Performance Evaluation of Virtual Appliances

Zhaoqian Chen and David Kaeli
Electrical and Computer Engineering Department
Northeastern University, Boston, MA
{stchen,kaeli}@ece.neu.edu

Kevin Murphy
Network Engines
Canton, MA
kevin.murphy@networkengines.com

Abstract

Virtualization technology has become standard in many computing environments. Independent Software Vendors (ISVs) are now using virtual machines to deliver multiple software appliances. However, the performance of these appliances when run on virtualized platforms is not well understood. In this paper, we present a performance evaluation framework for virtualized appliances. We are using the XenServer virtual machine system and evaluate its performance on four different classes of workload: 1) voice-over-IP, 2) TCPIP networking, 3) IO disk intensive, and 4) CPU intensive. Our primary interest is understanding the scalability of the VoIP application in the presence of the other three classes of workload. We also consider a range of different virtual machine configurations.

Our experiments show that the XenSource VMM can isolate hardware resource utilization (i.e., disk and network I/O) nicely across multiple virtual machines. But when running multiple virtual appliances concurrently, sharing physical CPUs across virtual machines introduces significant overhead. We present the results of this study and consider future work to address some of these performance issues.

1. Introduction

Today, virtualization technology plays a critical role in many computing enterprises. Virtualization can provide lower cost of ownership, improved cost/performance, and higher performance/watt. Virtualization was first popularized on mainframe-class systems, though it recent rise in popularity is tied to its ability to share hardware infrastructure across multiple virtual machines run on commodity hardware.

Server consolidation can help enterprises achieve higher system utilization, reduce complexity and total cost. The leading virtualization technology vendors such as Xen [1] and VMware ESX [2], along with the development of hardware support for virtualization from Intel [3] and AMD [4], are accelerating the move to deploying virtual machines in production appliances (i.e., virtual appliances).

Virtualization technology has been used effectively for server consolidation and optimization of hardware resources. When enterprises deploy virtualization, they also enjoy the benefits of a number of associated features that include: VM check-pointing, physical to virtual (P2V) migration, data management, and others. They pay less attention to how their applications are performing in this virtualized environment. When a virtualized appliance runs on the same platform with other virtualized appliances, we need to pay careful attention to the associated performance impact. For some near real-time applications (e.g., telephony server, game server, etc.), performance and latency are crucial and must be managed carefully.

There are two main approaches to virtualization. *Full virtualization* allows an unmodified operating system to run on top of a hypervisor without any awareness of the underlying hypervisor layer. The hypervisor is a customized operating system layer that provides for virtual machine management. When using full virtualization, the execution overhead is typically larger than on systems that use *paravirtualization* [5]. Paravirtualization runs a modified operating system that is fully aware of the hypervisor and cooperates in the virtualization process. The complexity and performance of each virtualization approach can vary greatly, though full virtualization seems to be gaining in popularity in server-class applications.

While some commercial studies have compared their systems, little has been done by the performance evaluation community to analyze the performance and scalability of these environments. The main goal of this work is to gain a better understanding of the overhead associated with virtualization by utilizing workloads that place load on different subsystems on the virtualized system. We measure a number of performance metrics to identify the overhead involved with different virtualization configurations on a virtualized version of a software appliance.

The rest of this paper is organized as follows. In the next section we present our profiling framework. Section 3 introduces the XenSource virtualization architecture that is used in this paper. Section 4 describes our synthetic workload used. In Section 5, we present performance results. Finally, Section 6 concludes the paper and presents directions for future work.

2 Evaluation Framework

Our Virtual Appliance (VA) performance evaluation framework is shown in Figure 1. The framework consists of two separate systems: 1) the VA tester/monitor and 2) the target platform (including the Virtual Appliances and VMM) under test. The VA tester/monitor software contains a VMM level profiler, client-based VA workload drivers to drive each VA, and a management user interface (UI). The VMM profiler probes the VMM to provide VM-level information, such as VM status, virtual/physical CPU utilization, VMM scheduling statistics, virtual I/O speed, etc.

A number of different VA workloads are configured on the target system to act as typical virtual appliances. The VA tester/monitor drives the different VAs with load. We then obtain VMM statistics and compile these back on the VA tester/monitor system. Using these workloads and the VMM profiler together, we are able to analyze each VA's performance. The management UI is designed to allow the tester/monitor system administrator to monitor VA performance performance through a graphical presentation.

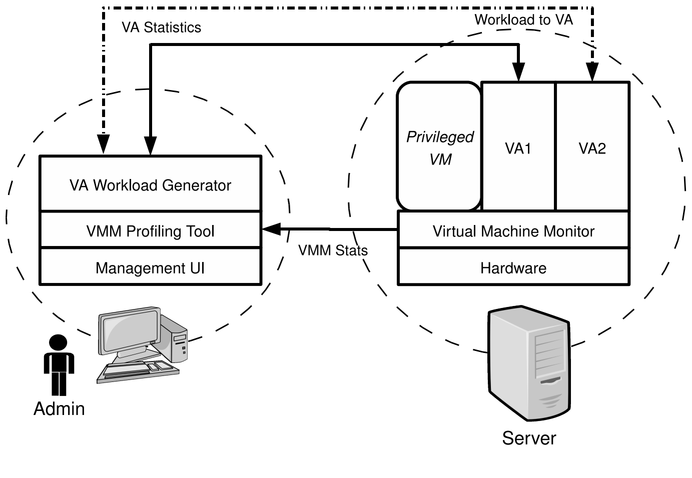


Figure 1. VA Performance Evaluation Platforms

Our VA test/monitor is presently configured on a separate system in order to eliminate any overhead. The VA tester/monitor could be integrated into the platform to reside on a privileged VM, as shown in Figure 2. This privileged VM has full access to the VMM so that our VA tester/monitor is still able to profile VMM stats and all other VMs. Thus, future platform vendors will be able to produce a single box with VAs installed and with the VA performance tester/monitor integrated.

