VIRTUALIZATION: CONCEPTS, APPLICATIONS, AND PERFORMANCE MODELING

Daniel A. Menascé*
Dept. of Computer Science
George Mason University
Fairfax, VA 22030, USA
menasce@cs.gmu.edu

Abstract

Virtualization was invented more than thirty years ago to allow large expensive mainframes to be easily shared among different application environments. As hardware prices went down, the need for virtualization faded away. More recently, virtualization at all levels (system, storage, and network) became important again as a way to improve system security, reliability and availability, reduce costs, and provide greater flexibility. This paper explains the basics of system virtualization and addresses performance issues related to modeling virtualized systems using analytic performance models. A case study on server consolidation is used to illustrate the points.

1 Introduction

Exactly thirty years ago I was a Ph.D. student at UCLA taking an operating systems course. The term project assigned to the class consisted of implementing a Virtual Machine Monitor (VMM) for a simulated architecture. At that time, system virtualization was a hot topic of academic and industrial research [5, 7], which saw the light of day in products such as IBM's VM/370 [3].

System virtualization adds a hardware abstraction layer, called the Virtual Machine Monitor (VMM), on top of the bare hardware. This layer provides an interface that is functionally equivalent to the actual hardware to a number of *virtual machines*. These virtual machines may then run regular operating systems, which would normally run directly on top of the actual hardware. This characterization is a bit oversimplified. There are various virtualization techniques as well as requirements for architectures to be virtualizable [7]. The main motivation for virtualization in the early 70's was to increase the level of sharing and utilization of expensive computing resources such as the mainframes. The 80's saw a decrease in hardware costs that caused a significant portion of the computing needs of an organization to be moved away from large centralized mainframes to a collection of departmental minicomputers. The main motivation for virtualization disappeared and with it their commercial embodiments.

The advent of microcomputers in the late 80's and their widespread adoption during the 90's along with ubiquitous networking brought the distribution of computing to new grounds. Large number of client machines connected to numerous servers of various types gave rise to new computational paradigms such as client-server and peer-to-peer systems. These new environments brought with them several challenges and problems including reliability, security, increased administration cost and complexity, increased floor space, power consumption, and thermal dissipation requirements.

The recent rebirth of the use of virtualization techniques in commodity, inexpensive servers and client machines is poised to address these problems [4, 9] in a very elegant way. This paper describes the basic concepts in virtualization and explains how virtualization can be used to support server consolidation efforts. A quantitative analysis of virtualization performance and some numerical examples are provided to illustrate the various issues.

2 Basic Concepts in Virtualization

Consider a production environment consisting of batch jobs and online transactions that run on top of a Transaction Processing Monitor (TPM). Developers need to test new features of the system. A typical approach is to use one machine for the production environment and another, typically smaller, for development and testing. Virtualization allows you to run the two environments on the same machine (see Fig. 1) in such a way that these two environments are completely isolated from one another. As the figure shows, the production environment runs on top of op-

*This work was supported in part by the National Geospatial-Intelligence Agency (NGA) contract NMA501-03-1-2033.

erating system OS1 and the test environment runs on top of operating system OS2. Both operating systems run on top of the Virtual Machine Monitor (VMM). The VMM virtualizes all resources (e.g., processors, memory, secondary storage, networks) and allocates them to the various virtual machines that run on top of the VMM.

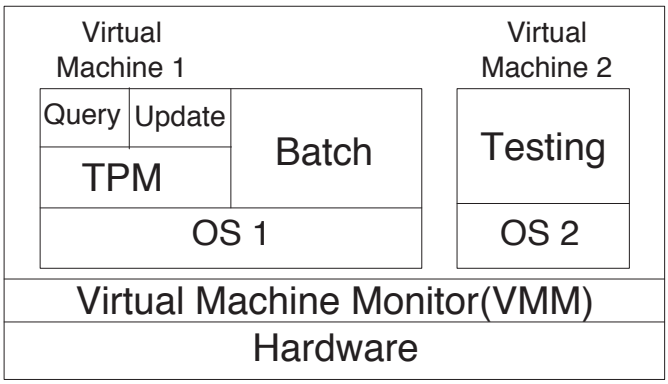

Figure 1: Virtualization Basic Concepts.

