure 1. CPU time per execution [μs/exe] for 1kB application under 1 VM and 4 VMs with 3 idle VMs setups

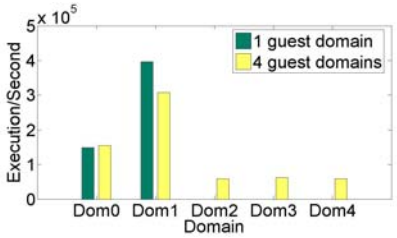

Figure 2. Exeution counts per second for 1kB application under 1 VM and 4 VMs with 3 idle VMs setups

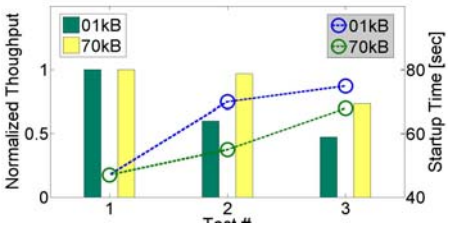

Figure 3. Throughputs for 1kB and 70kB applications and startup time [s] of one and two new guest domains.

We observe some interesting facts from Table I. First, there is no visible performance impact of keeping the idle domain(s) running when the running domain are serving 30kB, 50kB and 70kB applications, because their performance are network bounded. Second, the worst performance degradation occurs in the 1kB application, which is CPU bound. Compared with the single VM setup where the highest throughput value achieved is 1070 #req/sec, we see about 6% performance degradation when the number of guest domains is four (999 #req/sec). It is apparent that adding idle guest domains induced overhead which can impact the performance of CPU intensive applications in the running domain.

Figure 1 and Figure 2 present more detailed measurements of performance impact of idle instances, which helps us to quantitatively characterize the overhead occurred for 1kB application. We measured the CPU time per execution as well as the number of execution counts per second with one VM and four VMs with three idle setups for 1kB application. From Figure 1 and Figure 2, we make two observations. First, on average, each of the three idle guest domains can get about 250μs for each run, which is about only 10% of the CPU time of Domain0 for each execution. Second, comparing 4 VMs with 3 idle VMs setup with single VM setup, we see the CPU time for each run is dropped from 2464 μs to 2407 μs in domain0 and from 2130μs to 2046 μs in domain1, and similarly, the execution count per second is dropped from 400,000 to 300,000 in domon1, though the execution count per second in domain0 sees a slight increase. The drop in CPU time per execution and execution per second is primarily due to the following two factors: (1) the execution of timer tick for the idle guest domain and the context switch overhead, and (2) the processing of network packets such as address resolution protocol (ARP) packets, which causes I/O processing in guest domain.

Now we report the second set of experiments in this group. In this set of experiments, we want to study how CPU intensive applications and network I/O intensive applications may impact the throughput performance when an idle instance is present. We also want to understand the startup time for creating one or more new guest domains on demand and the other factors that may impact such start-up time. We adopted the methodology as follows. Domain1 is serving the 1kB or 70kB applications alone. Then we create one or two idle instances. Figure 3 records the fluctuations in Domain1's throughput and startup time for the idle guest domains. It shows the throughput for running Domain1 alone (Exp1), running Domain1 with startup one VM on demand (Exp2), and running Domain1 with startup two VMs on demand (Exp3), respectively. In this set of experiments, the request rate is fixed at 900 requests/second for the 1kB application or 150 requests/second for the 70kB application, both of which are approaching 90% of maximum throughput values given in Table I. The primary y-axis is the normalized throughput with 900 successful requests/sec for 1kB application or 150 #req/sec for 70kB application as baseline. The second y-axis denotes the start-up time (sec) for one, two or three VMs. Note that the circles in Exp1 denote the startup time for one single instance without running Domain1.