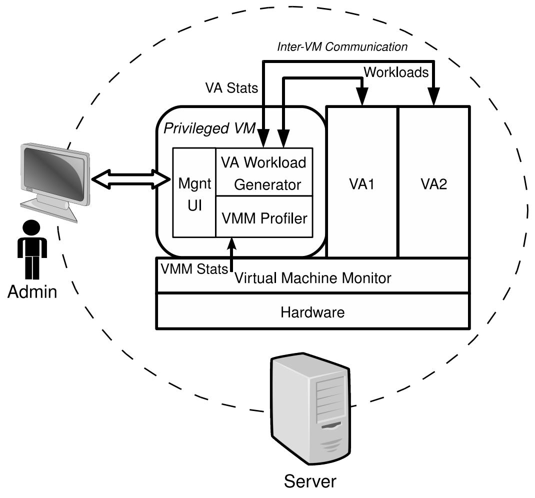


Figure 2. Integrated VA performance monitor and evaluation.

3 Implementation

We have implemented this framework on a XenServer virtual machine. The main appliance we are targeting is AsteriskNOW [6], which is a well-known telephony server appliance running the SIP protocol. In this section, we describe the platform and appliance being tested.

3.1 The Xen Architecture

Xen is a free and open-source virtual machine monitor which allows multiple guest operating system instances to run concurrently on a single physical machine. It supports both para-virtualization and full virtualization. The Xen virtualization platform has multiple layers. Figure 3 provides an overview of the Xen architecture.

The lowest and most privileged layer is the Xen VMM, which presents a virtual hardware abstraction slightly different from the underlying hardware. Each guest operating system (OS) runs on a separate virtual machine (or guest domain in Xen terminology). The domains are scheduled by the Xen VMM to make effective use of the available physical CPUs. Each application, running in the guest OS, accesses hardware devices via a special privileged virtual machine called Domain-0 (or IDD, isolated driver domain). This privileged VM owns specific hardware devices and talks to the I/O device drivers. All other guest domains use a simple device driver that communicates with Domain-0's driver to access real hardware devices. Domain-0 can directly access the hardware devices it owns. However, a guest domain accesses the hardware devices indirectly through a virtual device connected to Domain-0. Domain-0 maps through bridges and routes each physical interface through page-sized buffers which are transferred over the I/O channel. This helps to avoid copying during execution.

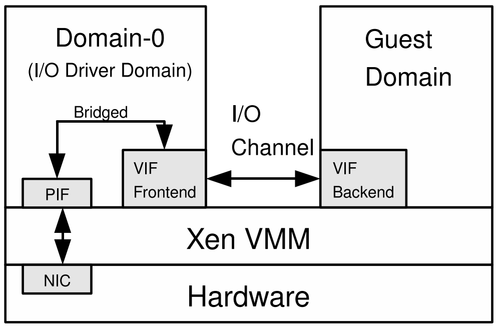


Figure 3. Xen Architecture

3.2 An Overview of the SIP Protocol

The Session Initiation Protocol (SIP) [7] is an application layer control protocol for creating, maintaining, and tearing down sessions for various types of media, including voice, video, and text. SIP is growing in importance as it is being used for many media-oriented applications such as Voice-over-IP (VoIP), voicemail, instant messaging, telepresence, IPTV, network gaming, and more. It is also the core protocol for the IP Multimedia Subsystem, the basis for the 3rd-Generation Partnership Program for both fixed and wireless telephone networks. SIP relies on an infrastructure of servers, which are responsible for maintaining the location of users and forwarding SIP messages across the application-layer SIP routing infrastructure toward their eventual destinations. The performance of these SIP servers is thus crucial to the operation of the infrastructure, as they can have a primary impact on the latency of media applications (e.g., for initiating a phone call). Service providers clearly require performance information to understand how to provision their infrastructure to provide reasonable QoS.

Figure 4 shows a basic stateful SIP scenario without authentication. The SIP client wishes to establish a session via the server and sends an INVITE message to the server. Since the scenario is stateful, it responds with a TRYING message to inform the client that the message has been received. The server generates an OK message, which is sent to the client. The client then generates an acknowledgment message. The session is now established. When the conversation is finished, the client user hangs up and generates a BYE message. The server side responds with an OK to the user.