I describe now in more detail what is known as virtualization through *direct execution* [9]. Let us start with some basic facts of computer architecture (see [6] for more details on computer architecture). The instruction set is generally divided into (at least) two categories: non-privileged and priviliged instructions. The former instructions do not change the allocation (and in some cases the state) of any of the resources of the machine that are shared among the various executing processes. Examples of such resources include processors, main memory, secondary storage devices, network connections, timer, and special purpose registers such as the program counter and mode bit. Privileged instructions include all those that are used to change the allocation or state of a machine's shared resources. Examples of such instructions include: halt the machine, set the timer, set the program counter, change the value of memory allocation registers, set the mode bit, and I/O-related instructions.

A machine operates in two modes: user and supervisor. In supervisor mode, the entire instruction set can be executed. This is the mode in which the operating system runs. In user mode, only non-privileged instructions can be executed. The operating system sets the mode bit to user before giving control of the CPU back to a user program. If a privileged instruction is executed in user mode, an interrupt is generated and control is passed to an interrupt handling routine, which is part of the operating system. Most architectures have more than two levels of privilege. For example, the x86 architectures has four levels, called *rings*, numbered from 0 to 3. Ring 0 has the highest privilege and this is the level at which the operating system runs in non-virtualized environments.

In a virtual machine environment, the VMM runs in supervisor mode and controls access to the resources shared by all virtual machines and the virtual machines run in user mode. The VMM schedules the virtual machines, in a manner similar to how an operating system schedules processes, and allocates processor cycles to them. Consider any given virtual machine running on top of the VMM. Under the direct execution technique, any non-privileged instruction executed by a virtual machine is executed directly by the hardware. However, if the virtual machine executes a privileged instruction because the OS of that virtual machine (called *guest OS*)is executing, an interrupt is generated (because the virtual machine runs in user mode). The VMM then has to emulate the execution of the privileged instruction. For example, if the instruction executed by a virtual machine is a halt instruction, the issuing virtual machine is stopped, but all other virtual machines continue to run. If a virtual machine issues an I/O operation, the VMM has to map that I/O into an operation to be carried out at one of the real devices that are used to support the virtual devices seen by the virtual machines. This is illustrated in Fig. 2, that shows that virtual disks VD-A and VD-B are mapped to physical disk A and that virtual disks VD-C, VD-D, and VD-E are mapped to physical disk B. The VMM keeps track of the mapping information as part of a virtual machine's state.

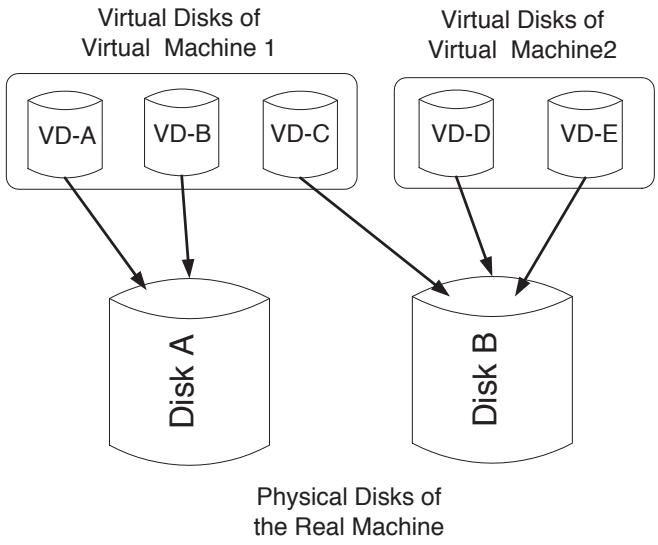

Figure 2: Mapping of Virtual to Physical Disks.

The slowdown, S_v, of running a virtual machine in the direct execution approach is a function of the number of privileged instructions executed by the guest OS and of the number of instructions needed to emulate a privileged instruction. Let f_p be the fraction of privileged instructions executed by a a virtual machine and let N_e be the average number of instructions required by the VMM to emulate a

privileged instruction. Then,

$$S_v = f_p \times N_e + (1 - f_p) = f_p(N_e - 1) + 1. \quad (1)$$

For example, if $f_p = 0.1\%$ and $N_e = 500$ the slowdown is $S_v = 0.001 \times 499 + 1 = 1.5$. This means that applications running on virtual machines will see their CPU time increased by 50%. That does not mean that the application will run 50% times slower. The application slowdown depends on the fraction of the application's time spent in I/O vs. CPU.

