IEEE Computer Society

Proceedings of the
Sixth Annual Workshop on Workload Characterization

Austin, Texas
October 27, 2003

Sponsored by the IEEE Computer Society and the
Technical Committee on Computer Architecture

An Analysis of Disk Performance in VMware ESX Server Virtual Machines

Irfan Ahmad, Jennifer M. Anderson, Anne M. Holler, Rajit Kambo, Vikram Makhija
VMware Inc.
Palo Alto, CA 94304 USA
{irfan,jennifer,anne,rajit,vmakhija}@vmware.com

Abstract

VMware ESX Server is a software platform that efficiently multiplexes the hardware resources of a server among virtual machines. This paper studies the performance of a key component of the ESX Server architecture: its storage subsystem. We characterize the performance of native systems and virtual machines using a series of disk microbenchmarks on several different storage systems. We show that the virtual machines perform well compared to native, and that the I/O behavior of virtual machines closely matches that of the native server. We then discuss how the microbenchmarks can be used to estimate virtual machine performance for disk-intensive applications by studying two workloads: a simple file server and a commercial mail server.

1. Introduction

VMware ESX Server is a software platform that enables multiple virtual machines to share the hardware resources of a single server. Each virtual machine (VM) is securely isolated from all other VMs, and is given the illusion that it is running directly on a dedicated server. Virtual machines are gaining in popularity, as they present attractive solutions to problems in areas such as server consolidation, high availability and failover, and software development and testing.

IBM developed the concept of virtual machines in the 1960's to allow concurrent and interactive access to mainframe computers [1]. Virtual machines provide users with the ability to run multiple copies of potentially different operating systems, along with benefits like isolation and resource sharing. VMs are also convenient abstractions of workloads, since they cleanly encapsulate the entire state of a running system, including both user-level applications and kernel-mode operating system services. The current version of VMware ESX Server virtualizes servers based on the Intel IA-32 architecture [2].

The performance of applications running within VMs is a significant factor in their adoption. VMware ESX Server was designed for high performance, and its architecture is streamlined to provide high-speed I/O. In this paper, we focus on one component of ESX Server's I/O architecture, its storage subsystem. We look at the characteristics of a series of disk microbenchmarks on several different storage systems: a direct-attached disk, a RAID array and a storage-area network (SAN). We compare the throughput obtained on the native server to the throughput on a virtual machine. We show that despite differences in the I/O characteristics of the different storage subsystems, the behavior of the VM matches that of the native server closely; for example, in the configurations where the native server shows higher throughput, the virtual machine shows a similar increased throughput. Next, we discuss how to apply the microbenchmarks to model virtual machine performance for disk-intensive applications, and present case studies for two workloads: a simple file server and a commercial mail server. We show that in these cases, the microbenchmark data, in conjunction with characteristics of the native application, can be used to generate an accurate estimate of the application's performance in a virtual machine.

The rest of this paper is organized as follows. Section 2 presents an overview of the ESX Server architecture. Section 3 describes our experimental setup and presents the data from the disk microbenchmarks on three different storage subsystems: a direct-attached disk, a RAID array and a SAN. Section 4 discusses how we model the virtual machine performance of a simple file system workload, and then Section 5 shows how we model the performance of a more complex application, a commercial mail server benchmark. We present related work in Section 6 and our conclusions and future work in Section 7.

2. VMware ESX Server architecture

Figure 1 illustrates the organization of VMware ESX Server. ESX Server is a thin software layer

(shown in gray in the figure) that runs directly on the physical machine. It has direct access to the physical hardware of the server, enabling high-speed I/O as well as complete control over resource management.
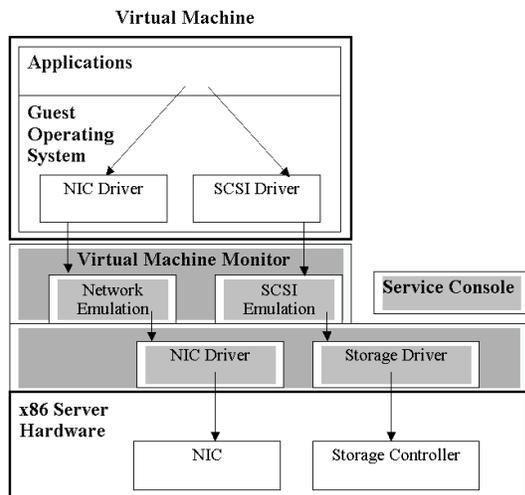

Figure 1. VMware ESX Server architecture. The components of ESX Server are shown in gray. ESX Server runs directly on the native x86 hardware and includes a virtual machine monitor that virtualizes the CPU. The service console is used to handle booting and administration. To provide high performance disk and network I/O, ESX Server implements an efficient path from the I/O requests in the applications running in the virtual machines down to the physical devices.

Physical devices, such as network interface controllers (NICs) and disks, are presented as virtual devices to the virtual machines running on ESX Server. The set of potential virtual devices seen by the guest operating system is the same, regardless of the underlying physical hardware. In particular, each virtual disk is presented as a SCSI drive connected to a SCSI controller. This device is the only storage controller used by the guest operating system, despite the wide variety of SCSI, RAID and Fibre Channel adapters that might actually be used on the native server.

The *Virtual Machine Monitor* (VMM) is responsible for virtualizing the IA-32 CPU. Under certain conditions the VMM can run the guest instructions directly on the underlying physical processor. This mode of execution, called *direct execution*, runs at near-native speed. Otherwise, the guest instructions must be virtualized. This adds varying amounts of CPU overhead depending on the specific operation. Typically, user-level code runs in

direct-execution whereas operating system code requires virtualization. Any such overhead gets translated into higher CPU utilization, or *occupancy*.

ESX Server implements a streamlined path to provide high-speed I/O for the performance critical devices, network and disk. An I/O request issued by the guest operating system first goes to the driver in the virtual machine. For storage controllers, ESX Server emulates LSI Logic or BusLogic SCSI devices, so the corresponding driver loaded into the guest will be either an LSI Logic or BusLogic driver. The driver typically turns the I/O requests into accesses to I/O ports to communicate to the virtual devices using privileged IA-32 IN and OUT instructions. These instructions are trapped by the VMM, and then handled by device emulation code in the VMM based on the specific I/O port being accessed. The VMM then calls device independent network or disk code to process the I/O. For disk I/O, ESX Server maintains a queue of pending requests per virtual machine for each target SCSI device. The disk I/O requests for a single target are processed in a round-robin fashion across VMs by default. The I/O requests are then sent down to the device driver loaded into ESX Server for the specific device on the physical machine.