Performance is measured by clients using the SIPp workload generator [8], which send packets over a 100Mb Ethernet. SIPp provides for a wide range of SIP scenarios to be tested, such as user-agent clients (UAC), user-agent servers (UAS) and third-party call control. SIPp is also extensible by writing third-

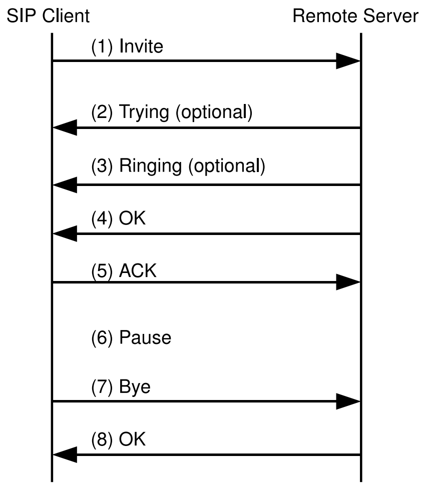


Figure 4. Basic stateful SIP scenario without authentication.

party XML scripts that define new call flows. SIPp has many run-time options, such as multiple transport (UDP/TCP/TLS) support and MD5-based hash digest authentication.

4 Experimental setup

The hardware used in our experiments is an Intel Dual-Core CPU Xeon 3060 that supports Intel VT [3], running at 2.4GHz with 4MB L2 Cache and 4GB main memory. We use XenServer 4.0.1 which is an enterprise version of Xen from Citrix, Inc [9]. XenServer consists of the Xen VMM, management software, and a number of service features (P2V conversion, migration, built-in guest templates and etc.). The Xen VMM profiling tool we created is written in Python and utilizes the XenAPI [10] to collect VMM statistics (e.g., CPU utilization, network interface speed). The sampling granularity is adjustable.

Two VMs are setup for all of our experiments. We instantiate either two identical Asterisk VMs, or one Asterisk VM and one Debian-based VM (described later). Each VM is assigned 1GB of memory and sufficient disk space. The VM configurations being tested are shown in Table 1.

We are using AsteriskNOW version 1.0 (we will refer to this VM as the “Asterisk VM”) running under hardware-assisted Virtual Machine mode. AsteriskNOW includes a copy of the rPath Linux operating system [11] and the Asterisk telephony server application. A Debian Etch (4.0) Linux operating system created from XenServer’s built-in templates pool is run on another VM (we label this second VM as the “Debian VM”). We run a number of workloads on the Debian VM, each stressing a different hardware resource.

Super Pi [12] is a CPU-intensive workload that calculates Pi to a specified number of digits after the decimal point. Nttcp [13] is a network-intensive workload that performs TCP

No.	# of VM	VCPU/VM	Description
1	1	1	Only Asterisk VM
2	1	2	Only Asterisk VM
3	2	1	Asterisk VM and Debian VM (CPU-intensive workload)
4	2	2	Asterisk VM and Debian VM (CPU-intensive workload)
5	2	1	Asterisk VM and Debian VM (Network-intensive workload)
6	2	1	Asterisk VM and Debian VM (Disk-intensive workload)
7	2	1	Two identical Asterisk VM working at the same time

Table 1. Different VM configurations considered in this work.

throughput testing. This workload starts a network connection between two systems and then transfers data. Both sides measure the time and number of bytes transferred. IOzone [14] is a disk-intensive workload that generates and measures a variety of file operations with both random and sequential operations.

We run the SIPp workload from another independent computer system. The SIPp program is configured as a UAC client that sets up a socket and employs the default UDP port 5060 to send multiple messages to the server. The SIP scenario is a stateful proxying without authentication. This configuration can achieve fairly good performance [15]. We are interested not only in the maximum capacity of the server (i.e., throughput), but also how the application behavior degrades under excessive load. We gradually increase the workload (calls per second), making a slow transition from non-overloaded to overloaded status.

5 Performance Analysis

The statistics collected by our profiling tools include physical CPU (PCPU) utilization, virtual CPU (VCPU) utilization for all VMs, and physical network (PIF) I/O speed. The SIPp benchmark provides statistics about the number of successful/failed calls, response time distributions, and retransmission statistics.

5.1 Modeling a Single Asterisk VM

Figure 6 shows the resulting performance of running the Asterisk VM with only one VCPU. We find that this VCPU gets pinned to a single physical CPU by the Xen VM scheduler. The other physical CPU is used by Domain-0 to handle I/O processing. Running in a stable (non-overloaded) environment, the total network speed is around 2MB/s. We have found that when the Asterisk server reaches the maximum throughput, the CPU usage of the Asterisk application is about 70%. After the server is overloaded, it fails to process calls in time and starts sending out RTP traffic indicating that the server is currently too busy to respond. This impacts network traffic severely.

The CPU usage of the Asterisk VM decreases due to less call processing work to perform. But the CPU used by Domain-0 is impacted since it is encountering extra network traffic. For our experimental environment, the maximum throughput is around 700 calls per second (CPS).

When the Asterisk VM is assigned 2 VCPUs, we see the impact on the two physical CPUs which will have almost the exact same utilization, as shown in Figure 7). This means the total workload is being balanced across both CPUs by the Xen VM scheduler. The two virtual CPUs of the Asterisk VM are floating on the two physical CPUs. Since there are only two physical CPU cores on the system, the VCPUs of the Asterisk VM compete with the VCPU of Domain-0. From the results shown in Figure 7e, the performance (maximum throughput) is 600 CPS which is even lower than the previous configuration (1 VCPU for the Asterisk VM). The overhead of frequent CPU swapping and scheduling degrades call throughput by as much as 100 CPS.