3 Advantages of Virtualization

There are several advantages to virtualization across several dimensions:

- *Security:* by compartmentalizing environments with different security requirements in different virtual machines one can select the guest operating system and tools that are more appropriate for each environment. For example, we may want to run the Apache web server on top of a Linux guest operating system and a backend MS SQL server on top of a guest Windows XP operating system, all in the same physical platform. A security attack on one virtual machine does not compromise the others because of their isolation.
- *Reliability and availability:* A software failure in a virtual machine does not affect other virtual machines.
- *Cost:* It is possible to achieve cost reductions by consolidation smaller servers into more powerful servers. Cost reductions stem from hardware cost reductions (economies of scale seen in faster servers), operations cost reductions in terms of personnel, floor space, and software licenses. VMware cites overall cost reductions ranging from 29 to 64% [11].
- *Adaptability to Workload Variations:* Changes in workload intensity levels can be easily taken care of by shifting resources and priority allocations among virtual machines. Autonomic computing-based resource allocation techniques, such as the ones in [2], can be used to dynamically move processors from one virtual machine to another.
- *Load Balancing:* Since the software state of an entire virtual machine is completely encapsulated by the VMM, it is relatively easy to migrate virtual machines to other platforms in order to improve performance through better load balancing [10].
- *Legacy Applications:* Even if an organization decides to migrate to a different operating system, it is possible to continue to run legacy applications on the old OS running as a guest OS within a VM. This reduces the migration cost.

4 Performance Modeling of Virtualized Environments

I describe in this section how analytic queuing models can be used to model virtualized environments. First, let us briefly review the type of analytic models and the notation used throughout the paper (for more details see [8]). An analytic queuing network model (QN) is a representation of computer systems that can be used to derive performance metrics from the model's input parameters. A QN is composed of resources (also called queues) which may be of three types: load independent (the service rate does not depend on the queue length), load dependent (the service rate is a function of the queue length), and delay resources (there is no queuing). The unit of work that flows through a QN is generally called a $customer$, and represents transactions, requests, jobs, processes, etc.

A QN model may have more than one class of customers. Multiple class models are used to represent situations in which customers may have different service times at the various resources or different workload intensity levels. The classes of a QN model may be open or closed. The workload intensity of an open class r is specified as an average arrival rate λ_r. The workload intensity of a closed class r is specified as the concurrency level N_r of that class, i.e., the number of concurrent customers of that class in the system. QN models with both open and closed classes are called mixed QN models.

The parameters of a QN are divided into two categories: $service\ demand$, $D_{i,r}$, of customers of class r at resource i and workload intensity levels per class (arrival rates or concurrency levels). The service demand $D_{i,r}$ is the total service time of customers of class r at resource i. According to the Service Demand Law [8], $D_{i,r} = U_{i,r}/X_{0,r}$ where $U_{i,r}$ is the utilization of resource i by class r customers and $X_{0,r}$ is the throughput of class r customers.

We illustrate the process of modeling a virtualized environment using the example of Fig. 1. First, let us discuss the issue of data collection in virtualized environments. Consider that the system of Fig. 1 was monitored during 30 minutes and during that period measurements were taken at various levels, i.e., by the VMM, by OS 1, by OS 2, and by the Performance Monitor of the TPM. Table 1 shows the values of the various metrics obtained at the various layers of the virtualized environment. The notation for a metric is in the form $M_{i,r}^{\text{layer}}$ where M is a metric which can be T for total time, U for utilization, and N for number of IOs or transactions. The superscript indicates where the measurement is taken, and i and r represent a resource and customer class, respectively. For example, $T_{\text{cpu,query}}^{\text{tpm}}$ stands for the total CPU time due to all query transactions as measured by the Performance Monitor of the TPM.

