

Performance Implications of Hosting Enterprise Telephony Applications on Virtualized Multi-Core Platforms

Devdutt Patnaik^{*}
College of Computing
801 Atlantic Drive
Georgia Institute of Technology
Atlanta, GA 30332
devdutt.patnaik@gatech.edu

A. S. Krishnakumar P. Krishnan
Navjot Singh Shalini Yajnik
Avaya Labs
233 Mt. Airy Rd.
Basking Ridge, NJ 07920
{ask,pk,singh,shalini}@avaya.com

ABSTRACT

Virtualization technology has gained significant adoption in various domains as a means to lower costs and enable greener solutions. Recently, there has been a significant amount of interest in employing virtualization technology in the telecommunications domain in order to save costs through server consolidation and to provide energy-efficient solutions. The availability of high-end multi-core servers provides powerful platforms for deployment. However, the telecommunications domain poses unique challenges for virtualization technology to be successfully deployed even in these compute-rich multi-core environments. This work discusses these challenges. It provides a detailed analysis of the performance implications of hosting enterprise IP telephony infrastructure in virtualized environments. Unlike signaling applications that are comparatively more tolerant of underlying platform performance, media applications are far more demanding. Our work, therefore, focuses on the performance of media applications (media server, voice-mail, etc.) in virtualized environments. We develop a model for workloads used in enterprise IP telephony. We then evaluate the impact of various hypervisor scheduler and I/O parameters in order to determine good parameter settings for such workloads. Our experiments use the Xen virtualization platform. The results presented in this work will be useful for telecommunication solution providers to understand the capabilities and limitations of virtual environments when deploying their applications.

Keywords: Virtualization; Xen; Enterprise telephony workloads; Server consolidation; Virtualization overheads

1. INTRODUCTION

Recent developments and optimizations in virtualization technology [1, 2, 3, 4, 5] have led to its widespread adoption.

^{*}This work was done while the author was a summer intern at Avaya Labs.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IPTCOMM'09, July 7-8, 2009, Atlanta, Georgia, USA.
Copyright 2009 ACM.

Virtualization has already become ubiquitous in data centers for the consolidation of multiple servers on a single platform. For the general class of applications, virtualization has effectively helped reduce costs, space and power. Different market segments are now seriously evaluating the use of virtualization to host their infrastructure and solutions. As virtualization becomes ubiquitous, it is important to evaluate how the technology can be utilized in multiple segments and domains. Recently, telecommunication solution providers have been looking closely at virtualization technology. Enterprise IP telephony applications (especially media-based applications) pose unique challenges to virtualization technology due to their near real-time performance requirements. Virtualization technology has improved in terms of performance in the last few years but still faces significant performance challenges when it comes to real-time applications. Therefore, it is important to measure and analyze the performance implications of hosting such applications in virtualized environments. Our work examines the demands of enterprise IP telephony workloads and measures performance overheads of virtualizing such applications.

Even though significant amount of effort has been dedicated to optimizing the performance of virtual environments [6, 7, 8], I/O virtualization still remains an area that poses challenges to the adoption of the technology in certain domains, like enterprise telephony. The key factors affecting the performance of enterprise telephony workloads are scheduling and network I/O performance. For specific domains and applications like web hosting, email, java, and bulk data transfers, researchers have characterized the corresponding workloads and their performance evaluation has helped identify challenges and possible solutions for these domains. These studies are summarized below. However, *the practical impact of virtualization on the real-time requirements of enterprise telephony have not been well-studied*. We study this aspect for the first time here. Understanding how real-time applications, that otherwise work adequately in a non-virtualized environment, behave in a virtualized deployment is critical to the adoption of virtualization for enterprise IP telephony. Our work presents insights into the behavior of such applications and provides results that will be useful to telecommunication solution providers.

Apparao et al. [6] studied the performance of a server consolidation benchmark (specifically, the java, web and mail

benchmarks) on a multi-core platform running the Xen virtual environment [9]. They looked at the architectural characteristics such as CPI (cycles per instruction) and L2 cache misses per instruction and analyzed the benefits of larger caches for these benchmarks. Menon et al. [7] built a framework for monitoring called *xenoprof*, and using this framework identified that the network performance of a para-virtualized Linux VM running under Xen was significantly lower when compared to the native Linux execution performance. Zhang et al. [10] observed that system performance bottlenecks in hardware-assisted virtualized environment are mainly introduced by memory and I/O virtualization overheads. Several past papers [8, 11, 12, 13] have addressed the problems of network I/O virtualization. The approach taken by Menon et al. [8] optimizes the I/O channel between the driver domain and guest domains and utilizes high-level network offload features present in modern network cards. Their optimizations improved the receive performance but it still did not match the native Linux performance. Oi et al. [12] analyzed and improved network throughput of a Xen virtualized system with schemes like Large Receive Offload in the virtual interfaces for packets with 4KB MTU. The work by Santos et al. [14] on improving network I/O performance uses Netperf [15] as the workload generator and studies the impact of their optimizations on large packets. All the above research efforts have been targeted at optimizing network I/O throughput for sending large sized packets over the network. Media streams in IP telephony are usually small UDP packets (about 172 bytes of UDP payload) which arrive at constant intervals. Optimizations for such traffic have not really been the focus of any of the research efforts listed above. In the area of hardware transport technology, NetXen [16], Crossbow [17], and Intel VMDq [18] are trying to address the network I/O virtualization performance issues by having virtualization-aware ethernet NICs with multiple Rx/Tx device queues to offload network I/O data processing from the VMM software to the hardware. There has also been recent work on improving other aspects of virtualized I/O performance [19, 20]. Recently the Xen development community has introduced stub-domains [21] which aim at overcoming the problems in hardware-assisted VMs (HVMs) that arise due to Dom0 being a bottleneck for I/O intensive environments. However, the above work is relatively new and the technology has not been fully adopted into mainstream hardware products, and hence its applicability to the performance of current products is limited.

Besides the I/O path, another key parameter that impacts the performance of media in telephony applications is the hypervisor scheduler. Research efforts such as [22, 23, 24] study the impact of VM scheduling on I/O virtualization performance and highlight the need for improved scheduling techniques that help I/O bound VMs while maintaining fairness. This research on new scheduling techniques is useful for future products, however, it does not address the current needs of vendors who have to deploy real-time applications on existing virtual environments and understand how their applications will behave in these environments.

As discussed above, there is a gap in understanding how the enterprise IP telephony vendors can deploy their products on existing virtual infrastructures and what impact the infras-

tructure has on the product performance. This work aims to address that need. We model the characteristics of enterprise telephony workloads, and using this model evaluate the behavior of enterprise telephony applications (specifically the media server application) deployed in virtual environments. We believe that the enterprise telephony signaling server workloads are characterized fairly closely by workloads similar to medium intensity I/O applications like the web server. Such workloads have been extensively studied [6, 8] and will not be the focus of the current work. Virtualization of *servers processing media traffic* is an area that is ill-explored and the strict real-time demands of such workloads make their deployment on a virtualized platform a major challenge. We characterize the behavior of media-based telephony applications and using this model, we explore various scheduler and I/O configurations suited for such workloads and provide detailed results of our investigation. We also present a number of optimizations to help improve performance of such media-processing applications. Our experiments were done using the Xen virtualization platform [9]. The results of our experimentation will be useful to telecommunication providers who are considering virtualization technology to host their applications.