5.2 Two Virtual Machines

One Asterisk VM and One Debian VM

Next we consider configuring two VMs. We assign a VCPU to each VM. Since two VMs are competing for hardware resources (i.e., shared memory and buses), the Asterisk application fails to achieve the same performance as the single-instance case of Asterisk (see Figure 8). The performance of the Debian VM running the *SuperPi* program is affected by the Asterisk VM as well. The total execution time of 231.4 seconds for *pi* computed to 2^{22} digits in a non-virtualized environment, versus the *Super Pi* program spends 253.8 seconds in the Debian VM, experiencing a 10% execution slowdown.

When we assign two VCPUs to each VM, the performance degrades even further and the maximum throughput is only about 100 CPS (see Figure 9). Since we have only a single dual-core CPU, the Xen VMM must virtualize a total of six VCPUs (each of the two VMs and Domain-0 each have two VCPU). Creating more VCPUs without considering the number of PCPUs can lead to poor performance.

Next, we configure the system so each VM has a single VCPU. We run the network workload nttcp in the Debian VM. We use another independent physical machine to transfer data to the Debian VM on the network while Asterisk is processing calls. This network-hungry Debian VM competes for network bandwidth with the Asterisk VM. Table 2 shows the performance of running the nttcp workload. In a virtualized environment, it only introduces a 5% overhead. The maximum throughput of the Asterisk VM is 650 CPS (see Figure 10), which is not far from the baseline (1 VM with 1 VCPU). The total network traffic is about 12 MB/s. From the previous statistics, we know that the network traffic just from the Asterisk VM is 2 MB/s, which means that network is not a limiting factor for the Asterisk VM.

To better understand how Asterisk competes for disk I/O, we ran the IOzone workload in the Debian VM. We configured the system identically to the last experiment, with the exception of replacing nttcp with IOzone. The IOzone workload obtains the same disk performance in both virtualized and non-virtualized environments. The overhead of disk latency is about 2% (378.85s vs. 385.35s). The maximum performance of the Asterisk VM is about 480 CPS.

Two identical Asterisk Virtual Machines

To complete this analysis space, we ran two Asterisk VM at the same time. From our testing results, we found that the system behavior and performance of two VMs are almost identical. The workload reaches a maximum throughput at 800 CPS as shown in Figure 12).

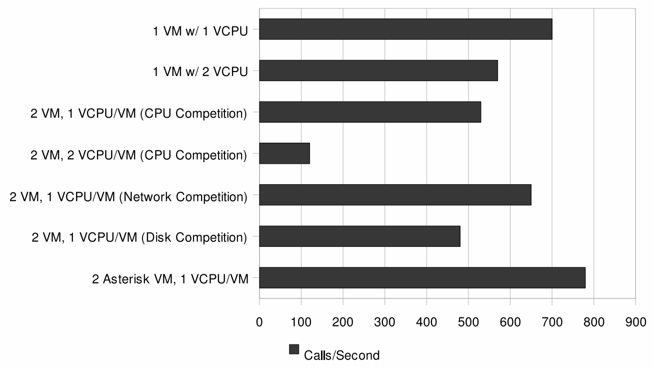


Figure 5. Performance comparison under different configurations.

Figure 5 compares of all the configurations we have tested. These experimental results tell us that for multiple CPU-intensive applications running on a virtualized platform, sharing CPUs introduces significant overhead and degrades the performance. One solution is to provide multiple CPUs so that each VM possesses its own VCPUs.

6 Related Work

SIPp, the workload generator we used, is a free Open Source test tool/traffic generator for the SIP protocol. It can read XML scenario files describing testing configuration files. It features a dynamic display of statistics (call rate, round trip delay, and message statistics), periodic CSV statistics dumps, TCP and UDP over multiple sockets or multiplexed with retransmission management, regular expressions and variables in scenario files, and dynamically adjustable call rates. Our analysis tool collects SIP data from SIPp log.

VMmark [16] was developed by VMware and is a tile-based benchmark consisting of several workloads running simultaneously in separate virtual machines. Each workload component is based upon a single-system benchmark running at less than full utilization. The performance of each workload is measured and used to form an aggregate score. The scores generated when running multiple tiles simultaneously can be summed to increase the overall benchmark score. However, this benchmark can only be used in VMware products, which lacks universality on other platforms.

We are hoping to utilize standardized tools and benchmarks, such as SPEC benchmarks. SPEC organization is currently working on both a SIP and a virtualization benchmark. The SPEC Virtualization Committee [17] is to deliver a benchmark that will model server consolidation of commonly virtualized systems, to provide a means to fairly compare server performance while running multiple virtual machines. SPEC SIP2008 [18] is designed to evaluate a system's ability to act as a SIP server supporting a particular SIP application. The application modeled is a VoIP deployment for an enterprise, telco, or service provider, where the SIP server performs proxying and registration.

Xenoprof [19] is a system-wide profiler for Xen virtual machine environments, capable of profiling the Xen virtual machine monitor with multiple Linux guest operating systems, and applications running on each of them. Xenoprof is modeled after the OProfile profiling tool available on Linux systems. It provides profiling data at a granularity of individual processes and routines executing in either of the virtual machines or in the Xen VMM.