The system has a single CPU and two physical disks, D1 and D2. Virtual machine 1 has three virtual disks, vd1, vd2, and vd3, which are mapped to physical disk D1. Virtual

Metric	Value
$T^{\text{vmm}}_{\text{cpu,vm1}}$	420 sec
$T^{\text{vmm}}_{\text{cpu,vm2}}$	220 sec
$U^{\text{vmm}}_{\text{cpu}}$	0.40
$U^{\text{vmm}}_{\text{D1}}$	0.35
$U^{\text{vmm}}_{\text{D2}}$	0.45
$N^{\text{tpm}}_{\text{query}}$	5400 transactions
$N^{\text{tpm}}_{\text{updt}}$	1200 transactions
$T^{\text{tpm}}_{\text{cpu,query}}$	110 sec
$T^{\text{tpm}}_{\text{cpu,updt}}$	150 sec
$N^{\text{tpm}}_{\text{vd1,query}}$	6300 IOs
$N^{\text{tpm}}_{\text{vd2,query}}$	8640 IOs
$N^{\text{tpm}}_{\text{vd1,updt}}$	9450 IOs
$N^{\text{tpm}}_{\text{vd2,updt}}$	9720 IOs
$N^{\text{os1}}_{\text{vd3,batch}}$	18390 IOs
$T^{\text{os1}}_{\text{cpu,batch}}$	100 sec
$T^{\text{os1}}_{\text{cpu,tpm}}$	290 sec
$N^{\text{os2}}_{\text{vd4,testing}}$	5400 IOs

Table 1: Measurement for the example of Fig. 1.

machine 2 has a single virtual disk, vd4, mapped to physical disk D2. The problem now is how to obtain the input parameters to a multiclass QN model where the classes are Query (Q), Update (U), Batch (B), and Testing (T), and the resource are CPU, D1, and D2. We illustrate the procedure through a few examples starting with the CPU. Let us obtain $U_{\text{cpu,q}}$, the utilization of the CPU due to query transactions. From Table 1, we know that the total CPU utilization as measured by the VMM, $U^{\text{vmm}}_{\text{cpu}}$, is 0.4. In order to find which portion of that utilization is due to virtual machine 1 (vm1), we use the total CPU time measured by the VMM as an apportionment factor. Thus, the CPU utilization of vm1 is given by

$$U^{\text{vmm}}_{\text{cpu}} \times \frac{T^{\text{vmm}}_{\text{cpu,vm1}}}{T^{\text{vmm}}_{\text{cpu,vm1}} + T^{\text{vmm}}_{\text{cpu,vm2}}} = 0.4 \times \frac{420}{420+220} = 0.2625. \quad (2)$$

Note that if we tried to compute the CPU utilization of vm1 as $T^{\text{vmm}}_{\text{cpu,vm1}}/1800 = 420/1800$ we would get 0.2333 and not the 0.2625 obtained in Eq. (2). The difference, i.e., 2.92% is due to the VMM overhead to manage the workload of vm1.

The next step is to determine what fraction of vm1's CPU utilization is due to the TPM. We use the CPU times measured by OS1 as an apportionment factor. Hence, the CPU utilization of the TPM is given by

$$0.2625 \times \frac{T^{\text{os1}}_{\text{cpu,tpm}}}{T^{\text{os1}}_{\text{cpu,tpm}} + T^{\text{os1}}_{\text{cpu,batch}}} = 0.2625 \times \frac{290}{290+100} = 0.1952. \quad (3)$$

In order to obtain the CPU utilization due to query transactions, we need to use CPU times measured by the TPM as follows:

$$0.1952 \times \frac{T^{\text{tpm}}_{\text{cpu,query}}}{T^{\text{tpm}}_{\text{cpu,query}} + T^{\text{tpm}}_{\text{cpu,updt}}} = 0.1952 \times \frac{110}{110+150} = 0.0826 \quad (4)$$