The rest of the paper is organized as follows: Section 2 gives some background on virtualization and in particular discusses the Xen virtual machine architecture. Section 3 introduces the basics of enterprise IP telephony applications and their requirements. Section 4 defines the model for the workloads used in our evaluation. Section 5 defines the parameters and metrics relevant to enterprise telephony that were used in the tests. Section 6 gives a brief description of the hardware and software platform and presents the results of the evaluation. Section 7 concludes the paper.

2. VIRTUALIZATION

Many enterprises are finding that they have a lot of servers doing either a specific task or a combination of small tasks, hence under-utilizing the processing power of these servers. Virtualization is becoming popular as it allows to run multiple virtual servers on a single physical machine. The consolidation of multiple servers on a single machine helps reduce the cost, space and power consumption.

There are a number of vendors providing virtualization solutions such as KVM [1], VMware [2], Xen [3], Microsoft's Hyper-V [25] to mention a few. This study was done using Xen, a popular open source virtualization solution.

Figure 1 shows the Xen Virtual Machine Monitor and its components. The Xen hypervisor has the highest privilege level and interacts directly with the hardware at the lowest level. The Xen Domain called Dom0 and Guest Domains called DomU reside above the hypervisor. Dom0 is a special purpose domain that is created at boot time and is allowed to access the control interfaces of the hypervisor. Dom0 is responsible for creating/terminating the guest domains through the control interface. The DomU's OS runs at a lower privilege level than Dom0 OS.

The hypervisor virtualizes the physical resources such as CPUs, memory, and I/O devices for the guest domains. Most of the non-privileged instructions can be executed by

the guest domains natively without the intervention of the hypervisor. However, the privileged instructions will generate a trap into the hypervisor. The hypervisor validates the request and allows it to continue.

As the I/O devices are shared across all the guest domains, the hypervisor controls access to them for each and every request. Xen provides a split device driver model where each I/O device driver runs in Dom0 and is known as the backend driver. The guest domain has the frontend driver that communicates with the backend driver via shared memory. The packets coming from the guest domain are written to the frontend driver in that domain. The frontend driver copies the packets to the shared memory. The backend driver in Dom0 picks up the packet and passes it through the real network drivers in Dom0 to the physical network interface via the hypervisor. This split I/O model needs changes to be made to the guest OS. In Xen, this type of deployment is called para-virtualization.

In addition, Xen allows deployment of the guest OS without any modification. This type of deployment is called Hardware Virtual Machine (HVM). The I/O support for the unmodified guest OS is provided via an emulated device driver in Dom0. The HVM device driver communicates directly with the emulated device driver in Dom0. However, the HVM needs to be scheduled each time it needs to communicate with the Dom0. In addition, the emulated device driver needs to be scheduled when it wants to pass the packets to the Dom0 network driver. So there is an additional level of indirection leading to poor I/O performance in HVMs [26]. Additionally, the overheads due to the emulation itself are very high.

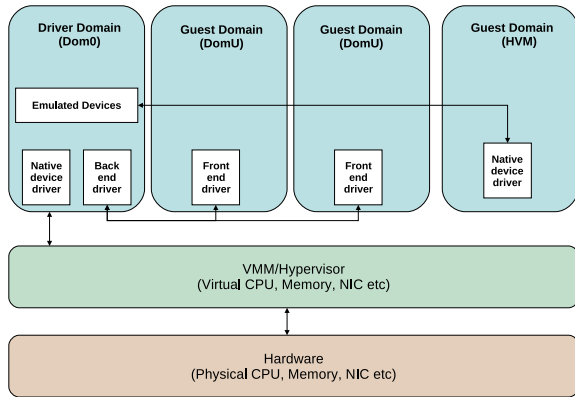


Figure 1: The Xen Virtual Machine Monitor (VMM)

3. ENTERPRISE IP TELEPHONY

An enterprise IP communication system deploys several components to provide the telephony services for the end-users. The main components are (i) call server, (ii) media servers or gateways, and (iii) a number of telecommunications end-points. The call server, also known as the signaling server, is the brain of the enterprise communication system. It performs various tasks such as end-point registration, deregis-

tration, address translation, and call control. A single call server can control multiple media servers/gateways and end-points. The media server provides the mapping and translation between different IP and telephony networks such as IP end-points or PSTN phones. One of the main functions of the media server is to convert the media stream supported by these networks e.g. G.711 speech into G.729 speech and vice versa. The media gateway is also responsible for support services such as echo cancellation and tone generation.

Figure 2 shows an enterprise IP telephony system that sets up and controls voice conversations between end-points such as IP phones. Once the call is setup the voice conversation is carried over Real-Time Transport Protocol (RTP) [27] between the endpoints.

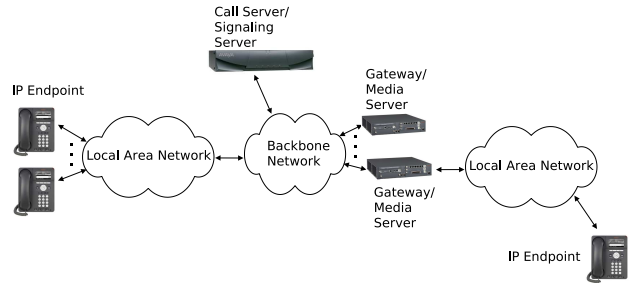


Figure 2: An Enterprise IP Telephony System

4. MODELING ENTERPRISE IP TELEPHONY WORKLOADS

In order to accurately model real-world workloads in telephony environments, we define the characteristics of the applications that comprise the enterprise telephony workload. As mentioned in Section 3, the key components of an enterprise telephony system are the signaling servers, the media servers and the clients. We defined a suite of applications that closely model the behavior of these three components. We mapped the operations of each server/client in terms of its three basic components - CPU, I/O and disk - and modeled a similar behavior of those components in our applications. We did not use the actual servers in our experiments to avoid being implementation-specific and to also allow tunability in terms of the various basic operations (e.g. compute time per stream, I/O behavior etc). Figure 3 gives an overall picture of how the application suite was deployed in our tests. The details of each of these components and their models are as follows:

- **Media Server:**

The job of the media server component is to handle incoming media streams from clients, process them and forward them to the terminating clients. The processing of each packet of the stream may involve functions like transcoding, jitter buffer manipulation, echo cancellation, tone detection, etc. We modeled this application with a pool of threads that serviced incoming client streams. Each thread picked up a packet from an incoming stream, performed some computation on it and then relayed the packet to another client. This media server model application was hosted in a single

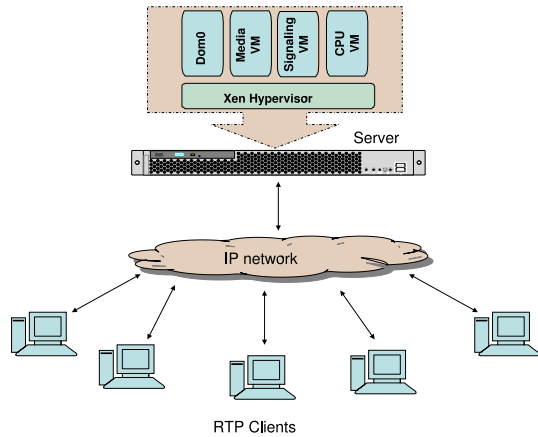


Figure 3: Modeling Enterprise Telephony Workloads

virtual machine (VM). Henceforth, in this work, we will refer to this VM as the *Media VM* which will be the focus of our performance evaluation.