Figure 3 shows 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 2GB guest domain instance. As the measurement results showed, to start up a new VM, the average CPU consumption is about 20%. The peak CPU consumption to finish this task can be as high as 75%. In addition, it requires about 900 virtual block device read operations and about 200 virtual block device write operations. These I/O related activities to start-up new domains cannot be finished without the presence of Domain0, which plays a key role in processing Domain1's web workloads. Our second observation is that the 70kB application suffers less in terms of start-up time than the 1kB application. This is because the performance of the 70kB application is network bounded, and it consumes less CPU, which alleviates the CPU contention. In our case, it will consume about 90% CPU resources in addition to about 5400 memory page exchanges per second between Domain0 and Domain1 to serve the 900 requests/sec for 1kB application. In contrast, only 60% CPU resource is reserved to serve 150 requests/sec for the 70kB application. Furthermore, for the 1kB application, the startup time for creating two guest domains in Exp3 grows from 47 sec in Exp1 to 75 sec, which is about 1.5 times bigger. In contrast, for 70kB application, the difference in start-up time from creating two VMs to one VM is relatively smaller. This shows 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 start-up time than network I/O bounded applications. Finally, our third observation is that the duration of performance degradation experienced due to creating new VMs on demand is typically bounded within 100 seconds in our experiments. Our experience shows that, the experienced duration of performance degradation is related with the machine capacity, the workload level in the running domain, and the number of new VM instances to start up.

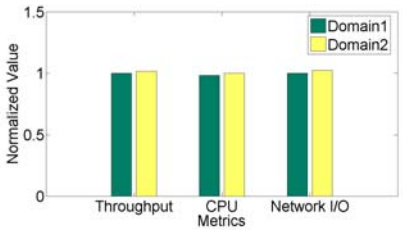

Figure 4. Normalized throughput, CPU utilization and Network I/O between Domain1 and Domain2, both with identical 1kB application at 50% workload rate

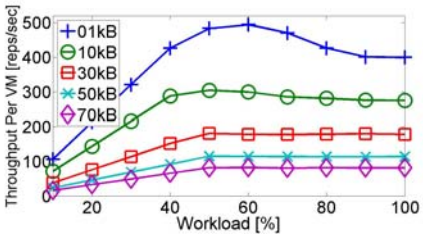

Figure 5. Average throughput [#req/sec] per guest domain, with both guest domains running identical application at the same workload rate

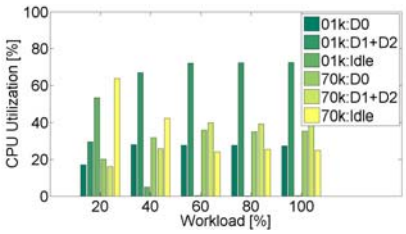

Figure 6. CPU usage for Domain0, aggregated CPU usage for guest domains, and percentage of idle CPU [%]

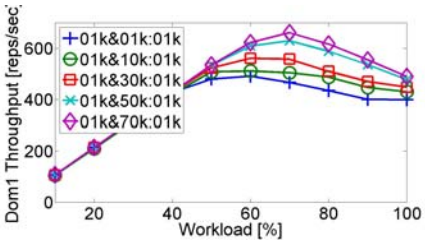

Figure 7. Domain1 throughput when Domain1 is serving 1kB appliaction and Domain2 is serving 1kB to 70kB applications

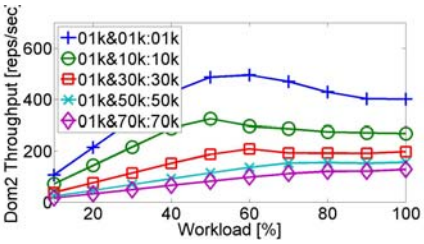

Figure 8. Domain2 throughput when Domain1 is serving 1kB application and Domain2 is serving 1kB to 70kB applications

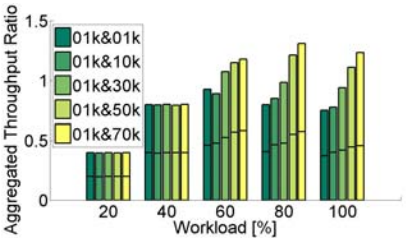

Figure 9. Aggregated throughput ratio for Domain1 and Domain2 across five applied workload rates

IV. Impact of Neighbor Application

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 impact of co-locating patterns of applications, we choose the five I/O applications of 1kB, 10kB, 30kB, 50kB and 70kB in our experiments, but divide the experiments into two 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.

A. Co-locating Identical Applications

In this group of experiments, we design two guest domains, Domain1 and Domain2, 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.