3. Disk microbenchmarks

In this section, we present results for a series of disk microbenchmarks run on three different storage subsystems connected to different servers. We ran the experiments both natively and in virtual machines on ESX Server 2.0, the latest released version. We first compare the results across the different systems and show that they have widely varying behavior, and then compare native to VM on the same system to quantify VM disk subsystem performance.

3.1. Microbenchmark software

We used *Iometer* [3] (version dated 2003.05.10) for the measurement and characterization of native and virtual machine disk subsystems. Iometer is an I/O subsystem measurement and characterization tool for single and clustered systems. It can be used both as a *workload generator* (it performs I/O operations in order to stress the system) and a *measurement tool* (it examines and records the performance of its I/O operations and their impact on the system). Iometer can be configured to emulate the disk or network I/O load of a program or benchmark, or can be used to generate entirely synthetic I/O loads. It is a standard industry benchmark and the sources are available, allowing us to study the code when needed.

We created an Iometer configuration file that varied three parameters: block size for I/O operations,

percentage of read operations and degree of randomness. We collected the following data points in the parameter space:

$$\left(\frac{Blocksize}{1K,4K,8K,16K}\right) \times \left(\frac{\%read}{0,25,50,75,100}\right) \times \left(\frac{\%random}{0,25,50,75,100}\right)$$

We then plotted and compared the surfaces of this data on different systems.

3.2. Experimental setup

We performed experiments on several machine configurations to compare native vs. VM performance. Relevant system details are shown in Table 1:

Table 1. Machine and storage subsystem specifications

Machine Name	Mojave	Thar	Atacama
Storage System	Direct-Attached Disk	RAID Array	Storage-Area Network (SAN)
CPU	Dual-P4 Xeon @ 1.8 GHz	Quad-P4 Xeon MP @ 1.6 GHz	Dual-P4 Xeon @ 2.0 GHz
Total System Memory	2 GB	8 GB	1 GB
Native OS	Linux 2.4	Win2k Adv Server (SP3)	Win2k Adv Server (SP3)
Data Disk (ESX Server and native)	Seagate Barracuda ST336938 LW; SCSI 10K RPM 34 GB	IBM EXP300; 14-disk RAID-0; SCSI 10K RPM 36 GB drives	Fujitsu Siemens S60; 5-disk RAID-0; FC 10K RPM 18 GB drives
Controller	Adaptec 7892P	IBM ServeRaid 4Mx	Qlogic 2312
Protocol	Ultra160 SCSI	Ultra160 SCSI	SAN 1Gbps Fiber-Channel

Identical workload configurations were used for each experiment and the same physical disks were used to compare VM and native throughput on each machine. The guest operating system installed in the virtual machines was the same as the operating system in the native experiments on each server. Also, both the VM and native machines were configured to use equal amounts of memory (128 MB for Mojave, 256 MB for Thar, and 256 MB for Atacama) when running the experiments. The Mojave and Thar VMs used the VMware BusLogic virtual SCSI device with the BusLogic driver supplied with VMware Tools and the Atacama VM used the VMware LSI Logic virtual SCSI device with the vendor-supplied driver. Each Iometer experiment was set up with one worker and the number of outstanding I/Os (28 for Mojave, 47 for Thar, and 64 for Atacama) was chosen to expose available hardware parallelism. The think time (delay) between I/Os was set to zero in the Iometer

specification, and each data point was run for three minutes to give the throughput values sufficient time to stabilize. Finally, in each case Iometer was configured to access the disk through the file system rather than directly through a raw or physical device interface.

3.3. Disk microbenchmark data

Different storage subsystems behave differently under varying loads. By studying their behavior over a large parameter space, we can get a good understanding of how the systems behave across a variety of workloads. We present the data as *throughput surfaces*: 3-d plots of disk throughput, for different degrees of randomness and reads versus writes. Figure 2 and Figure 3 show the native and virtual machine throughput surfaces, respectively, for Mojave attached to a single Seagate Barracuda disk. Figure 4 and Figure 5 show the throughput characteristics for Thar attached to an IBM EXP300 disk array, and Figure 6 and Figure 7 show the same for Atacama attached to a Fujitsu Siemens (FSC) SAN. Each graph in the figures shows the throughput surfaces for a given blocksize. On Mojave, data for the 1 KB block size was not collected due to a limitation in the Linux version of Iometer. The graphs plot the throughput (in megabytes per second) on the z-axis, with the percentage of reads versus writes on the x-axis and percentage of random versus sequential accesses on the y-axis. Data were collected for those points on the graph labeled on the x- and y-axes.

3.4. Hardware and workload characteristics

The throughput surfaces obtained with the Iometer microbenchmarks are characteristic of the combination of hardware and workload. By looking at the graphs from the native runs, we can see how the systems compare in both behavior and absolute throughput. For example, the single directed-attached Barracuda disk (Figure 2) exhibits similar behavior to the FSC SAN system (Figure 6) in that the throughput surfaces have the same basic shape, but the direct-attached disk has lower overall throughput than the FSC SAN. On the other hand, the FSC SAN system (Figure 6) and the IBM EXP300 disk array (Figure 4) show marked differences in behavior. The FSC SAN has lower throughput in the random I/O case than the EXP300 disk array, but much higher sequential throughput.

In order to explain the shapes of the throughput surfaces, we consider the way Iometer is generating I/Os. As the %read parameter is varied, Iometer computes the probability that the next operation is a

read as $p(read) = \frac{\%read}{100}$, and that it should seek to a random offset as $p(seek) = \frac{\%rand}{100}$. For example, for $(\%read = 50, \%rand = 0, blocksize = 16KB)$, Iometer will perform completely sequential I/O on 16 KB blocks where roughly half the operations will be reads.

This helps explain the unusual shape of the throughput surface along the $\%rand = 0$ (sequential I/O) slice. The direct-attached disk (Figure 2) and the FSC SAN (Figure 6) display this most prominently. Along the indicated slice of sequential I/O, the throughput peaks at 100% writes and 100% reads, but mixed (interspersed) reads and writes appear to suffer.

There are multiple factors that account for this behavior: peculiarities of the workload generator itself or caching behavior for reads versus writes. In the sequential case of mixed reads and writes, Iometer does sequential I/O accesses and chooses for each transaction whether it will be a read or a write based on the access specification provided to it. In other words, the reads and writes are not two separate sequential streams. Depending upon how disk controllers and/or drives reorder accesses or perform caching optimizations, this can affect the throughput significantly. We believe that the read-ahead and write caching algorithms might interfere in this case resulting in lower throughput.