- **Signaling Server:**

Another key component of an enterprise telephony platform is the signaling server whose main function is call control. In addition to basic functions like setup and teardown, this component also provides more complex features like transfer, hold, multi-party calling etc. We modeled this application by driving it with traffic that was equivalent to a call setup/teardown traffic (around 10 packets per call setup and teardown). The application also performed some basic computation and invoked some disk I/O operations. The signaling server application was hosted in a separate virtual machine which we refer to as the *Signaling VM*.

- **RTP Clients:**

Real-Time Transport Protocol (RTP) [27] is the most common protocol for carrying media between IP endpoints. Our IP endpoints were modeled by client applications generating RTP traffic over UDP transport. We chose a 20 milliseconds packetization interval for the voice streams. Hence, each stream resulted in a 172 byte UDP payload packet every 20 msec. The client application attached a timestamp to each packet (hidden in the RTP payload) before the packet was sent on the network. The receiving client application interpreted this timestamp and also generated a timestamp for each packet on arrival. The time stamps on the packets were used to calculate the round trip time (RTT) for packets. In order to avoid time synchronization issues, the sending and the receiving client for a given stream were deployed on the same machine. Sequence numbers in the RTP payload were used to detect lost or dropped packets. RTP clients were hosted on several machines distributed across the network, with each machine contributing upto 200 RTP streams.

4.1 Deployment Scenarios

Our study targets two typical scenarios for deployment of enterprise telephony platforms. The two scenarios are described briefly below. A more detailed exploration of the two scenarios is given in Section 6. In both deployment scenarios, the experimental focus is on understanding the performance of the media VM.

- **Mixed Workload Deployments**

A very common deployment scenario is where an existing virtualization infrastructure is already deployed within an enterprise. The enterprise telephony servers (e.g. the Media VM and the Signaling VM) are distributed as *virtual appliances* or *virtual disk images* that are given to the customer to deploy within the existing infrastructure. These server VMs will have to co-exist with other virtual machines already running on the host machine. We define this case as the *Mixed Workload Deployment* scenario. We model the extra workload on the host system through the addition of one or more CPU-intensive VMs. In this case, the telephony applications will require some guarantees from the environment, as they may share the execution environment with other workloads that may interfere with their resource (CPU, disk, I/O) usage. It is very important to study the implications of deploying telephony workloads in mixed workload environments, so that a vendor can clearly specify what kind of guarantees the application needs from the host machine and what parameters should be tuned to achieve those guarantees.

- **Turnkey Deployments**

The second deployment scenario is when the telephony applications are deployed on a dedicated server platform. In this case it is important to extract the best performance from the virtualized environment and therefore, to understand the hardware and software configurations that will deliver optimal performance for such applications. This will be the focus of the experiments evaluating this scenario. We call this scenario *Turnkey Deployment*.

5. PARAMETERS AND METRICS

In order to evaluate the performance of enterprise telephony applications, we identified some critical parameters and metrics based on the nature of these workloads.

5.1 VM Configurations and Parameters

Three key parameters have the potential to affect performance of our workloads: (a) the VMM scheduler, (b) CPU configurations, and (c) device and network configurations. We evaluate our workloads with respect to these parameters.

5.1.1 Credit Scheduler Parameters

Credit-based scheduler [28] is the default scheduler used in all recent versions of Xen. It is fairly configurable and allows for flexibility in terms of coarse grained VM priorities by allowing the user to specify two parameters, *weights* and *caps*, for each virtual machine.

1. **Weights:** Each VM is allocated credits by the scheduler based on its weight. The scheduler allows the user to specify weights for VMs based on how much share of the CPU needs to be given to a particular VM. The default value of weight for each VM is 256. In each scheduling interval, a virtual machine is allotted time slices in proportion to its weight. In each accounting interval (smaller than a scheduling interval), the weight of each VM is updated based on the time slice it used during that interval. By giving higher weights to Media and Signaling VMs we can ensure that the media and the signaling server applications get a good share of the CPU.
2. **Caps:** Consider the case when the credit scheduler has a VM which has exhausted its credits in the current scheduling interval but still has some work to do. If the CPU is idle (i.e. there are no other higher weight VMs waiting to be scheduled in that interval), by default, the credit scheduler will let the VM use the extra slices from the available CPU. This is called *work-conserving* mode for the scheduler. However, the scheduler also provides a parameter called *caps* to prevent a virtual machine from utilizing more than its share of CPU. Setting the value of cap to 1 for a VM informs the scheduler that the VM should not be given CPU over and above its weighted share. However, it is *non-work conserving* and may lead to wasted CPU resources.

5.1.2 CPU Configurations

Xen provides custom CPU resource management by allowing each virtual machine to have one or more virtual CPUs (vCPUs). On a single core machine, there is not much advantage to giving more than one virtual CPU to a virtual machine. However, in a multi-core CPU environment, giving multiple vCPUs to a virtual machine allows the VMM to parallelize the execution of that VM by assigning different vCPUs to multiple cores. A vCPU can also be *pinned* to a particular core.

We can pin a VM (and its vCPUs) to one or more physical CPUs using *VM pinning*. This may allow the cache to remain warm for a certain VM, but at the same time may result in less than optimal system utilization as there may be unused cycles that other VMs cannot utilize if they are pinned away from this CPU. Pinning can be used to set up the system for evaluating the impact of cache affinity on VMs that share data and code.

5.1.3 Network and Device Configurations

1. Directed I/O emulated through VMM bypass

As shown in Figure 1, the default network device configuration in Xen 3.2.1 has all device drivers residing in the privileged driver domain - Dom0. All network I/O operations require the involvement of the VMM and the Dom0. This leads to Dom0 being a bottleneck for systems with heavy I/O demands. The heavy reliance on Dom0 may lead to degraded performance for Media VM which is heavily I/O intensive. A technique called directed I/O presents a secure way to allow guest VMs access to the NIC via IOMMU [29]. However, support for this technique is not fully implemented in the current hardware/OS. VMM bypass [20], a device

configuration that allows time-critical I/O operations to be carried out directly in guest domains, is available in current systems and allows us to evaluate the performance benefits of directed I/O. In this configuration, network I/O operations do not require the involvement of the VMM and/or Dom0. The guest domain is granted direct access to the network interface card (NIC). The domain directly loads the device driver that controls the NIC.