Xenmon [20] is a performance monitor for profiling infrastructure. It captures how time is spent by each domain (CPU usage, waiting time, etc.). It also measures I/O requests by counting the number of memory page exchanges between a guest domain and Domain-0. The number of memory pages exchanged may not accurately represent the actual amount of data transferred, but reflects how many I/O are requested by a domain.

However, Xenoprof requires kernel support from Domain-0 which is absent in XenServer's Domain-0. And according to the Xenoprof documents, Xenoprof only profiles para-virtualized VMs that use a Xen-aware Linux kernel. Thus, we can not use it for our purposes. Xenmon is only partial working on XenServer. The I/O activity record always reports zero for our VMs in HVM mode.

	Real (sec)	CPU (sec)	Real (Mbit/s)
Non-Virtualized	313.25	2.53	94.15
Virtualized	329.32	2.63	89.55

Table 2. Performance comparison of nttcp benchmark.

7 Conclusion

In this paper, we presented a performance evaluation framework for virtual appliances. We created an environment based on the XenServer virtualization platform and evaluated the performance of AsteriskNOW, a SIP proxy server appliance. We also considered a range of configurations when studying performance. From our experimental results, we find that the Xen VMM manages hardware resource utilization (i.e., disk and network I/O) effectively across VMs. but performance varies wildly when running multiple virtual appliances at the same time. Sharing CPUs among VMs can also significantly impact performance. The best way to address this is to provide additional physical cores, so that each VM can run on separate PCPUs.

As future work, we plan to move to a multiprocessor multicore system, as well as consider environments that run multiple virtual appliances concurrently. We also want to utilize more detailed profiling in our work to more precisely identify bottlenecks in the system.

References

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, 2003.

[2] VMware Inc., "VMware ESX Server," http://www.vmware.com/products/vi/esx/.

[3] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, and Rich Uhlig, "Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization," Tech. Rep., Intel Inc., August 2006.

[4] Advanced Micro Devices, "AMD's virtualization solutions," http://enterprise.amd.com/us-en/AMD-Business/Business-Solutions/Consolidation/Virtualization.aspx.

[5] VMware Inc., "Understanding Full Virtualization, Paravirtualization," http://blogs.vmware.com/vmtn/2007/11/white-paper-und.html.

[6] Digium Inc., "AsteriskNOW," http://www.asterisknow.org/.

[7] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," 2002.

[8] R. Gayraud and O. Jacques, "SIPp," http://sipp.sourceforge.net.

[9] Citrix Inc., "Citrix XenServer Standard Edition," http://www.citrixxenserver.com/products/Pages/XenServer.aspx.

[10] Citrix Inc., "XenServer SDK Documentation," http://docs.xensource.com/XenServer/4.0.1/api/.

[11] "rPath Linux," http://www.rpath.com/.

[12] "Super Pi," http://home.istar.ca/ lyster/pi.html.

[13] Mike Muuss, "nttcp, a TCP throughput testing tool," http://sd.wareonearth.com/ phil/net/ttcp/.

[14] William D. Norcott and Don Capps, "IOzone Filesystem Benchmark," http://www.iozone.org/.

[15] Erich M. Nahum, John Tracey, and Charles P. Wright, "Evaluating SIP server performance," in *SIGMETRICS '07: Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, New York, NY, USA, 2007, pp. 349–350, ACM.

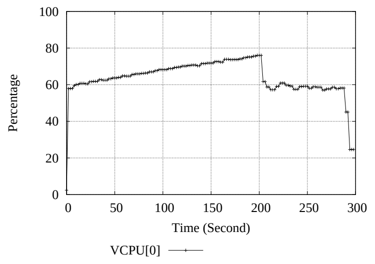
[16] Vikram Makhija, Bruce Herndon, Paula Smith, Lisa Roderick, Eric Zamost, and Jennifer Anderson, "VMmark: A Scalable Benchmark for Virtualized Systems," Tech. Rep., VMware Inc., September 2006.

[17] SPEC Virtualization Committee, "SPEC Virtualization Benchmark," http://www.spec.org/specvirtualization/.

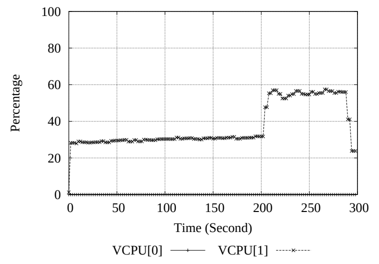
[18] SPEC SIP Committee, "SPEC SIP Benchmark," http://www.spec.org/specsip/.

[19] Aravind Menon, Jose Renato Santos, Yoshio Turner, G. (John) Janakiraman, and Willy Zwaenepoel, "Diagnosing performance overheads in the xen virtual machine environment," in *VEE '05: Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments*, New York, NY, USA, 2005, pp. 13–23, ACM.

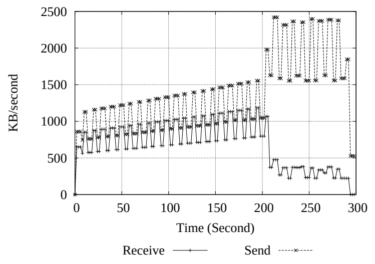
[20] Gupta Diwaker, Gardner Rob, and Cherkasova Ludmila, "XenMon: QoS Monitoring and Performance Profiling Tool," Tech. Rep., HP Labs, 2005.