Even though the throughput surfaces have unusual characteristics, these are inherent to the combination of the hardware and the workload generator, and thus do not affect the ability of microbenchmarks to model application performance in virtual machines.
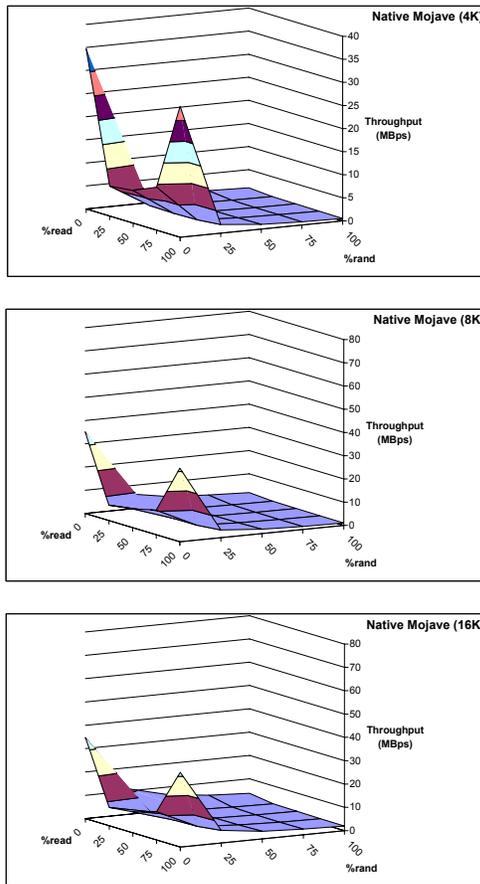

Figure 2. Native Iometer results with a direct-attached disk on Mojave. The graphs plot the disk throughput for different percentages of reads and degrees of randomness for three different block sizes: 4 KB, 8 KB and 16 KB, respectively.
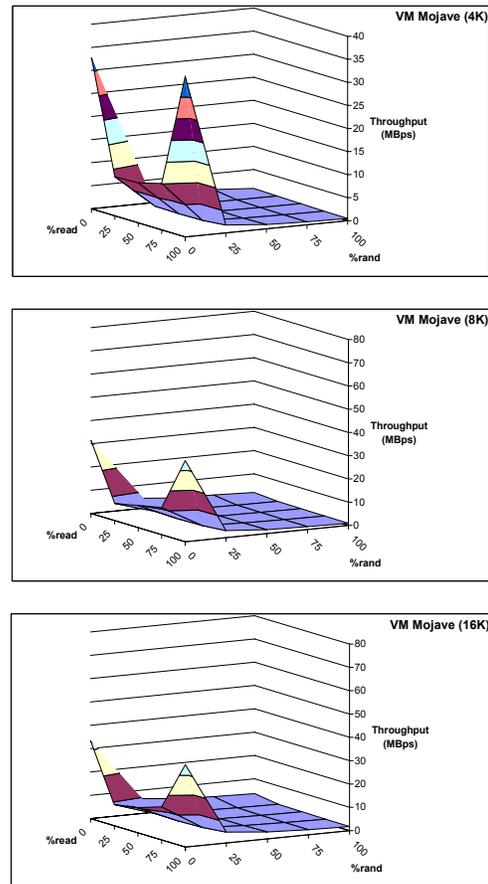

Figure 3. VM Iometer results with a direct-attached disk on Mojave. The graphs plot the disk throughput for different percentages of reads and degrees of randomness for three different block sizes: 4 KB, 8 KB and 16 KB, respectively.
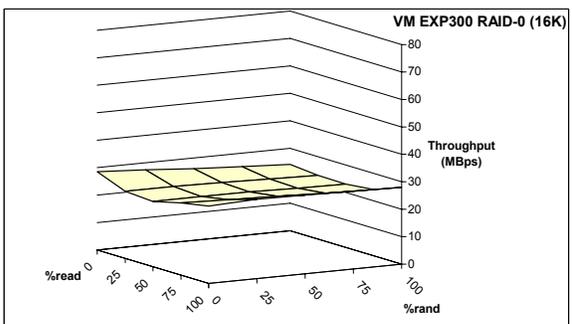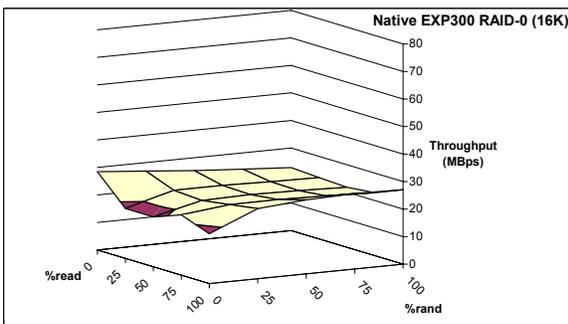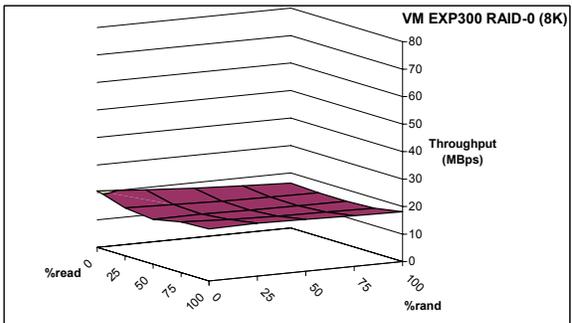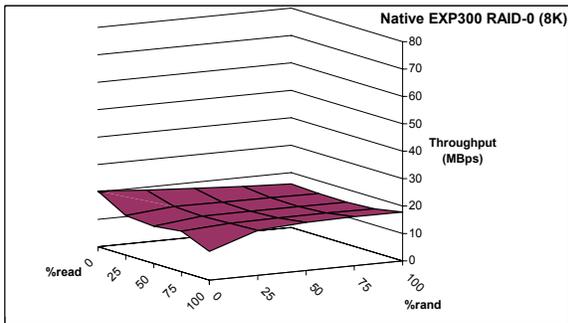
Native EXP300 RAID-0 (1K)

VM EXP300 RAID-0 (1K)

Native EXP300 RAID-0 (4K)

VM EXP300 RAID-0 (4K)

Native EXP300 RAID-0 (8K)

VM EXP300 RAID-0 (8K)