So, putting it all together, the CPU utilization of query transactions is given by

$$U_{\text{cpu,q}} = U^{\text{vmm}}_{\text{cpu}} \times \frac{T^{\text{vmm}}_{\text{cpu,vm1}}}{T^{\text{vmm}}_{\text{cpu,vm1}} + T^{\text{vmm}}_{\text{cpu,vm2}}} \times \frac{T^{\text{os1}}_{\text{cpu,tpm}}}{T^{\text{os1}}_{\text{cpu,tpm}} + T^{\text{os1}}_{\text{cpu,batch}}} \times \frac{T^{\text{tpm}}_{\text{cpu,query}}}{T^{\text{tpm}}_{\text{cpu,query}} + T^{\text{tpm}}_{\text{cpu,updt}}}. \quad (5)$$

The throughput, $X_{0,\text{q}}$, of query transactions is

$$X_{0,\text{q}} = \frac{N^{\text{tpm}}_{\text{query}}}{1800} = 5400/1800 = 3 \text{ tps} \quad (6)$$

and the service demand of query transactions at the CPU is given by $D_{\text{cpu,q}} = U_{\text{cpu,q}}/X_{0,\text{q}} = 0.00826/3 = 0.0275$ sec.

We now give an example of the computation of the utilization, $U_{\text{D1,q}}$, of disk D1 by query transactions. The procedure used below is very similar to the one used for the CPU except that instead of using total CPU times measured at each layer as an apportionment factor, we use the total number of IOs. The total number of IOs, N_{D1}, on physical disk D1 is given by the sum of IOs on all virtual disks mapped to D1. Thus,

$$\begin{aligned} N_{\text{D1}} &= N^{\text{tpm}}_{\text{vd1,query}} + N^{\text{tpm}}_{\text{vd2,query}} + N^{\text{tpm}}_{\text{vd1,updt}} + N^{\text{tpm}}_{\text{vd2,updt}} + N^{\text{os1}}_{\text{vd3,batch}} \\ &= 6300 + 8640 + 9450 + 9720 + 18390 \\ &= 52500 \text{ IOs}. \end{aligned} \quad (7)$$

Hence,

$$U_{\text{D1,q}} = U^{\text{vmm}}_{\text{D1}} \times \frac{N^{\text{tpm}}_{\text{vd1,query}} + N^{\text{tpm}}_{\text{vd2,query}}}{N_{\text{D1}}} = 0.35 \times \frac{6300+8640}{52500} = 0.0996. \quad (8)$$

The service demand, $D_{\text{D1,q}}$, of query transactions at disk D1 is then given by $D_{\text{D1,q}} = U_{\text{D1,q}}/X_{0,\text{q}} = 0.0996/3 = 0.0332$ sec. The procedure above assumes that the service time per IO is the same for all classes. If this assumption is not true, then an appropriate apportionment factor is the

ratio of the products of the total number of IOs multiplied by the service time per IO.

Table 2 shows the results obtained by following the procedures outlined above for all classes and all resources.

Resource	Query	Updt	Batch	Test
CPU	0.0275	0.1689	0.0673	0.1375
D1	0.0332	0.1917	0.1226	0.0000
D2	0.0000	0.0000	0.0000	0.4500

Table 2: Service demands (in sec) for the example of Fig. 1.

Once the model parameters are computed, one can use well-known QN model solution techniques to obtain the performance metrics of interest [8].

5 Server Consolidation Example

As indicated previously, virtualization may be used for server consolidation. Figure 3(a) shows the typical architecture of an e-commerce site with separate machines for Web servers, applications servers, and database servers. Figure 3(b) shows these servers being consolidated on a single, albeit more powerful, server.

We use QN models to compare these two alternatives. In order to do that, we have to map the input parameters used to model the situation in Fig. 3(a) to that of Fig. 3(b). The service demand at the CPU for the consolidated environment, $D^{\text{cons}}_{\text{CPU},r}$ is given by

$$D^{\text{cons}}_{\text{CPU},r} = \sum_{s=1}^{S} D_{\text{CPU}_s,r} \times S_v/C_s \qquad (9)$$

where CPU_s stands for the CPU of the s-th individual server being consolidated, S_v is the slowdown due to virtualization already defined before, and C_s is the speedup of the consolidated server with respect to individual server s.