2. Interrupt Coalescing

A very common source of bottleneck for high speed I/O is the number of interrupts the receiving OS has to process. During normal operation, each received packet generates an interrupt which is handled by the device driver. Context switches and interrupt handling lead to significant CPU resource being consumed. Interrupt coalescing is a feature of the latest network-interface cards that allow us to wait until several packets have been received before an interrupt is generated. This prevents the host from being flooded with too many interrupts and lowers the CPU utilization. A timer starts when the first packet arrives. The interrupt can be delayed for n microseconds or until m packets arrive, where n and m are configurable parameters of the NIC. Note that while interrupt coalescing lowers CPU overhead, it also increases the delay in delivering the packet to the application.

5.2 Key Performance Metrics

Due to the real-time nature of media streaming, most media applications use UDP as their transport protocol. However, UDP does not give any packet delivery guarantees and in adverse network/host conditions can create packet losses. Media (i.e. voice and video) quality is very susceptible to packet losses and degrades significantly with losses. Packet delays in the network and/or the host also create additional quality issues with media. Hence, packet loss and delay are the two key metrics that we aim to evaluate and minimize in our experiments.

6. EXPERIMENTAL RESULTS

In this section, we first describe the hardware and software platform built for the experiments. Then we discuss the experimental configuration and the results.

6.1 Hardware and Software Platform

The choice for the hardware platform was based on real-world hardware configurations that are typically chosen for deploying telephony infrastructure and VoIP applications such as media and signaling servers. A Dell server was chosen for our experiments. It had an Intel quad-core Xeon processor with each core running at 1.6 GHz, 4Gbytes of RAM and two 1 Gb NIC cards.

The open-source Xen 3.2.1 hypervisor was used as the Virtual Machine Monitor for our experiments. We used the default SMP credit scheduler in Xen, as this is the scheduler that will be supported in future releases, and is optimized for SMP performance. The *sedf* scheduler [30] provides finer-grained allocation of time slices and lets the real-time applications have a better control of how time is allotted to

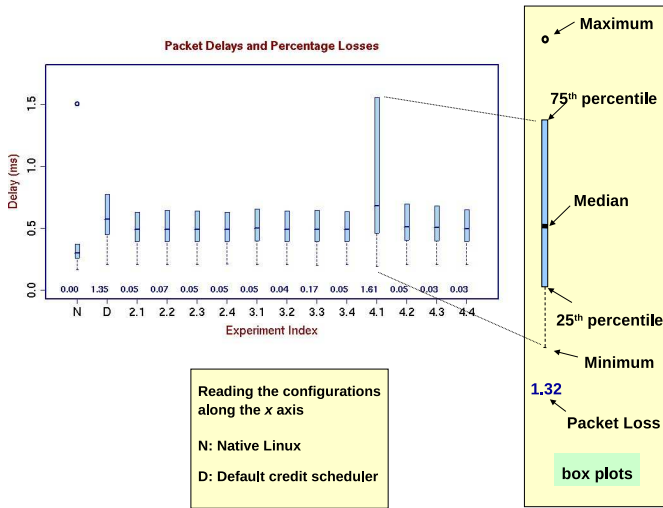


Figure 4: Reading the Graphs

them. However, future support for sedf is not guaranteed and hence we chose to use the credit scheduler and tune its parameters to our requirements. Hardware Assisted VMs (HVM) have been shown to have poor I/O performance [26] and since our applications were I/O intensive we chose to deploy them on para-virtualized VMs. The VMs ran para-virtualized Linux 2.6.18.

As discussed in Section 4.1, our experiments were performed for two deployment configurations while varying parameters described in Section 5.1. For all these experiments the credit scheduler in Xen 3.2.1 was used. The Media VM was given 2 vCPUs in order to exploit parallelism when running on more than 1 core. Other VMs were given a single vCPU each. Since the machine had 4 physical cores, by default Xen assigns 4 vCPUs to Dom0, so that it has the ability to use all the cores. The Media VM and the Signaling VM were each assigned 1GB of memory. Dom0 was given the residual memory (which was different for the two deployment scenarios) after the other VMs had been assigned their memory quotas. The graphs for the experiments show boxplots of the packet delays in milliseconds and also show the percentage of packets lost during the experiment. Figure 4 shows how these graphs should be interpreted. The X-axis gives the experiment set. Experiment *N* on the X-axis gives the results for native Linux. Experiment *D* gives the results when all parameters are set to their default values on the virtualized platform. The configurations for the rest of the experiments are either self-evident from the axis tag or are explained in the table below the X-axis. The Y-axis gives the delay values in milliseconds. The boxplots give the variance of delay (details are as shown in the figure). The number below the boxplot gives the percentage of packet loss experienced by the media streams in the experiment.

6.2 Experimental Methodology

Our experiments are targeted at first understanding the impact of each parameter like the scheduler, I/O etc. on the performance of media streams and then applying this knowledge to tune the parameters for each deployment scenario. The methodology we used for experimentation is shown in

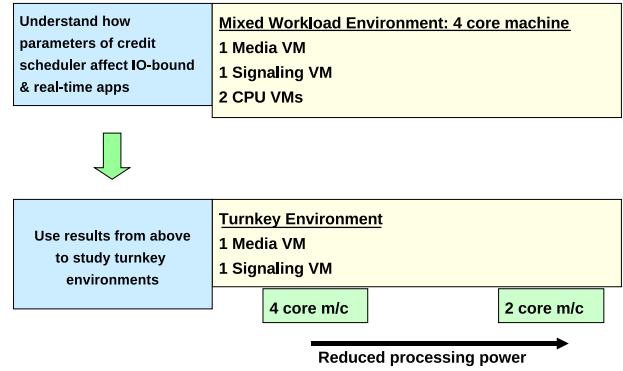


Figure 5: Experimental Methodology

Figure 5. The top part of the figure shows how the mixed workload environment is used to understand the impact of each individual parameter. The bottom half of the figure shows how the turnkey deployment scenario is evaluated. While choosing the hardware for deploying a turnkey solution, one of the key criteria used by vendors is that the hardware should be cost-effective, i.e. have sufficient resources to meet the application demands at the lowest cost possible. Our aim while studying the turnkey scenario was to find the minimal set of resources needed for our workload. In this scenario, we start with a fully provisioned system and then reduce the processing power to find the optimal processing power needed for meeting the demands of the workload. The knowledge obtained from the previous experiments was applied here to get the best possible performance from the limited resources available.