Native EXP300 RAID-0 (16K)
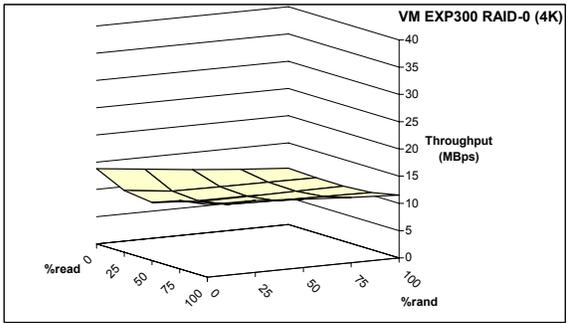
VM EXP300 RAID-0 (16K)

Figure 4. Native Iometer results with an EXP300 RAID array on Thar. The graphs plot the disk throughput for different percentages of reads and degrees of randomness for four different block sizes: 1 KB, 4 KB, 8 KB and 16 KB, respectively.
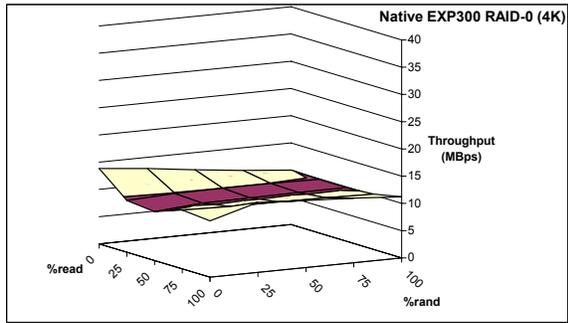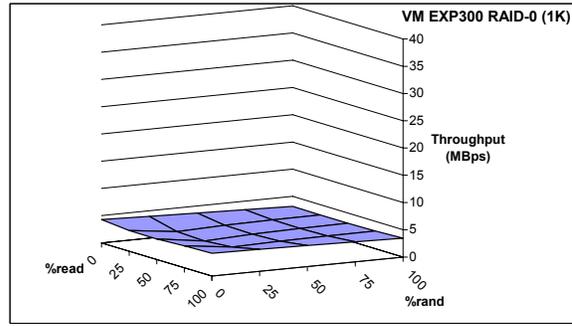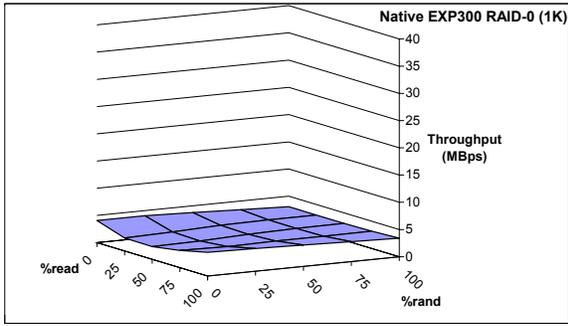
Figure 5. VM Iometer results with an EXP300 RAID array on Thar. The graphs plot the disk throughput for different percentages of reads and degrees of randomness for four different block sizes: 1 KB, 4 KB, 8 KB and 16 KB, respectively.

Figure 6. Native Iometer results with a FSC SAN on Atacama. The graphs plot the disk throughput for different percentages of reads and degrees of randomness for four different block sizes: 1 KB, 4 KB, 8 KB and 16 KB, respectively.

Figure 7. VM Iometer results with a FSC SAN on Atacama. The graphs plot the disk throughput for different percentages of reads and degrees of randomness for four different block sizes: 1 KB, 4 KB, 8 KB and 16 KB, respectively.
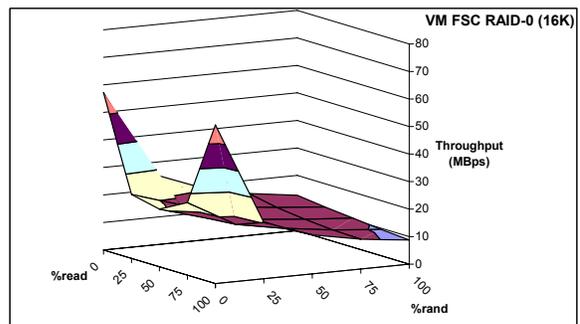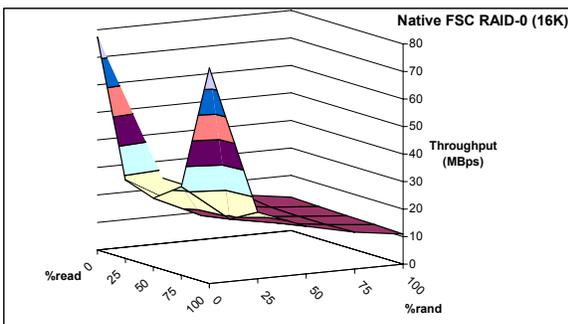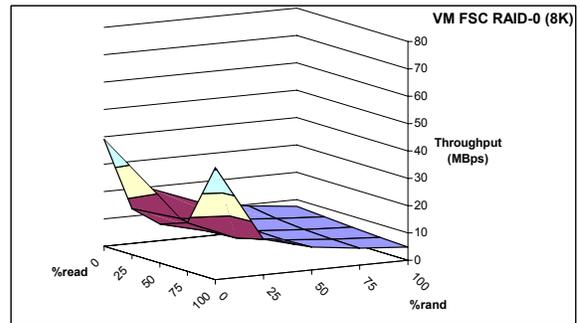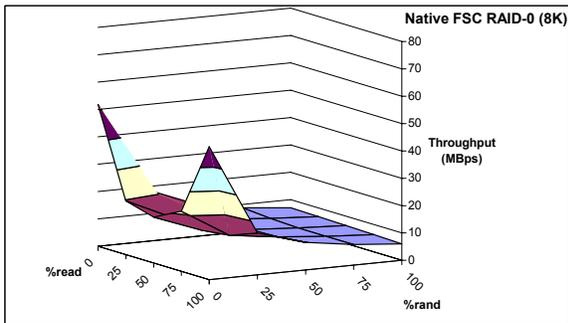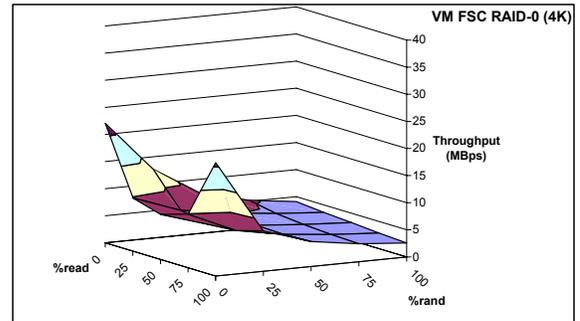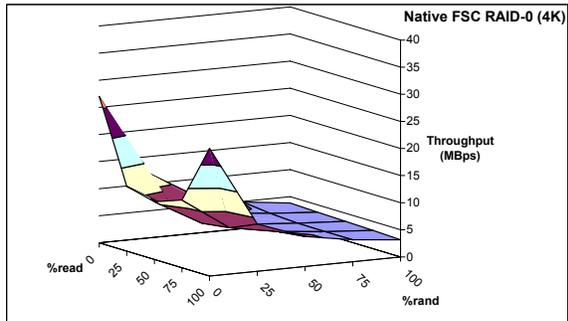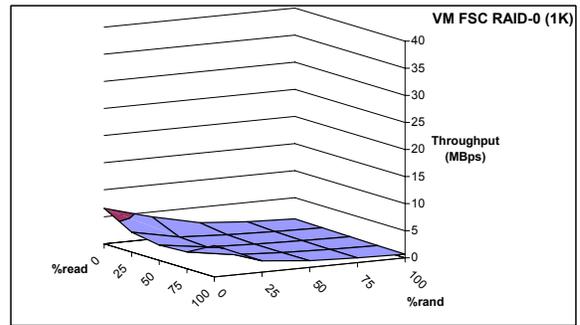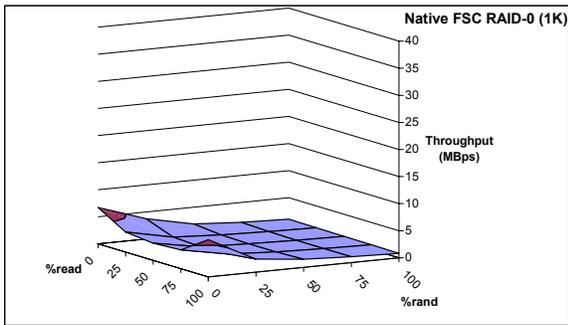
3.5. Virtual machine performance