We applied these conversions to a situation in which the original site was an online auction site with three separate servers. Each one had a CPU and one disk. We consolidated the three servers into a single server with two disks. In the virtual environment, the Web server, application server, and database server run on different virtual machines. The virtual disks used by the virtual machines for the Web server and application server are mapped to one of the physical disks of the consolidated server. The virtual disks of the virtual machine that runs the database server is mapped to the second physical disk of the consolidated server.

Figure 4 shows the ratio of the response time for the View Bid class of requests under the consolidated scenario over the response time obtained under the individual server scenario. The x-axis is the consolidated server speedup, C_s. We assume in that figure that the virtualization slowdown is $S_v = 1.5$. As the figure indicates, if $C_s = 1$, the response time of the consolidated server is 3.6 times higher than in the original case. For a speedup of $C_s = 2.5$, the consolidated server has a response time that is only 4% higher than in the original case. For a speedup of $C_s = 3.0$, the consolidated server has a slightly smaller response time than the three original servers. The consolidated case may still offer reduced costs and improved manageability as discussed before.

It should also be pointed out that the performance gain of the consolidated environment decreases very fast at the beginning and slower at the end as the processor no longer is the bottleneck.

6 Concluding Remarks

Virtualization may bring several advantages to the design of modern computer systems including better security, higher reliability and availability, reduced costs, better adaptability to workload variations, easier migration of virtual machines among physical machines, and easy coexistence of legacy applications. Many vendors including Sun, IBM, and Intel have already announced or already have virtualization solutions. Intel has just announced a new architecture, called Intel Virtualization Technology, that provides hardware support for virtualization allowing the virtual machine monitor to run at a protection level below ring 0. Sun has introduced the concept of *zones* in Solaris 10, which allows for many Solaris 10 instances to coexist on the same machine (www.sun.com/bigadmin/content/zones/). IBM provides Logical Partitioning (LPAR) technology on its p, i, and z platforms. This technology was originally developed for its mainframes but is now available on its midrange servers (www-03.ibm.com/servers/eserver/iseries/lpar/ and www-03.ibm.com/servers/eserver/pseries/lpar/)

It is important to briefly discuss two major directions for virtualization that can be encountered on the market. One is called *full virtualization* and the other *paravirtualization*. In the former case, the Virtual Machine Monitor provides an identical abstraction of the underlying hardware to the virtual machines. However, not all architectures are *virtualizable* [7]. The x86 architecture is such an example [9]. Paravirtualization can then be used to overcome these situations by providing an "almost" identical abstraction of the underlying machine to the virtual machines. This abstraction implements some new virtual instructions so as to make the machine virtualizable. The drawback is that the guest operating system has to be modified to use these instructions while in the full virtualization case this is not required. Paravirtualization provides better performance than full virtualization since the guest operating systems are aware that they running on a VM and therefore can be optimized for that type of environment. Examples of virtual machine monitors that use paravirtualization include the open source Xen [1] and Denali [12]. An example

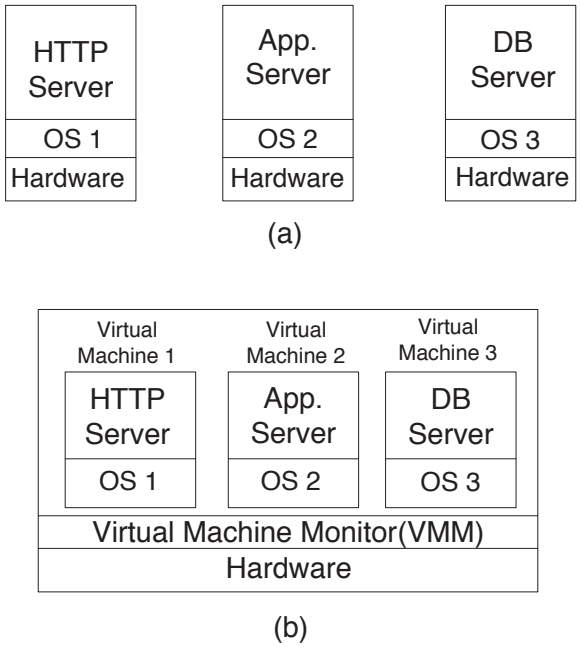

Figure 3: Server consolidation scenario.