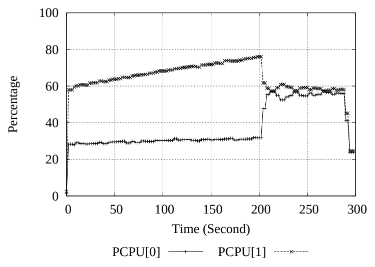
(a) CPU Util of Asterisk VM



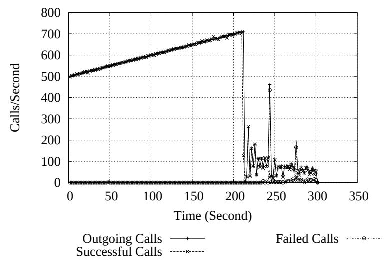
(b) CPU Util of Domain-0



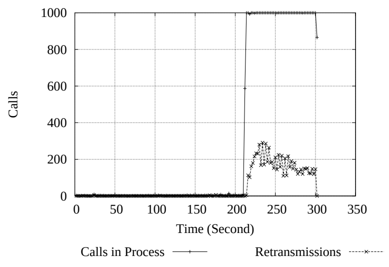
(c) Network Throughput



(d) Physical CPU Util

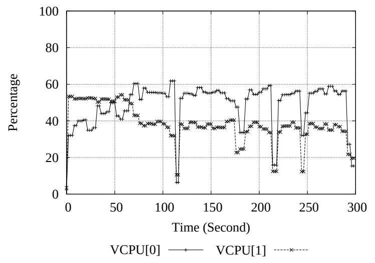


(e) Calls Statistics

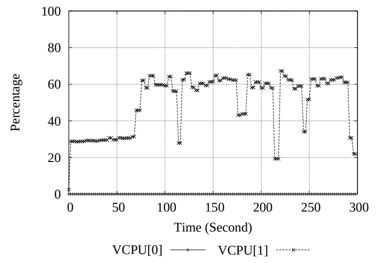


(f) Retransmissions

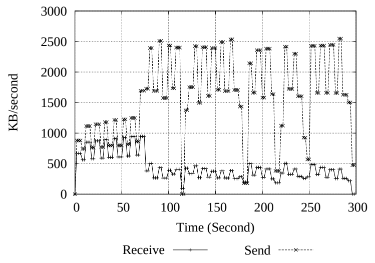
Figure 6. Statistics of [*1 Asterisk VM with 1 VCPU***]**



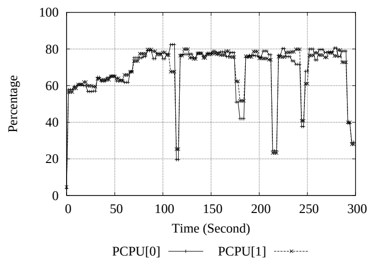
(a) CPU Util of Asterisk VM



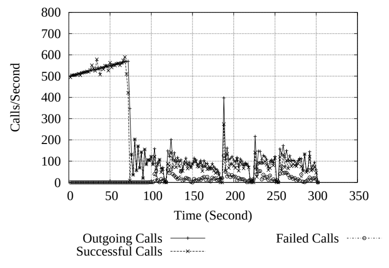
(b) CPU Util of Domain-0



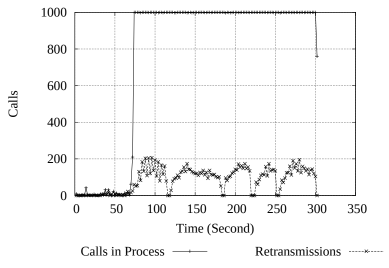
(c) Network Throughput



(d) Physical CPU Util

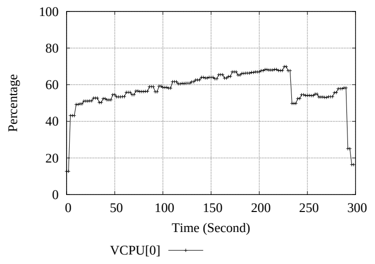


(e) Calls Statistics

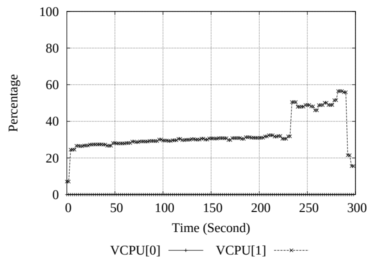


(f) Retransmissions

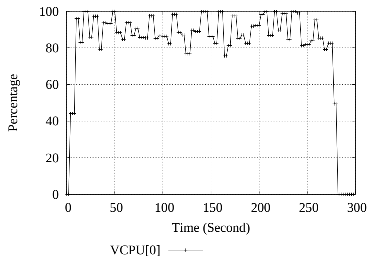
Figure 7. Statistics of [*1 Asterisk VM with 2 VCPUs***]**