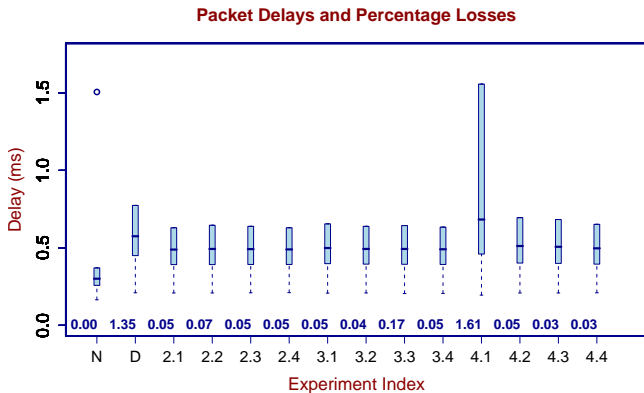
6.3 Mixed Workload Environments

We first evaluate the mixed workload deployment environment. This environment emulates the scenario where enterprise telephony servers are deployed in a pre-existing virtualized platform and they may coexist with other virtual machines on the platform. In addition to the Signaling VM and the Media VM, our mixed workload environment consists of *two CPU-bound VMs* that model other workloads coexisting with our enterprise telephony VMs. The CPU-bound VMs run highly compute-intensive tasks in between small idle periods. The CPU-bound VMs were given 512MB of memory each and Dom0 was assigned the residual memory, which in this case was 1 GB. With this configuration of 4 virtual machines (the Signaling VM, the Media VM and 2 CPU-bound VMs), we tune parameters specified in Section 5.1 individually and study the impact of each parameter on the packet loss and the delay suffered by the packets processed by the Media VM. Note that similar study was done for the packets handled by the Signaling VM. However, the focus of this study is the media path and hence the results for the other VM are not presented here. All these configurations were tested with 800 media streams being sent to the Media VM and an equivalent signaling load for call setup on the Signaling VM.

The tests were first run on a native Linux configuration (N) where each server was run as a separate application process on the same OS. Native Linux was able to handle the load of 800 streams with minimal delays (less than 0.5 msec) and 0% packet loss. It is important to note that in the results presented in the subsequent sections, since the endpoints were on the same high-speed LAN, all delay and loss numbers shown in the experiments are effectively that of the server deploying the applications. Also note that although the loss numbers of 1-2% may seem low, it is unacceptable for a server itself to induce losses of this magnitude.

6.3.1 Weight Configurations

The credit-based scheduler in Xen 3.2.1 uses a notion of weights associated with each VM to decide what proportion of the CPU will be given to the VM. In our experiments the weights of the Media VMs and Dom0 were increased to improve performance. In general it was observed that keeping Dom0 and the Media VMs at higher weights than the CPU bound VMs gave reasonably good performance with minimal losses. Figure 6 gives the results for various experimental configurations. Weights of 1024 (1K), 4096 (4K), 8192 (8K) and 65535 were evaluated for Dom0. Experiment set 2. i gives the Media VM the same weights as Dom0, Experiment set 3. i gives half the weight of Dom0, and Experiment set 4. i gives quarter of the weight of Dom0. The CPU bound VMs were given the default weight of 256. The results show that higher weights being given to the Media VM and Dom0 is important for streaming data and helps reduce the delays and losses suffered by the streaming data. Experiment 4.1 shows the significant impact that weights have on performance. In this experiment, Dom0 has a weight of 1024 and the Media VM is at quarter the weight of Dom0 which computes to 256. Since the weight of the Media VM is equal to the default (and hence equal to the weight of CPU-bound VM), we observe that the Media VM does not get enough resources and incurs higher delay and packet loss.

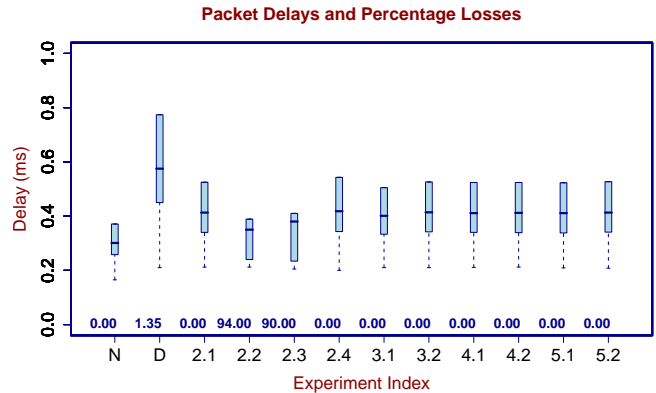


Exp.#	Description
2. i	wt(Media VM) = wt(Dom0)
3. i	wt(Media VM) = wt(Dom0)/2
4. i	wt(Media VM) = wt(Dom0)/4

Figure 6: Effect of VM weights. Configuration $C.i$ has Dom0 weights as 1K, 4K, 8K, and 65535 for $i = 1, 2, 3,$ and 4 respectively.

6.3.2 Cap Configurations

Next, we evaluate the impact of the credit scheduler's cap parameter. In this set of experiments, we cap various virtual machines and see the impact on the delay and losses. Figure 7 gives the results of the experiments. Experiment set 2. i caps each of the four VMs. Note that the weights of the VMs were left at default values (256) and only the cap parameter was tweaked for this set of experiments. Experiment 2.1 and 2.4 cap both CPU-bound VMs and a single CPU-bound VM respectively and yield similar results. For our workload, capping one CPU VM was enough to bring the packet loss down to 0. As expected, capping Dom0 (Experiment 2.2) and the Media VM (Experiment 2.3) gives very bad results with packet losses as high as 90%. A cap of any of these two VM results in the VMs not being scheduled often enough and hence the receive and transmit buffers fill up and packets get dropped. Experiment set 3. i and 4. i study the combined effects of caps and weights. The CPU-bound VMs are capped and weights of Dom0 and the Media VM are increased. Results indicate that for our workloads, capping CPU-bound VMs is enough to achieve 0% packet loss. We do not need to additionally adjust the weights of the VMs.



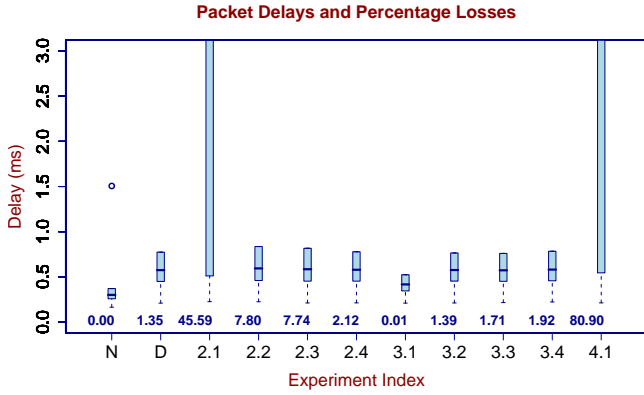
Exp.#	Description
2. i	For $i = 1 \dots 4$ resp., caps on both CPU VMs, only Dom0, only Media VM, and one CPU VM
$C.i,$ $C=3-5$	For $i = 1, 2$ resp., caps on both CPU VMs, and cap on one CPU VM. For $C=3-5$, $C = 3$, wt(Media VM) = wt(Dom0) = 8K $C = 4$, wt(Media VM) = wt(Dom0)/2 = 4K $C = 5$, wt(Media VM) = wt(Dom0)/2 = 32678

Figure 7: VM Cap Configurations

6.3.3 Pinning Configurations

In these configurations, VMs were pinned to specific cores to evaluate if it would help the Media VM perform better. Reserving resources for the Media VM by pinning cores may give better quality of service to the streaming data at the cost of less than optimal overall CPU utilization. The weights of all VMs for these experiments were kept at the default value of 256.