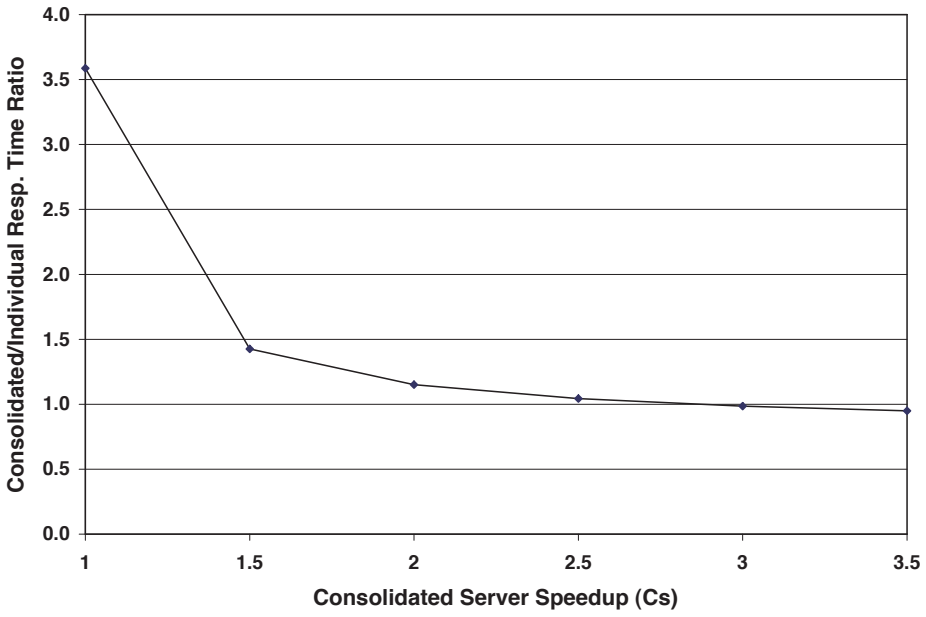

Figure 4: Performance comparison for server consolidation scenario.

of a virtual machine monitor that uses full virtualization is VMware (www.vmware.com).

This paper presented some of the issues involved in using well-known queuing network models for virtualized environments. A case study on server consolidation was used to illustrate the point.

References

[1] P. Barham et al., "Xen and the Art of Virtualization," *Proc. ACM Symposium on Operating Systems*, Bolton Landing, NY, October 19–22, 2003.

[2] M.N. Bennani and D.A. Menascé, "Resource Allocation for Autonomic Data Centers Using Analytic Performance Models," *Proc. 2005 IEEE International Conference on Autonomic Computing*, Seattle, WA, June 13-16, 2005

[3] R.J. Creasy, "The Origin of the VM/370 Time-Sharing System," *IBM J. Research and Development*, Sept. 1981, pp. 483–490.

[4] R. Figueiredo, P.A. Dinda, and J. Fortes, "Resource Virtualization Renaissance," *IEEE Internet Computing*, May 2005, Vol. 38, No. 5.

[5] R.P. Goldberg, "Survey of Virtual Machine Research," *IEEE Computer*, June 1974, pp.34–45.

[6] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd edition, Morgan Kaufman, 2003.

[7] G.J. Popek and R.P. Goldberg, "Formal Requirements for Virtualizable Third-Generation Architectures," *Comm. ACM*, July 1974, pp. 412–421.

[8] D.A. Menascé, V.A.F. Almeida, and L.W. Dowdy, *Performance by Design: Computer Capacity Planning by Example*, Prentice Hall, Upper Saddle River, 2004.

[9] M. Rosenblum and T. Garfinkel, "Virtual Machine Monitors: Current Technology and Future Trends," *IEEE Internet Computing*, May 2005, Vol. 38, No. 5.

[10] R. Uhlig et. al., "Intel Virtualization Technology," *IEEE Internet Computing*, May 2005, Vol. 38, No. 5.

[11] VMware, "Consolidating Mission Critical Servers," www.vmware.com/solutions/consolidation/mission_critical.html

[12] A. Whitaker, R.S. Cox, M. Shaw, and S.D. Gribble, "Rethinking the Design of Virtual Machine Monitors," *IEEE Internet Computing*, May 2005, Vol. 38, No. 5.