Figure 4 shows the experimental results for two VMs when both are serving 1kB applications with 50% workload rate. We measured throughput, CPU utilization, Network I/O. For example, Domain1 consumes 36.1% CPU resources while Domain2 consumes 36.8% CPU resources. The throughputs and network bandwidths for Domain1 and Domain2 are: 480 #req/sec and 487 #req/sec, 609 kByte/sec and 622 kByte/sec respectively. We present these three metrics in normalized values to show their similarities. For each metrics pair, we use the value for Domain1 as the comparative baseline. In Figure 4 the difference between the measurement in VM1 and the measurement in VM2 is trivial and can be ignored.

Figure 5 measures the average throughput of Domain1 and Domain2 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%, (2) there is crucial difference between small-sized file application and large-sized file application. For small-sized file application such as 1kB and 10kB, obvious performance degradation can be observed at workload rates higher than 50% or 60%. However, this is not the case for large-sized file applications. The significant skew happened in the 1kB 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 I, the overhead has increased due to the network bridging happened in Domain0, and the context switch.

Figure 6 measures the CPU usages for 1kB and 70kB applications under varying workload rates. We add up CPU used by Domain1 and Domain2 together since the results in Figure 4 indicate that Domain1 and Domain2 always get almost the same amount of CPU allocation. Figure 6 shows: under the same workload rate, the guest domain CPU usage for 1kB file is much larger than that of the 70kB application, despite the fact that the memory page exchange rate for 1kB file is much less than that of the 70kB application. This is because 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, the connection phase demands significantly more CPU resources than the transportation phase.

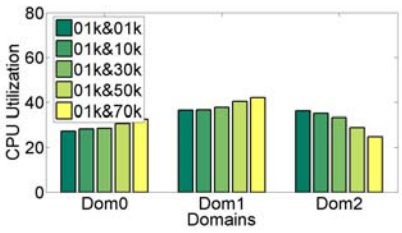

Figure 10. CPU utilization for Domain1 is serving 1kB with Domain2 is serving 1kB to 70kB when the applied workload is 100%

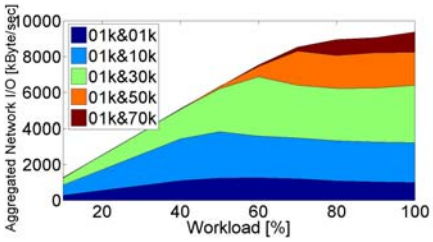

Figure 11. Aggregated Network I/O when Domain1 is serving 1kB application and Domain2 is serving 1kB to 70kB applications

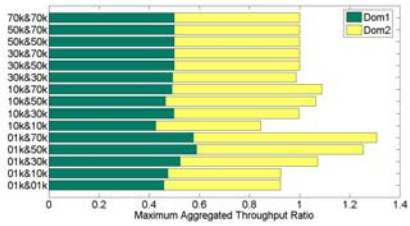

Figure 12. Maximum aggregated throughput ratio for all combinations with respect to dual guest domain tests

B. Co-locating Different Applications

From experimental results in the previous subsection, 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 co-located on the same physical host are applied workload rates and resource usage patterns of applications. In this subsection 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 Domain1 and serving the 1kB application, the other is using Domain2, running the application, which is by design varying from 1kB to 70kB.

Figure 7 and Figure 8 measure the throughputs for Domain1 and Domain2 under the 70% workload respectively. We observe two interesting facts: (1) although Domain1 always serves the 1kB file, its performance highly depends on the application running in its neighbor Domain2. For example, in the 1kB and 70kB combination (661 reqs/sec for 1kB) compared with in the 1kB and 1kB combination (494 reqs/sec for 1kB), the performance difference can be 34%. (2) The highest throughput points occurring in Figure 7 and Figure 8 show considerably different tendencies. Take the 1kB and 70kB 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 1kB file appears at 70% workload rate, while it comes at 100% workload for the 70kB application. Clearly, this phenomenon is due to the resource usage pattern of 1kB and 70kB applications, 1kB is CPU bounded and 70kB is network bounded.