We compare the difference in I/O performance between native and virtual machine configurations over the parameter space described in Section 3.1. Figure 3, Figure 5 and Figure 7 show the throughput of a single VM running Iometer on ESX Server. The workload specifications used are exactly the same as in the corresponding native case (Figure 2, Figure 4 and Figure 6). Note that, for each of the systems, the characteristic shape formed by the data points is retained between the native and VM cases.

Though the shapes match in the two cases, the absolute throughput between the two does vary, depending on characteristics of the hardware and the particular workload. Figure 8 shows the ratios between the corresponding VM and native throughputs, sorted in ascending order of ratio value. (There are fewer data points for Mojave because experiments for the 1 KB block size case could not be run due to a limitation in the Linux version of Iometer.) On Mojave, VM throughput ranges between 0.6 and 1.3 of native, with an average ratio of 0.95; on Thar, it ranges between 0.98 and 1.5, with an average of 1.08; and on Atacama, it ranges between 0.75 and 1.1 of native, with an average of 0.92. Since virtual machine technology introduces additional layers for each disk I/O operation (see Section 2), VM overhead over native is expected. However, it is counter-intuitive that disk I/O throughput can be higher when running in a VM than when running natively.
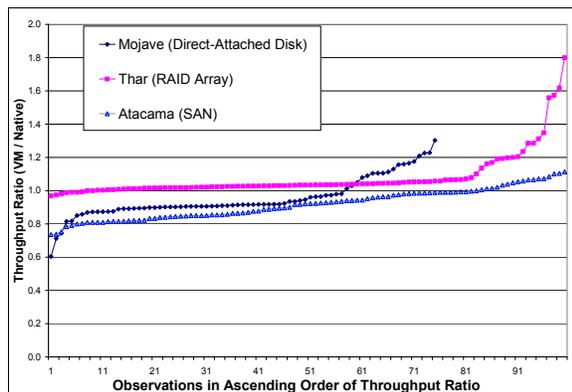

Figure 8. VM/Native throughput ratios. The graph plots the ratio of VM/Native throughput for each data point collected on the three different systems. The points are plotted in ascending order by throughput ratio.

There are two reasons that ESX Server's disk I/O throughput can sometimes be higher than native. First, VMware virtual SCSI devices support a larger SCSI device queue in the guest than that typically provided for native SCSI devices. ESX Server virtual SCSI devices advertise themselves as being able to handle wider requests than hardware SCSI devices typically do, so wide requests do not have to be split before being queued. Also, they allow for more queue slots than are generally available in native drivers. A larger SCSI queue potentially exposes more concurrency, allowing ESX Server more opportunity to keep the SCSI device busy. Second, ESX Server is highly tuned for SCSI hardware; ESX Server drivers may use larger values for queue depth and maximum transfer size for certain adapters than native drivers, based on the characteristics of that hardware. Native operating systems and drivers could potentially match these ESX Server advantages.

4. Fileserver workload modeling

For our fileserver workload, we used dbench [4]. Dbench was created by Andrew Tridgell to capture the file system operations from the Netbench benchmark run against a Linux host. Dbench exercises only the I/O calls that an smbd server produces when executing Netbench; it does not do any of Netbench's networking operations. Version 2.0 of dbench was used for the runs in this study.

Dbench I/O behavior with respect to disk operations is complicated because the benchmark creates and deletes the files that it uses, and depending on how quickly these operations occur with respect to the frequency with which cached dirty buffers are written to disk, disk I/O on behalf of the files may be completely avoided. (Because of this, dbench can be used as a memory benchmark by tuning system parameters to skip write-back completely.) The higher the system load, the more load the time-based write-back operations add to the disk I/O observed during the benchmark run. For our runs, we used the default system parameter values with respect to write-back and we chose a system load high enough to ensure sustained heavy disk I/O traffic during the benchmark

The system labeled Mojave in Table 1 was used for native and VM dbench runs of 30, 40, and 50 clients. The three workloads were comprised of 50% read operations with an average block size of 21 KB. Their randomness level is unknown; it is not expected to be 0% because of the large number of clients simultaneously accessing files.

We can use our Iometer results for Mojave to predict the performance of dbench in a VM. We compute the throughput ratio of VM to native for each measurement of 16K block size, with 50% read, and 25%, 50%, 75%, and 100% randomness. It is about 90% for each of the four cases. Then, we multiply that value by the native dbench throughput to project the

VM dbench throughput. Table 2 shows the Native performance, along with the projected and measured VM performance. The projected VM performance closely matches the measured VM performance.