Figure 8 shows the results for various experimental configurations. As observed in Experiment set 2. i , pinning Dom0 to fewer cores impacts both the packet loss and delay numbers



Exp.#	Description
2. <i>i</i>	Dom0 pinned to <i>i</i> cores
3.1	CPU VM pinned to 1 core
3.2	CPU VM pinned to 2 cores
3.3	CPU VMs and Sig. VMs pinned to 2 cores
3.4	CPU VM pinned to cores 2, 3
4.1	Media and Sig. VM pinned to cores 0, 1 Dom0 and CPU VM pinned to 2 cores

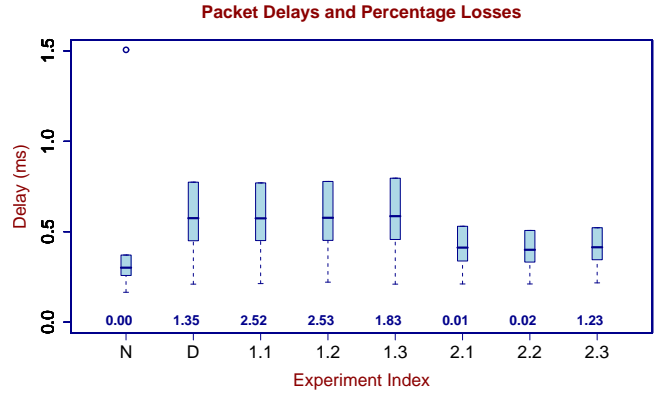
Figure 8: VM Pinning Configurations

negatively. When Dom0 has only one core available (Experiment 2.1), the packet losses are as high as 45%. As stated in Section 2, Dom0 is the driver domain that performs I/O on behalf of all VMs and hence needs access to more CPU resources for handling each incoming and outgoing packet in a timely manner. The best result for our workload is seen when Dom0 is allowed to float on all 4 cores in order to use as much CPU as possible on any available core. Experiment 3.1 shows that limiting CPU bound VMs to a single core frees up cores for the Media VM and Dom0 and the best results for this experiment set 3 are obtained under this configuration. Experiment 4.1 shows that when Dom0 shares cores with CPU-bound VMs, it gets starved for resources and that leads to bad overall performance. In conclusion, pinning the Media VMs to cores improves performance as long as enough cores are available. Additionally, the performance of the Media VM depends heavily on the availability of resources for Dom0.

6.3.4 CPU Affinity Configurations

Packets received and/or transmitted by the Media VM traverse through the drivers in Dom0. Packet buffers are exchanged between the Dom0 and the Media VM during this process. We wanted to evaluate the theory that a shared L2 cache between these two domains may help improve the performance. The processor on our hardware had two L2 caches, with one L2 cache shared by cores 0 and 1 and the other L2 cache by cores 2 and 3. In order to study the impact of possible sharing of L2 cache by Dom0 and the Media VM we performed a set of pinning experiments. The results of the experiments are given in Figure 9.

Experiment *D* is the default case with default weights of 256 and no pinning of VMs. In Experiment 1.1, Dom0 and Media VM were pinned to cores 0 and 1 to exploit any possible



Exp.#	Description of sharing cores
1.1	Dom0 & Media VM on 0, 1, others on 2, 3
1.2	Media VM on 0, 1, Dom0 floats, others on 2, 3
1.3	Media VM on 2, 3, Dom0 floats, others on 0, 1
2. <i>i</i>	Repeat of 1. <i>i</i> with no CPU VMs

Figure 9: Evaluating CPU Affinity between VMs

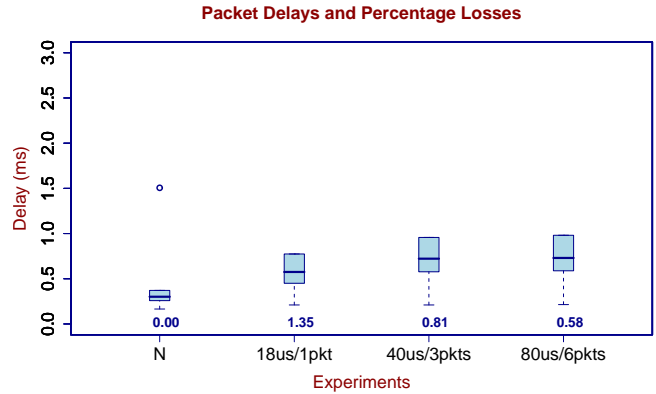


Figure 10: Interrupt Coalescing Experiments

benefits that may arise from cache sharing. In Experiment 1.3, we pinned the two VMs on cores that did not share any L2 cache. As seen from the graphs, cache sharing did not yield any noticeable improvement in performance. Experiment set 2.*i* repeats the same experiments but without any CPU-bound VMs. No noticeable change in performance was observed in this set. In general, performance of I/O bound VMs did not improve significantly even in cases where Dom0 and the Media VMs shared the same cache, i.e. the VMs were pinned to the same 2 cores. This result may also have been impacted by the fact that Dom0 was restricted to less than the maximum available cores and hence did not have enough resources available. Therefore, we can also conclude that allocating more resources to Dom0 was more important than having it share the cache with the Media VM.

6.3.5 Interrupt Coalescing Configurations

Interrupt coalescing is a feature of new NICs that allows us to delay the interrupt delivery to the processor. For high frequency I/O scenarios (like the media server), this could

potentially help reduce the overall processing per packet. By default, every time a packet is received, an interrupt is delivered to the CPU for packet processing. If the packets are batched before an interrupt is delivered, it will reduce the CPU overhead associated with context switching and interrupt delivery. The default configuration for our NIC was interrupt delivery for every packet. We tried configurations of $40\mu\text{s}$ or 3 packets and $80\mu\text{s}$ or 6 packets. This meant that the CPU would be interrupted once every defined time-slice or when x packets were received, whichever is earlier. We observed that while the overall delay increased due to holding back the packets, there was only a marginal improvement in packet loss numbers. Figure 10 shows the results of these experiments.

6.3.6 Summary of Mixed Workload Experiments

In summary, it was observed that although the performance of media streams with native Linux was good, when deployed within the Xen virtual environment it did not meet the media quality requirements. The performance of Media VM with respect to handling streaming packets was improved by a combination of one or more of the following techniques: (a) increasing the weight of Dom0 and Media VM with respect to the CPU-bound VMs, (b) putting a cap on the CPU-bound VMs, and (c) pinning CPU-bound VMs away from Dom0 and Media VMs and limiting them to one or two cores.

There were no visible improvements with cache affinity configurations or interrupt coalescing. The key observation from the experiments was that there is a heavy dependence of the Media VM on Dom0 as Dom0 needs to do the actual I/O on behalf of other VMs. Therefore, Dom0 can be a bottleneck and needs ample resources for I/O intensive workloads. Even in that case Dom0 can become a potential bottleneck as it hosts all the physical drivers.