Figure 9 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 I as the baseline to get individual throughput ratio for each guest domain under each specific workload. For example, the throughput for Domain1 is 661 #req/sec under 70% workload, thus the throughput ratio is about 62% (i.e., 661/1070). Similarly we have the aggregated throughput ratio of 130% for the 70kB application. From the results for five combinations of neighboring applications in Figure 9, we observe that the best co-locating case is the 1kB and 70kB combination with aggregated throughput ratio of 1.3, and the worst case is the 1kB and 1kB combination with aggregated throughout ratio of 0.92, The performance difference could be more than 40% ((1.3-0.92)/0.92=41%).

Figure 10 measures the CPU usages of Domain0, Domain1, and Domain2 for five different combinations of applications. We make three interesting observations from this set of experiments: (1) The decrease in Domain2 CPU usage is expected when the application in Domain2 is resized from 1kB to 70kB and from CPU-intensive to network intensive application. (2) The increase in Domain0 CPU utilization is expected when the neighbor application combination is changed from 1kB and 1kB to 1kB and 70kB, this is because for the 70kB application, the large amount of data transferred cause higher consumption of the device driver in Domain0. (3) The increase in Domain1 CPU utilization is also expected since it explains the performance improvement occurred for the 1kB application in Figure 7, when domain2 is changing to pairing with 70kB network bounded application, allowing the released CPU in domain2 to be utilized by the CPU intensive 1kB application in domain1.

Figure 11 plots the aggregated network I/O consumption as a function of workload rates across five different combinations of neighboring applications. We observe that the 1kB and 70kB application combination consumes the highest network bandwidth. This experimental result is consistent with the results in Figure 9 and Figure 10. For the 1kB and 70kB application combination, while the majority of requests (80%) processed are the 1kB file retrieval requests (see domain1 CPU utilization for 1kB&70kB), the majority of network bytes consumed (94%) are due to transferring the 70k file objects.

Figure 12 shows the maximum aggregated throughput ratio for all the possible combinations with two guest domains and five applications of different file sizes. Our goal for this experiment is to comprehensively examine the performance impact of different types of application combinations. This is a super set of the experiments in Figure 9 where five combinations are examined. We draw three insights from Figure 12: (1) If we co-locate two CPU intensive applications, such as 1kB and 10kB combination, then the performance of both applications will suffer. (2) If we choose the network intensive application combinations such as 30kB and 50kB, then each will equally contribute 50% of the aggregated throughput performance, and there is no performance degradation. (3) The best co-locating strategy is to use the CPU intensive and network intensive application combination, which always achieves strikingly high aggregated throughput.

C. Co-locating Applications among Multiple VMs

We have studied the impact of co-locating 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 co-location strategy.

Our first set of experiments is designed by varying the number of guest domains from one to six and each guest

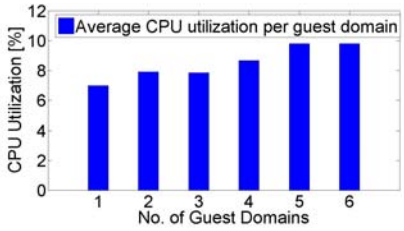

Figure 13. Average CPU utilization for each VM when varying the number of VMs from one to six, each is serveing 10% workload rate

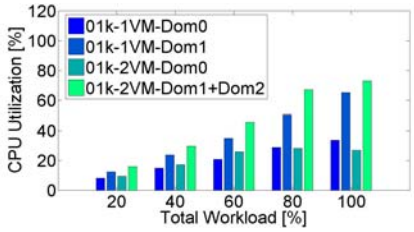

Figure 14. CPU usage by one and two guest domains with varying workload rates

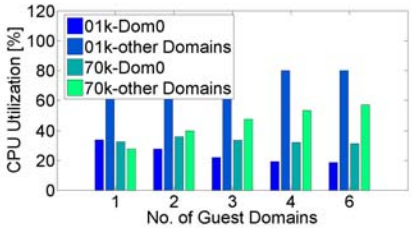

Figure 15. CPU usage for one, two, three, four and six guest domains with 120% workload rate

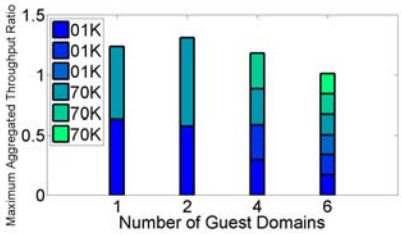

Figure 16. Aggregated throughput ratio for one, two, four and six VMs, half serving 1kB files, the other half serving 70kB files.