Table 2. Native and VM dbench disk I/O results

	Native	Projected VM	Actual VM	Projected /Actual VM
	MB/s	MB/s	MB/s	
30 clients	7.89	7.10	6.85	1.04
40 clients	7.16	6.44	6.40	1.01
50 clients	7.19	6.47	6.06	1.07

These benchmark runs show that dbench VM I/O throughput performance is about 87% of native. For this processor/disk combination, the workload is I/O bound and native CPU utilization does not exceed 4%, even for 100 client runs (not shown). CPU utilization is higher for the VM, at 10%. This increase in CPU occupancy is due to ESX Server's virtualization overheads. In this case, the virtualization overheads do not affect the throughput because there is enough CPU headroom to absorb them, and VM performance remains close to native performance.

That the projection from Iometer performance works in this case may not be surprising, since dbench could be considered a microbenchmark like Iometer, albeit one in which the workload was drawn from tracing a particular real-world application. In an actual Netbench run, disk I/O throughput could be influenced by non-disk attributes of the workload and its performance in a VM. In the next section, we examine a complex benchmark workload with a large disk I/O component and explore how various areas of virtualization affect the prediction of VM performance from native performance characteristics.

5. Mail server workload modeling

In this section, we describe how we model the performance of a commercial mail server application running in a virtual machine. We first derive a set of parameters for the Iometer I/O generator that are used to simulate the workload. We then apply the microbenchmark data, coupled with native performance characteristics, to estimate VM performance on two different system configurations.

The workload we used was an existing benchmark of a commercial mail system. It models users performing various tasks, including reading and sending mail, deleting messages, and opening and closing mailboxes. For a given number of users, the test measures two values: completed mail operations per minute and average response time at maximum

capacity. The benchmark's score is a capacity metric that is a function of the number of simultaneous users that can be supported at an acceptable response time.

5.1. Experimental Setup

We ran the workload on three separate setups (see Table 1 for machine details):

1. Thar-native – Windows 2000 Advanced Server running natively on Thar with a RAID array.
2. Thar-VM – Windows 2000 Advanced Server running in a VM on Thar with a RAID array.
3. Atacama-VM – Windows 2000 Advanced Server running in a VM on Atacama with a SAN.

The Thar-native setup was used as the reference system for the model, which was then applied to predict the performance of the two virtual machine configurations. The native and VM experiments used 1 GB of memory while running the mail server workload and 256 MB while running the Iometer model. Both the Thar and Atacama VMs used the VMware BusLogic virtual SCSI device and the BusLogic drivers supplied with VMware tools in the guest operating system.

5.2. Iometer specification derivation

On Thar-native, the mail workload behaves as follows:

Table 3. Mail server workload characteristics

IO/sec	792
Reads/sec	512
Writes/sec	280
MB/sec	8.5
% CPU user time	70
% CPU system time	30
Capacity metric	7122
Simultaneous users	5000
Average # of worker threads	50

The values in Table 3 were either generated by the application or collected using `perfmon,` the Windows performance monitor.

The Iometer microbenchmark data in Section 3.3 characterizes the throughput of an I/O device as a function of just three variables: %read, %random and blocksize. The %read and blocksize for the mail server workload can be easily measured: I/Os issued are distributed as 64% reads and 36% writes, and the average I/O request size is approximately 10.9 KB (8.5 MB/sec / 792 IO/sec). To determine the degree of randomness, we note that the mail server services requests from a large number of clients. In our tests, we used several thousand clients all accessing separate database files on disk through a pool of worker threads

on the server. Due to the high degree of multiprogramming, the effective distribution of I/O on disk is random.

The mail server workload is more complex than the I/O microbenchmarks and involves significant computation between I/O requests. We observed that for this workload, the network is not a bottleneck and that, depending on the hardware, either the CPU or the storage subsystem is the limiting factor that determines throughput. Moreover from our experience running the mail server workload, we were able to determine that the capacity metric of the workload is directly proportional to the number of I/Os it generates. Using this observation we construct a simple model of the time spent to execute a single mail transaction and use it to predict the overall throughput. We model the workload by representing the two main components, CPU and disk I/O. We approximate the computation time as a think time parameter in the Iometer specification and measure the I/O directly from the Iometer results.

We derived a think time value by varying the think time parameter in Iometer running on the reference system until we were able to match the I/O rate of 792 IO/sec, which is the rate achieved by the mail server workload. This resulted in a think time of 50 ms. In addition to think time, we also introduce parallelism in our Iometer specification and use 50 Iometer worker threads in place of the mail server worker threads.

Thus the full set of parameters for the Iometer model are: 64% reads, 100% random, 10.9 KB blocksize, 50 ms think time and 50 worker threads. Note that we created a model based on characteristics of the workload running natively on Thar, and applied it to predict the throughputs for virtual machines running on both Thar and Atacama. However, the choice of the reference system was arbitrary, and we could just as easily have used Atacama.

5.3. Virtual machine performance prediction

The goal for our model is to predict the capacity metric of the mail server workload running inside a virtual machine. Each thread within the model mimics a thread on the mail server. It performs some amount of work and then issues a single synchronous disk I/O. Upon completion of this I/O it repeats this sequence (note that if the I/Os were asynchronous, the I/O completion time would overlap with the computation time and the latencies would be more difficult to reason about). We do not know exactly what work the threads perform. However, from data collected in Section 5.2, we do know that the average

latency required to perform this computation is 50 ms on Thar-native.
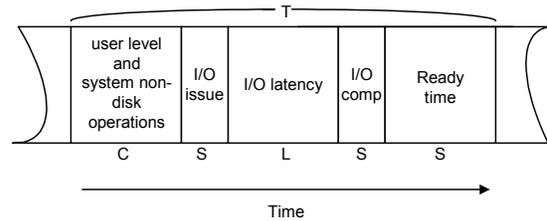

Figure 9. Breakdown of latencies in a single mail transaction

The sequence of events for a single mail server operation is shown in Figure 9. For brevity, we assign the following variables (shown as labels in the figure) to represent the time spent in each of the key components of the transaction:

- C is the latency (in ms) of executing user level code plus all system level code excluding disk I/O operations. This system code includes networking operations, memory management and other system calls.

- S is the latency (in ms) of executing system level code to perform disk I/O operations plus time spent in the process ready queue.