6.4 Turnkey Deployment

In the turnkey environment, a dedicated server is used to host the virtualized telephony applications. In this case, the load capacity of the virtualized infrastructure is measured in terms of the number of concurrent streams that the Media VM needs to support. As discussed in Section 4, a heavily I/O intensive VM is used to emulate the media server and a light I/O bound VM to emulate the signaling server. Besides these two VMs, there were no other competing virtual machines on the platform. The Media VM was allocated 2 vCPUs. The loads were varied from 120 concurrent streams to 720 concurrent streams. These loads are typical of a medium capacity enterprise deployment. The Signaling VM was loaded with traffic equivalent to 10 calls per second to 60 calls per second. The load on the Signaling VM was calculated based on a call hold time of 60 seconds and 10% of calls being processed by the media server. For other calls media was shuffled directly between the two endpoints. In these experiments, we used our understanding of the results from the mixed workload experiments and tuned the parameters that we found relevant from that experiment set to get the best performance for the turnkey deployment scenario.

6.4.1 4-Core Deployment

In this configuration, all 4 cores were used to deploy the two VMs (the Media VM and the Signaling VM) with the loads

varying from 120 streams (10 calls per sec for the Signaling VM) to 720 streams (60 calls per sec for the Signaling VM). Figure 11 shows the results of the experiments. It was observed that the performance was good and that it was an over-provisioned system as ample resources were available for the Media VM. Thus for our workloads, which model a medium capacity enterprise deployment, the results were favorable with a 4-Core system.

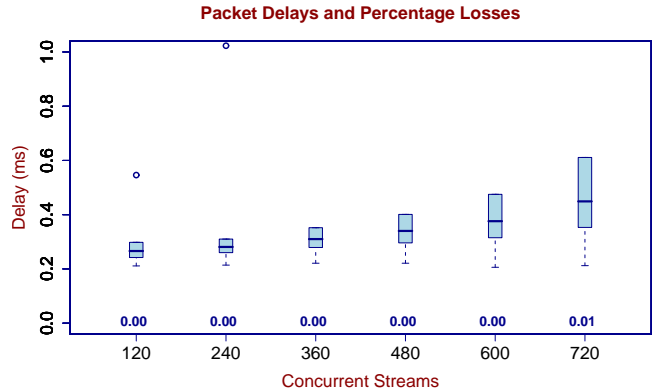
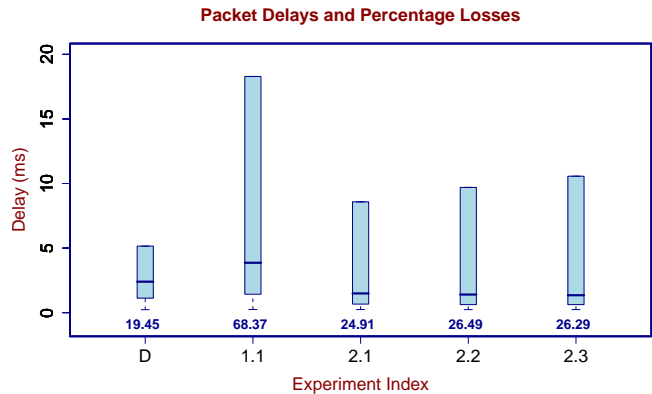


Figure 11: 4-Core Turnkey Configuration

6.4.2 2-Core Deployment

As was seen in the 4-Core deployment, there were ample resources to host the turnkey solution. In order to understand the hardware requirements of the two applications, we then limited the configuration to two cores. We ran experiments with 720 concurrent streams (with 60 calls per second load on the Signaling VM) and adjusted weights and pinning configurations to get the most optimal result from the limited hardware. Figure 12 shows the results of the experiments.



Exp.#	Description
1.1	Media VM on 1 core
2.i	Media VM on 2 cores: for $i = 1, 2, 3$ resp., $wt(\text{Dom0}) = wt(\text{Media VM}) = 1\text{K}, 4\text{K}, 8\text{K}$

Figure 12: 2-Core Turnkey Configuration

Experiment *D* gives the default configuration running on two cores. As can be seen from the graph, there are heavy

packet losses (about 19%) and high delays. Pinning the Media VM and Signaling VMs away from each other (Experiment 1.1) further degrades the result, since the Media VM is only allowed to use resources from 1 core. Giving higher weights to Media VM as compared to the Signaling VM (Experiment set 2.i) does not have any positive effect on the performance. This was clearly an under-provisioned system. Two cores were not enough to meet the processing demands imposed by 720 concurrent media streams.

6.4.3 Emulating Directed I/O through VMM Bypass

Since the 2-Core hardware configuration was found to be highly limited in terms of processing resources and performed very poorly in terms of packet loss and delays, we tried to eliminate some of the extra processing needed at the Dom0 for each packet by giving the Media VM direct access to the second NIC card through a technique called *VMM bypass* [20]. The next configuration that was evaluated was the same configuration as the 2-Core deployment but with VMM bypass turned on for the second NIC. In this configuration the second NIC card was hidden from Dom0 and the Media VM was given direct access to the card. This led to a significant improvement in performance. Figure 13 shows the results of the VMM bypass experiments.

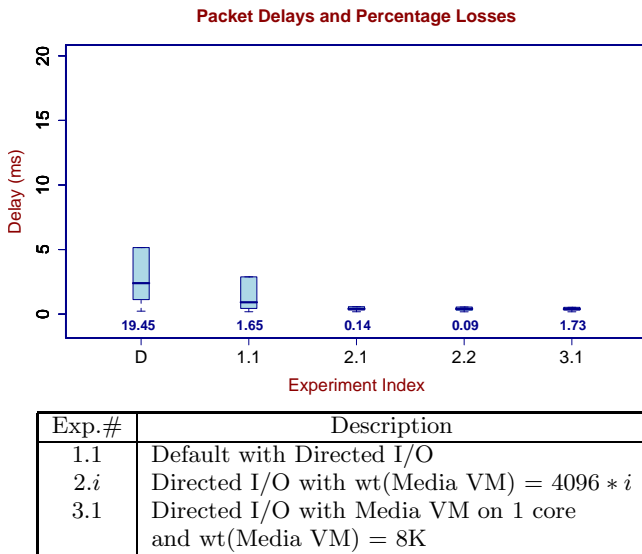


Figure 13: 2-Core Turnkey Configuration with VMM Bypass

It can be observed that Experiment 1.1, which used VMM bypass for I/O for the Media VM, gave a significant improvement in both packet loss and delay over Experiment D (which is the same as the 2-Core default configuration with 720 concurrent streams in Figure 12). Adjusting the weight of the Media VM (Experiment set 2.i) resulted in further improvement with nearly zero percent packet loss even for the limited 2-Core hardware configuration.

6.4.4 Summary of Turnkey Deployment Experiments

In conclusion, the turnkey deployment experiments showed that a 4-Core system performed well under the workloads needed for a small to medium-sized enterprise. In case of a

2-Core deployment, VMM bypass for I/O where the Media VM was given direct access to a network card, gave significant improvement in results which were improved further by increasing the weight assigned to the Media VMs.