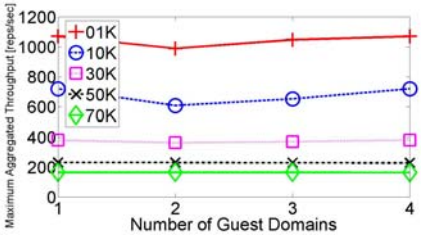

Figure 17. Throughputs for one, two, three, four guest domains with repsect to with repsect to five applications

domain serves 10% applied workload. Total workload rate can be calculated by multiplying the number of guest domains with the applied workload rate. Using 10% workload applied to each guest domain guarantees no severe resource contention will occur. Figure 13 shows when there are six guest domains running, the CPU time spent (9.8%) to process the same amount of I/O data (10% workload per guest domain) equals to 1.5 times of the CPU time spent (6.9%) in the single guest domain case. This group 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 [12], 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%. The 1kB and 70kB application are chosen as they are the two representative applications. Figure 14 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 dom0 CPU utilization for 1kB application running on a single VM. 01k-2VM-Dom0 denotes the measurement of dom0 CPU utilization for 1kB application running on two VMs. 01k-2VM-Dom1+Dom2 measures the combined CPU usage of both Domain1 and Domain2 for 1kB 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, Domain0 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 Domain0 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%.

Figure 15 shows the CPU usages under high contention situation. We varied the total workload rates to 120%. As seen from the figure, when the number of guest domains grows from one to six, the CPU share for Domain0 reduces at a more gradual rate for the 70kB application (32.5% to 31.3%). In contrast, when the number of guest domains is changed to six, the CPU utilization at Domain0 for the 1kB application is reduced from 33.8% to 18.6%. For the 1kB application, the significant reduction in Domain0 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 Domain0.

Figure 16 measures the impact of the number of guest domains on aggregated throughput ratio with half of guest domains serving the 1kB application and the other half serving the 70kB application. We use the maximum throughput of single VM for five applications in the first column of Table I as the baseline to get individual throughput ratio for each guest domain. The first observation is: The configuration for two guest domains outperforms than other configurations. When the number of guest domains switches to four or six, the aggregated throughput ratios reduce to one. This means: when the number of guest domains expended beyond some value, which is four in our case, the overhead raised by hosting multiple guest domains depletes the benefits of the best combination. Compared to the six guest domain case (1.01), the best case (1.31) when there are two guest domains shows about 30% performance gains.

Figure 17 measures the maximum overall throughput performance for five applications, each is distributed on multiple guest domains. This set of experiments shows how the application characteristics combined with the number of guest domains together may affect the overall throughput performance. For the 10kB application, the maximum aggregated throughput is 608 #req/sec when two guest domains

are running, compared with 720 #req/sec achieved in one guest domain experiment, showing 15% performance decline. When three guest domains are present, the aggregated throughput is 652 #req/sec. When compared with one single guest domain case, 10% performance decline has shown. The slight throughput increase for the case of four guest domains is due to the global load balancing capabilities of the credit scheduler [3].

V. CONCLUSIONS

To maximize the benefit and effectiveness of server consolidation and application consolidation in virtualized cloud environments, we argue that it is important to conduct in-depth performance measurements for applications running on multiple VMs hosted on a single physical machine. Such measurements can provide quantitative and qualitative analysis of performance bottlenecks that are specific to virtualized environments, offering deeper understanding of the key factors for effective resource sharing among applications running in virtualized cloud environments. We have presented our performance measurement study of network I/O applications in virtualized cloud environments. We focus our measurement based analysis on performance impact of co-locating applications in a virtualized cloud in terms of throughput performance and resource sharing effectiveness, including the impact of idle instances on applications that are running concurrently on the same physical host, and how different CPU resource scheduling and allocation strategies, and different workload rates may impact the performance of a virtualized system.

In this work, we showed that by strategically co-locating network I/O applications together, considerable performance gain could be obtained. However, we did not show how to utilize this strategy to help decision making in the cloud. In the future, we plan to utilize our findings as knowledge base to facilitate the scheduling in the cloud. Another limitation of this research work is we conducted our experiments on Xen platform. We are going to repeat some of these experiments in other virtual machine monitors.