- L is the latency (in ms) of the actual disk I/O operation.

- T is the sum of the previous three quantities and represents the time taken by a single thread to execute one mail transaction.

Thus the total time for a single transaction is:

$$T = C + S + L$$

By running the Iometer model constructed in Section 5.2 on the reference system (Thar-native) we are able to derive these values for the reference system. We have 792 IO/sec distributed among 50 threads, and each thread performs 15.8 IO/sec. Thus, a single I/O operation for a single thread takes 63.1 ms and the reference time is $T_R = 63.1$ ms. Similarly we observe that the think time of 50 ms represents the time spent in user level code plus system level code for non-disk I/O related operations, hence $C_R = 50$ ms. The average response time measured on the reference system by running the Iometer model is $L_R = 12$ ms. Since we know T_R, C_R and L_R, we can obtain S_R by simple subtraction: $S_R = 1.1$ ms. To summarize, for the reference system Thar-native:

$$T_R = C_R + S_R + L_R = 50 + 1.1 + 12 = 63.1 \text{ ms}$$

We now apply the model to predict the performance of the mail server workload running in a VM on the same machine as the reference. We

calculate the time T_V spent by a thread executing the model workload in a virtual machine, and then scale the reference capacity metric by the ratio $\frac{T_R}{T_V}$ to obtain the predicted capacity metric.

To compute the transaction time in a virtual machine, we first run the Iometer model in the VM and measure the I/O rate as was done on the reference system. However, there are key differences between running the workload natively and in a virtual machine. The latency of user level code and system non-disk operations (C_V) is a derived quantity that is approximated in the model by the think time parameter to Iometer. When running in a virtual machine, any virtualization overheads incurred by this computation must be taken into account. Similarly, if the model is applied to predict the performance of a VM running on a different machine, differences in system architecture will also impact the computation time. We factor these differences into the calculation of the transaction time T_V after running the Iometer model. Another option would have been to run the model with an adjusted think time to account for the computation time differences. However our experiments showed that due to limitations in Iometer, we were unable to accurately model the think time with the required granularity. Note that the other components of the transaction time in a virtual machine (S_V, L_V) are obtained directly from Iometer running in the virtual machine, so they already have any virtualization overheads and system differences factored in.

We introduce the variable V to account for virtualization overheads and the variable F to account for differences across machines. As differences in system architecture and memory latencies across machines make it difficult to compare the compute performance of different systems exactly, we use the CPU clock rate as an approximation. The time for each I/O in Iometer running in the virtual machine is given by $T_V^{'}$. Thus, the formula for T_V, the total time for a mail server transaction, is:

$$T_V^{'} = C_V + S_V + L_V$$

$$T_V = (C_V + V) \cdot F + S_V + L_V$$

By running the Iometer model on Thar-VM, we obtain the values shown in Table 4. Using these values we derive the following (all units in ms): $T_V^{'} = 72.5, C_V = 50, S_V = 9.5, L_V = 13$. Because the VM is running on the same machine as the reference system, the CPU speed is the same and $F = 1$.

We now need to account for the virtualization overhead to calculate the final value of T_V. We note that by virtue of running the Iometer model in a virtual machine, the virtualization overheads are already accounted for in S_V and L_V, so we only need to account for the overhead in C_V. Our experiments have shown that the mix of user level code and system non-disk operations in this workload typically incurs a 9.4% overhead when run in a virtual machine. The user-level code, which accounts for the majority of the computation (see Table 3), executes in direct execution mode within the VM and runs at near-native speed. Only the system non-disk component of C_V incurs any overhead. Applying the virtualization overhead gives $V = 0.094 C_V$, which in this case translates to 4.7 ms. Using this information we arrive at:

$$T_V = (50 + 4.7) \cdot 1 + 9.5 + 13 = 77.2 \text{ ms}$$

We now use the ratio $\frac{T_R}{T_V} = \frac{63.1}{77.2} = 0.82$ to scale the reference capacity metric and compare with the actual value:

Table 4. Thar-VM mail server workload prediction

	Native	Thar-VM Model	Thar-VM Actual
Machine	Thar	Thar	Thar
IO/sec	792	690	700
MB/sec	8.5	7.4	7.6
Average I/O response time (ms)	12	13	N/A
%CPU used for I/O	10	16.9	N/A
Capacity metric	7122	5840*	6123
* = predicted value; all others are measured			
N/A = values not reported by the mail server application			

The difference between the actual and predicted values of the capacity metric is 4.6%. For the mail server workload, the VM's actual performance with respect to native on Thar is 86%.

Next we applied the model to a virtual machine running on Atacama. We ran the Iometer model on Atacama-VM and observed the values shown in Table 5. Using these values we derive the following (all units in ms):

$$T_V^{'} = 90.3, C_V = 50, S_V = 13.3, L_V = 27$$

Since this virtual machine is running on a completely different machine than the reference system, the CPU scale factor F is calculated using the CPU speeds for the systems listed in Table 1:

$$F = \frac{1.6}{2.0} = 0.8$$

The virtualization overhead $(V = 0.094 C_V)$ is

4.7 ms, and the final value of T_V :

$$T_V = (50 + 4.7) \cdot 0.8 + 13.3 + 27 = 84.1 \text{ ms}$$

We again use the ratio $\frac{T_R}{T_V} = \frac{63.1}{84.1} = 0.75$ to scale

the reference capacity metric and compare with the actual value:

Table 5. Atacama-VM mail server workload prediction

	Native	Atacama-VM Model	Atacama-VM Actual
Machine	Thar	Atacama	Atacama
IO/sec	792	554	560
MB/sec	8.5	6.0	6.0
Average I/O response time	12	27	N/A
%CPU used for I/O	10	7.0	N/A
Capacity metric	7122	5342*	4921
* = predicted value; all others are measured			
N/A = values not reported by the mail server application			

In this case the predicted values are within 8.6%. Note that the predictor works reasonably well even though the workload is run on a different machine than the reference. However, we cannot directly compare the native performance of the mail server workload to VM performance when they are run on different systems.

5.4. Summary of mail server workload modeling

In this section we have shown how we were able to predict the capacity metric of a commercial mail server workload running in a virtual machine with reasonable accuracy using a simple model workload. We achieved this by:

- Using the Iometer microbenchmark we characterized a system along three dimensions: % of I/Os that are reads, % of I/Os that are random and I/O request size.

- We then characterized the mail workload along two additional dimensions, think time and outstanding I/Os, and created an Iometer specification that approximates the mail server workload.