7. CONCLUSIONS

Server consolidation using virtualization is gaining significant adoption in data centers. The deployment of enterprise telephony solutions in virtualized environments is of interest to reduce the footprint and to provide energy efficient solutions. However, the near real-time requirements of enterprise IP telephony applications pose unique challenges for deploying such applications in virtualized environments.

In this work we first modeled a medium-scale deployment of an enterprise IP telephony workload. We identified two typical scenarios for the deployment of enterprise telephony servers under virtualized environments: Mixed Workload deployment and Turnkey deployment. We then developed a methodology of evaluating these deployments by identifying key scheduler parameters and metrics for evaluating the performance of the near real-time requirements of the Media VM. The three key parameters (a) the VMM scheduler, (b) CPU configuration, and (c) device and network configurations were tuned to minimize the packet losses and delays incurred by the media streams. We performed various experiments to clearly understand the impact of these parameters on I/O performance. We identified configurations that proved to give optimal performance against the default or bad configurations.

Our observation is that the para-virtualized model of VM deployment is best suited for enterprise IP telephony applications. Further, we observe that while accommodating enterprise IP telephony, especially media applications, in a virtualized setup can be challenging, with appropriate tuning, it is possible to get reasonable performance for medium scale deployments. Xen provides sufficient tunability. However, care is required to choose appropriate parameters. In addition, in some instances, direct I/O access to the guest domain is essential to get acceptable performance and no amount of scheduler tuning can mitigate the I/O effect. In particular, our observations show that (i) Dom0 is a critical component and needs sufficient CPU resources for handling requests on behalf of all the VMs in a timely manner, (ii) increasing the weights of Dom0 and Media VM with respect to CPU-bound VMs gave reasonably good performance with minimal losses, (iii) pinning the CPU-bound VMs away from the Dom0 and Media VMs improves I/O performance, (iv) capping the CPU-bound VMs to strictly allow usage of only their allocated share of resources gave better results, and (v) VMM bypass configurations show great improvements over the traditional I/O model.

To our knowledge, this is the first time that an enterprise IP telephony workload was developed and evaluated in a virtualized environment. We believe that the results of this work will be very useful for telecommunication solution providers who wish to deploy their applications in virtualized environments. It will serve as a guideline for IT managers and help them deploy their IP telephony applications in mixed-workload environments as well as turnkey deployment environments. This work also provides feedback to VMM tech-

nology developers about possible areas of I/O performance improvement and how VMMs perform with near real-time workloads. Recently, there has been a significant amount of research and development to improve I/O performance in VMMs and our results should provide useful data to advance this effort.

8. REFERENCES

- [1] “Kernel-based Virtual Machine (KVM) for Linux.” <http://www.linux-kvm.org>.
- [2] VMware Inc., “VMware ESX and VMware ESXi.” <http://www.vmware.com/products/vi/esx>.
- [3] “Xen 3.3.” <http://www.xen.org/files/xen3.3press.pdf>.
- [4] Intel, “Intel virtualization technology (Intel VT).” <http://www.intel.com/technology/virtualization/>.
- [5] AMD, “AMD virtualization technology (AMD-V).” <http://www.amd.com/virtualization>.
- [6] P. Apparao, R. Iyer, X. Zhang, D. Newell, and T. Adelmeyer, “Characterization & analysis of a server consolidation benchmark,” in *VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pp. 21–30, 2008.
- [7] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. 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*, pp. 13–23, 2005.
- [8] A. Menon, A. L. Cox, and W. Zwaenepoel, “Optimizing network virtualization in Xen,” in *ATC '06: Proceedings of USENIX '06 Annual Technical Conference*, pp. 2–2, 2006.
- [9] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 164–177, 2003.
- [10] X. Zhang and Y. Dong, “Optimizing Xen VMM based on Intel virtualization technology,” in *ICICSE '08: Proceedings of the 2008 International Conference on Internet Computing in Science and Engineering*, pp. 367–374, 2008.
- [11] P. Apparao, S. Makineni, and D. Newell, “Characterization of network processing overheads in Xen,” in *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, p. 2, 2006.
- [12] H. Oi and F. Nakajima, “Performance analysis of large receive offload in a Xen virtualized system,” *Computer Engineering and Technology, International Conference on*, vol. 1, pp. 475–480, 2009.
- [13] P. Willmann, J. Shafer, D. Carr, A. Menon, S. Rixner, A. L. Cox, and W. Zwaenepoel, “Concurrent direct network access for virtual machine monitors,” in *HPCA '07: Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pp. 306–317, 2007.
- [14] J. R. Santos, Y. Turner, G. Janakiraman, and I. Pratt, “Bridging the gap between software and hardware techniques for I/O virtualization,” in *ATC'08: USENIX 2008 Annual Technical Conference*, pp. 29–42, 2008.
- [15] R. Jones, “Netperf: A network performance monitoring tool.” <http://www.netperf.org/>.
- [16] “NetXen.” <http://www.netxen.com/>.
- [17] “Crossbow.” <http://opensolaris.org/os/project/crossbow/>.
- [18] “Intel VMDq.” <http://www.intel.com/technology/platform-technology/virtualization/VMDq-whitepaper.pdf>.
- [19] H. Raj and K. Schwan, “High performance and scalable I/O virtualization via self-virtualized devices,” in *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing*, pp. 179–188, ACM, 2007.
- [20] J. Liu, W. Huang, B. Abali, and D. K. Panda, “High performance VMM-bypass I/O in virtual machines,” in *ATC '06: Proceedings of USENIX '06 Annual Technical Conference*, pp. 3–3, 2006.
- [21] S. Thibault, “Stub domains,” in *Xen Summit*, June 2008.
- [22] D. Ongaro, A. L. Cox, and S. Rixner, “Scheduling I/O in virtual machine monitors,” in *VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pp. 1–10, 2008.
- [23] 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,” in *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*, pp. 126–136, 2007.
- [24] G. Liao, D. Guo, L. Bhuyan, and S. R. King, “Software techniques to improve virtualized I/O performance on multi-core systems,” in *ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pp. 161–170, 2008.
- [25] Microsoft Inc., “Virtualization with Hyper-V.” <http://www.microsoft.com/windowsserver2008/en/us/hyperv.aspx>.
- [26] J. Harper and A. Liguori, “HVM I/O performance,” in *Xen Summit*, April 2007.
- [27] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A transport protocol for real-time applications.” <http://tools.ietf.org/html/rfc3550>.
- [28] E. Ackaouy, “New CPU scheduler w/ SMP load balancer.” <http://lists.xensource.com/archives/html/xen-devel/2006-05/msg01315.htm>.
- [29] Intel, “Intel virtualization technology for directed I/O.” <http://www.intel.com/technology/itj/2006/v10i3/2-io/3-vmm-software-architecture.htm>.
- [30] Xensource, “Xen wiki webpage for Xen scheduling.” <http://wiki.xensource.com/xenwiki/Scheduling>.