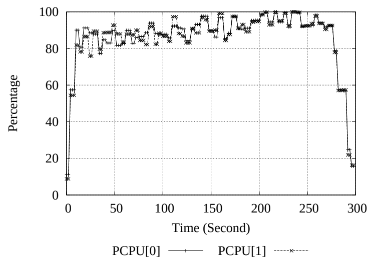
(a) CPU Util of Asterisk VM



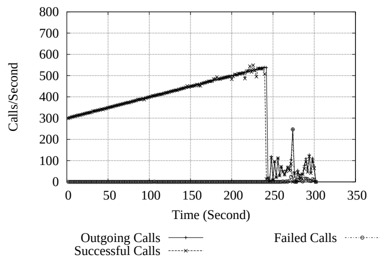
(b) CPU Util of Domain-0



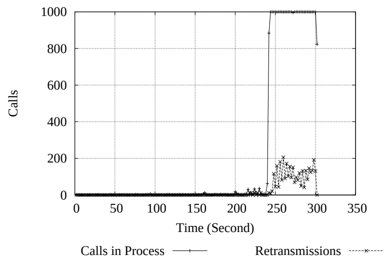
(c) CPU Util of Debian VM



(d) Physical CPU Util

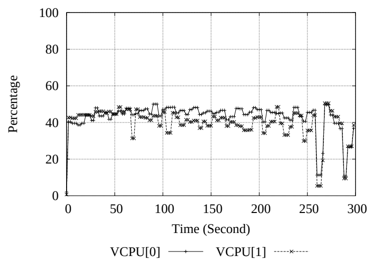


(e) Calls Statistics

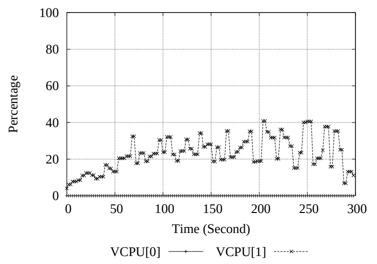


(f) Retransmissions

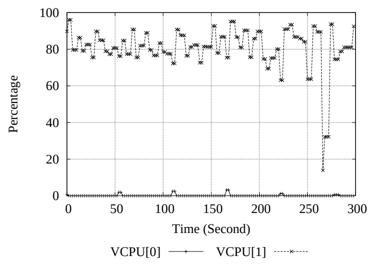
Figure 8. Statistics of [*2 VMs on CPU competition, 1 VCPU per VM***]**



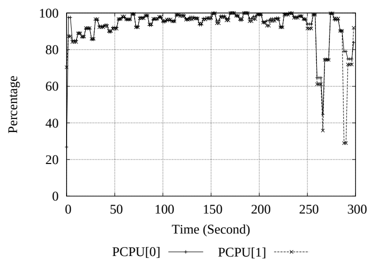
(a) CPU Util of Asterisk VM



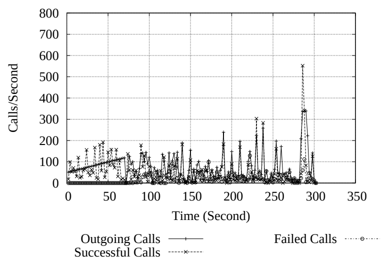
(b) CPU Util of Domain-0



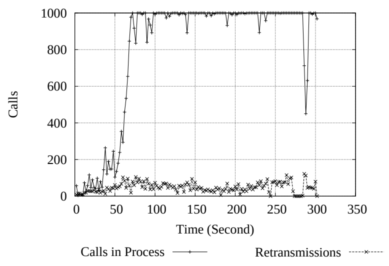
(c) CPU Util of Debian VM



(d) Physical CPU Util

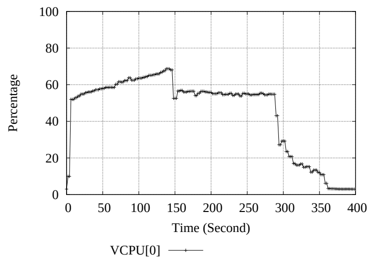


(e) Calls Statistics

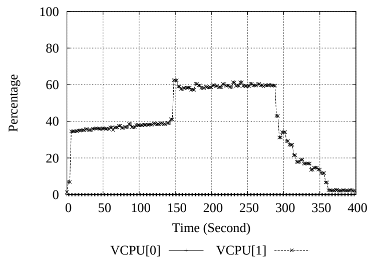


(f) Retransmissions

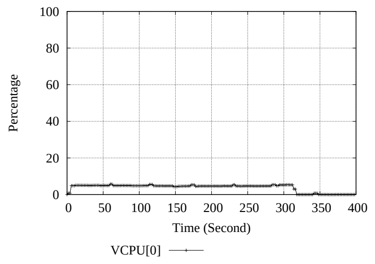
Figure 9. Statistics of [*2 VMs on CPU competition, 2 VCPUs per VM***]**