- Finally, using the Iometer microbenchmark and our knowledge of virtualization overheads, we created a mathematical model that would predict

the mail server capacity metric, based on the latencies involved in mail server operations.

Using a native system as reference and a simple working model of the mail workload, we were able to predict the eventual capacity of the system (M) running in a virtual machine as follows.

The reference time for a single mail server transaction is:

$$T_R = C_R + S_R + L_R$$

The time for a mail server transaction in a virtual machine is:

$$T_V = (C_V + V) \cdot F + S_V + L_V$$

Finally, the generalized predictor is then:

$$M = M_R \cdot \frac{T_R}{T_V}.$$

6. Related work

There have been a number of papers that have characterized the file system behavior of workloads [5][6][7][8][9][10]. These papers primarily focused on understanding the workload characteristics with the aim of improving overall application performance. Using synthetic workloads, such as Iometer, to model applications is a well-known technique [11][12]. In this paper, we are able to use a simple microbenchmark to compare disk performance of native and virtual machines. We then use the characterization of the workload, along with the microbenchmark model, to predict virtual machine performance.

Virtual machines were first developed in the 1960's and there is a wealth of literature describing their implementation and performance, for example [13][14][15]. A previous paper looked at virtualizing I/O devices on VMware's Workstation product [16] and focused on the networking path. The I/O architecture of ESX Server, on which this paper is based, is very different from that of VMware Workstation. VMware Workstation has a hosted architecture that takes advantage of an underlying operating system for its I/O calls; in contrast, ESX Server manages the server's hardware devices directly. This allows ESX Server to achieve higher I/O performance and more predictable I/O behavior. Finally, the mechanisms that ESX Server uses for memory resource management were introduced in [17].

7. Conclusion and future work

Virtual machines are becoming increasingly common in the data-center, and their performance is

an important consideration in their deployment. In general, virtual machine performance is influenced by a variety of factors. In this paper, we focused primarily on the performance of VMware ESX Server's storage subsystem.

We studied the performance of native systems and virtual machines using a series of disk microbenchmarks on three different storage systems: a direct-attached disk, a RAID array and a SAN. We showed that on all three systems, the virtual machines perform well compared to native, and that the I/O behavior of virtual machines closely matches that of the native server. We then presented two case studies that showed how microbenchmarks could be used to model virtual machine performance for disk-intensive applications. For these cases, the microbenchmark data, in conjunction with the characteristics of the native application, were able to estimate the application's performance in a virtual machine.

The work presented in this paper is part of a project seeking to accurately predict the performance of applications running in virtual machines for various workloads, operating systems and hardware configurations. We plan to conduct further experiments to see how our mail server workload model performs on other hardware configurations. We are also investigating ways to refine our estimation of the CPU virtualization overheads by measuring the overheads of specific operating system services, such as system calls, page faults and context switches. Finally, we are investigating ways to predict network I/O performance using a combination of microbenchmarks and application characteristics as was done in this paper for disk I/O.

8. Acknowledgements

We would like to thank Ed Bugnion, Bing Tsai and Satyam Vaghani for useful discussions on the data collected for this paper. We would also like to thank Beng-Hong Lim, Carl Waldspurger, Alex Protopopescu and the reviewers for their helpful comments.

9. References

[1] R. J. Creasy, "The Origin of the VM/370 Time-Sharing System", *IBM Journal of Research and Development*, 25(5), Sep 1981.

[2] Intel Corporation, *IA-32 Intel Architecture Software Developer's Manual, Volumes I, II and III*, 2001.

[3] http://sourceforge.net/projects/iometer

[4] http://samba.org/ftp/tridge/dbench/

[5] K. Keeton, A. Veitch, D. Obal and J. Wilkes, "I/O Characterization of Commercial Workloads", *Proc. Third Workshop on Computer Architecture Evaluation Using Commerical Workloads (CAECW-00)*, January 2000.

[6] J.K. Ousterhout, H. Da Costa, D. Harrison, J.A. Kunze, M. Kupfer and J.G. Thompson, "A Trace-driven Analysis of the UNIX 4.2BSD File System", *Proc. 10th Symposium on Operating Systems Principles*, pp. 15-24, December 1985.

[7] K.K. Ramakrishnan, P. Biswas and R. Karedla, "Analysis of File I/O Traces in Commercial Computing Environments", *Proc. 1992 ACM SIGMETRICS and PERFORMANCE '92 Intl. Conf. on Measurement and Modeling of Computer Systems*, pp. 78-90, June 1992.

[8] D. Roselli, J. Lorch, and T. Anderson, "A Comparison of File System Workloads", *Proc. 2000 USENIX Annual Technical Conference*, pp. 41-54, June 2000.

[9] C. Ruemmler and J. Wilkes, "UNIX Disk Access Patterns", *Proc. Winter '93 USENIX Conference*, pp. 405-420, January 1993.

[10] M.E. Gomez and V. Santonja, "Self-similarity in I/O Workloads: Analysis and Modeling", *Proc. First Workshop on Workload Characterization* (held in conjunction with the 31st annual ACM/IEEE International Symposium on Microarchitecture), pp. 97-104, November 1998.

[11] K. Keeton and D. Patterson, "Towards a Simplified Database Workload for Computer Architecture Evaluations," *Proc. Second Workshop on Workload Characterization*, October 1999.

[12] L. John, P. Vasudevan and J. Sabarinathan, "Workload Characterization: Motivation, Goals and Methodolgy", *Workload Characterization : Methodology and Case Studies*, IEEE Computer Society, edited by L. John and A. M. G. Maynard, 1999.

[13] R. P. Goldberg. "Survey of Virtual Machine Research", *IEEE Computer*, 7(6), June 1974.

[14] E. Bugnion, S. Devine, K. Govil and M. Rosenblum, "Disco: Running Commodity Operating Systems on Scalable Multiprocessors", *ACM Transactions on Computer Systems*, 15(4), Nov 1997.

[15] B. Ford, M. Hibler, J. Lepreau, P. Tullman, G. Back and S. Clawson. "Microkernels Meet Recursive Virtual Machines", *Proc. Symposium on Operating System Design and Implementation*, October 1996.

[16] J. Sugerman, G.Venkitachalam and B.-H. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor", *Proc. of Usenix Annual Technical Conference*, pp. 1-14, June 2001.

[17] C. A. Waldspurger, "Memory Resource Management in VMware ESX Server", *Proc. Fifth Symposium on Operating System Design and Implementation (OSDI '02)*, December 2002.