ACKNOWLEDGMENT

This work is partially supported by grants from NSF CISE NetSE program, NSF CISE CyberTrust program, and an IBM faculty award, an IBM SUR grant, and a grant from Intel Research Council. The first author performed this research as a visiting PhD student at the Distributed Data Intensive Systems Lab (DiSL) in the School of Computer Science, Georgia Institute of Technology, supported by China Scholarship Council and Department of CS in Xi'An Jiaotong University.

REFERENCES

[1] 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, ACM Symposium on Operating Systems Principles (SOSP), 2003.

[2] Z. Chen, D. Kaeli, and K. Murphy, Performance Evaluation of Virtual Appliances, First International Workshop on Virtualization Performance: Analysis, Characterization, and Tools (VPACT 08), 2008.

[3] L. Cherkasova, D. Gupta, A. Vahdat, Comparison of the Three CPU Schedulers in Xen, ACM SIGMETRICS Performance Evaluation Review, Vol. 35, Issue 2, September 2007. pp. 42-51.

[4] L. Cherkasova, R. Gardner. Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor. USENIX Annual Technical Conference, pp: 24-24, 2005.

[5] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, J. N. Matthews, Xen and the Art of Repeated Research, Annual Conference on USENIX Annual Technical Conference 2004.

[6] Credit Based Scheduler. http://wiki.xensource.com/xenwiki/.

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

[8] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, Enforcing Performance Isolation across Virtual Machines in Xen, Middleware 2006, LNCS 4290, pp. 342-362, 2006.

[9] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, A. Sivasubramaniam, Xen and Co.: Communication-aware CPU Scheduling for Consolidated Xen-based Hosting Platforms, VEE 07, pp. 126-136.

[10] H. Kim, H. Lim, J. Jeong, H. Jo, J. Lee, Task-aware Virtual Machine Scheduling for I/O Performance, VEE 09, pp. 101-110.

[11] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu. An Analysis of Performance Interference Effects in Virtual Environments. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2007, pp. 200-209.

[12] A. Menon, J.R. Santos, Y. Turner, G.J. Janakiraman, and W. Zwaenepoel, Diagnosing Performance Overheads in the Xen Virtual Machine Environment, ACM/USENIX International Conference on Virtual Execution Environments, VEE 05, 2005, pp. 13-23.

[13] A. Menon, A. L. Cox, W. Zwaenepoel, Optimizing Network Virtualization in Xen, 2006 USENIX Annual Technical Conference.

[14] D. Mosberger, T. Jin. Httperf-A Tool for Measuring Web Server Performance. ACM SIGMETRICS Performance Evaluation Review, Volume 26, Issue 3, December 1998. pp. 31-37.

[15] Naoki Nishiguchi: Evaluation and Consideration of the Credit Scheduler for Client Virtualization, http://www.xen.org/xensummit/xensummit_fall_2008.html

[16] D. Ongaro, A. L. Cox, S. Rixner, Scheduling I/O in Virtual Machine Monitors, Fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE 08), pp. 1-10.

[17] P. Padala, X. Zhu, Z. Wang, S. Singhal, K. G. Shin. Performance Evaluation of Virtualization Technologies for Server Consolidation. HP Laboratory, Palo Alto, April 11, 2007. http://www.hpl.hp.com/techreports/2007/HPL-2007-59R1.pdf

[18] F. Prefect, L. Doan, S.Gold, and W. Wilcke. Performance Limiting Factors in Http (Web) Server Operations. Proc. of COMPCON'96.

[19] K. K. Ram, J. R. Santos, Y. Turner, A. L. Cox, S. Rixner, Achieving 10 Gb/s using Safe and Transparent Network Interface Virtualization, VEE 09, pp. 61-70.

[20] G. Somani and S. Chaudhary, Application Performance Isolation in Virtualization, IEEE Int. Conf. on Cloud Computing, 2009.

[21] T. Wood, L. Cherkasova, K. Ozonat, and Prashant Shenoy, Profiling and Modeling Resource Usage of Virtualized Applications, Middleware 2008, LNCS 5346, pp. 366-387, 2008.

[22] http://aws.amazon.com/ec2/

[23] http://code.google.com/appengine/

[24] http://linux.die.net/man/1/xentop