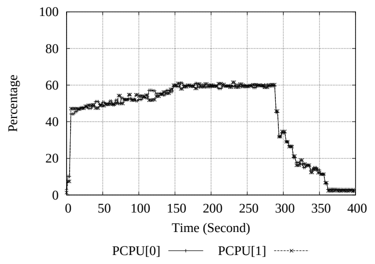
(a) CPU Util of Asterisk VM



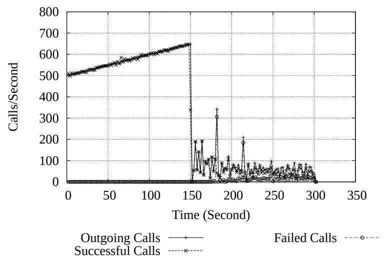
(b) CPU Util of Domain-0



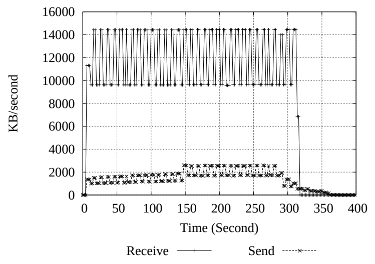
(c) CPU Util of Debian VM



(d) Physical CPU Util

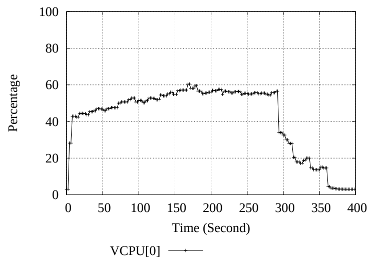


(e) Calls Statistics

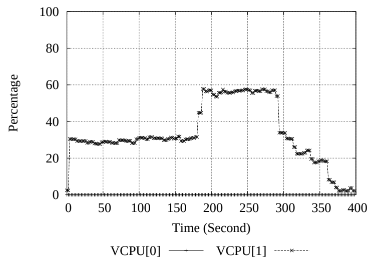


(f) Network Throughput

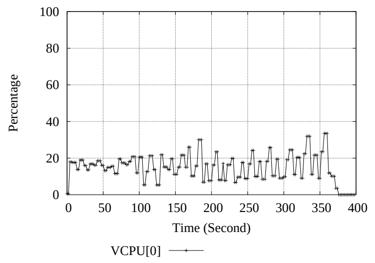
Figure 10. Statistics of [*2 VMs on Network I/O competition, 1 VCPU per VM***]**



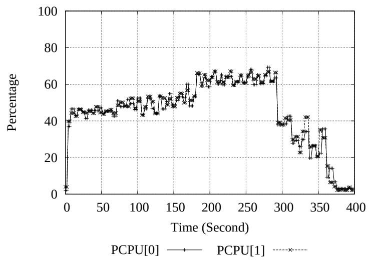
(a) CPU Util of Asterisk VM



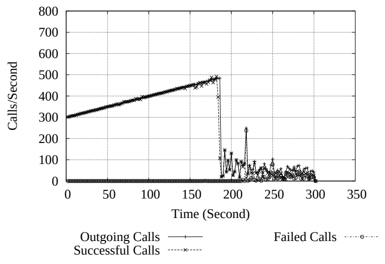
(b) CPU Util of Domain-0



(c) CPU Util of Debian VM

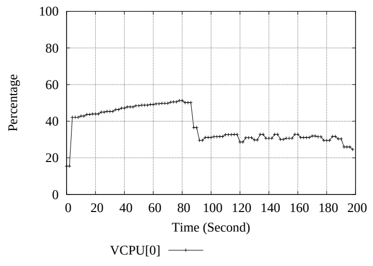


(d) Physical CPU Util

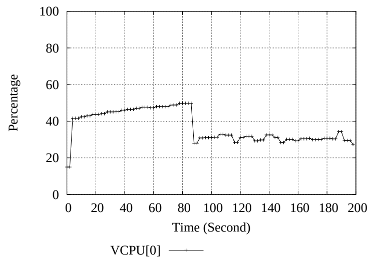


(e) Calls Statistics

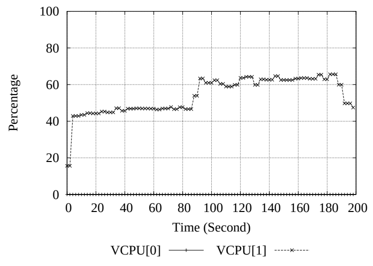
Figure 11. Statistics of [*2 VMs on Disk I/O competition, 1 VCPU per VM***]**



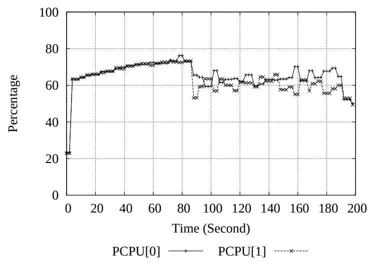
(a) CPU Util of Asterisk VM 1



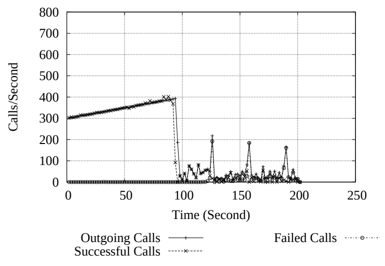
(b) CPU Util of Asterisk VM 2



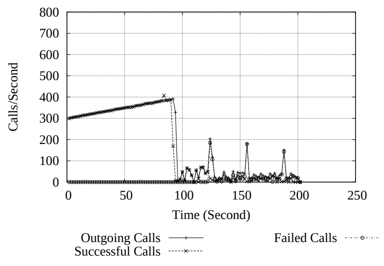
(c) CPU Util of Domain-0



(d) Physical CPU Util



(e) Calls Statistics of VM 1



(f) Calls Statistics of VM 2

Figure 12. Statistics of [*2 Identical Asterisk VMs, 1 VCPU per VM***]**