A framework to evaluate and predict performances in virtual machines environment*

Deshi Ye, Qinming He, Hua Chen, Jianhua Che
College of Computer Science, Zhejiang University,
Hangzhou 310027, China.
Email:{*yedeshi,hqm,solaris*}@*zju.edu.cn*.

Abstract

Virtualization technology becomes more and more important in area of compute science, such as data center and server consolidation. A large number of hypervisors are available to manage the virtualization either on bare hardware or on host operating systems. One of the important task for the designer is to measure and compare the performance overhead of given virtual machines. In this paper, we provide an analytic framework for the performance analyzing either without running a system or in a runnable real system. Meanwhile, analytic performance models that are based on the queue network theory are developed to study the designs of virtual machines. At the end, a case study of the mathematical models is given to illustrate the performance evaluation.

Keywords: *Performance evaluation; virtual machines; queue networks.*

1. Introduction

In computer science, virtual machines regain the interests to researches as well as the industries. The main advantages of the virtual machines are that multiple guest operating systems can co-exist on the same computer and it is in strong isolation from each other. Another important benefit of the virtual machines is to provide an instruction set architecture (ISA) that could be different from that of the real machine. Applications of virtual machines arise widely in computer science, such as in embedded systems, which support a real-time operating system at the same time as a high-level OS.

In virtual machines(VMs) environment, the software layer which provides the virtualization is called a *virtual machine monitor* (VMM) or a *hypervisor*. Generally, there are two kinds of hypervisors, one runs on bare hardware, and another runs on the top of a host OS. Sometimes we also call the multiple VMs as *guest operating systems*. The guest OSes do not have to be all the same, they could be different OSes or different version of an OS on the same computer. These characteristics of virtual machines are very useful for the server consolidations.

To the best of our knowledge, the main types of virtualization technologies are full-virtualization [16], para-virtualization [2], pre-virtualization [10], hardware-virtualization [17]. In full virtualization, the virtual hardware is identical to the actual physical hardware. This allows the guest OSes do not need to be modified. VMware uses this technology. In para-virtualization, the guest OSes is modified to avoid some performance overhead. In this technology, the low-level platform API is replayed by a high-level API that requires far fewer mode switches when running a de-privileged guest. Typical example of para-virtualization is Xen. In [10], L4Ka uses the technology called pre-virtualization which combines the performance of para-virtualization with the modularity of traditional virtualization. In hardware-virtualization, Intel and AMD have independently developed virtualization extensions to the x86 architecture.

No matter what technologies we taking for the virtual machines, one task is to quantify and predict the performance of the system. One intuition is to monitor the resource usage of different VMs and hypervisor itself. However, it will be impossible to do the monitoring in case of the early stage in designing a new hypervisor. Consequently, designing an analytic performance model to predict the performance is of great interests to the designer.

In this paper, based on the queueing network models, we provide a framework to analyze the performance of virtual machines system. In our framework the virtual machines either do not run at all or just monitor the virtual machines instead of the hypervisor. Finally, a case study is given to illustrate how the framework works.

*Research was supported by the State Key Development Program for Basic Research of China ("973 project", No. 2007CB310900).

Performance metrics To evaluate a system's performance, we need to determine which performance metrics we would take. In this paper, we distinguish the performance metrics into two categories: *absolute metrics* and *relative metrics*. The traditional metrics like system throughput, resource utilization, response time are belonging to the absolute case. The standard metric used in virtualization environment called *VM/native* is a relative metric, which is the ratio of some specific absolute metric obtained in virtual machine divided by the same metric obtained in the native physical machine that without virtualization, but with the same configuration as the virtual machine.

The relative performance metric easily gives the *performance overhead* of a system. It is worthy to note that the absolute performance metric is a foundation of the relative performance metric. We can get the relative performance immediately upon the absolute performance is available if we also have a physical computer with the same configuration as the virtual machine. However, it is not easy that we can always get the relative ratio because it could be hard and not flexible to have such a physical computer. Hence, we only study absolute performance metrics in the following.

Related work Using the monitoring method to evaluate the performance of a virtualization system has been extensively studied [2, 8, 14]. For hypervisor Xen, xm [2] and Xenoprof [14] are powerful tools to realize system profiling. Benevenuto et al. [4] developed an application called Xencpu to measure CPU busy time on Xen. Adams and Agesen [1] investigated the x86 virtualization. Two technical reports [18, 19] both compared the performance of products of Xen and VMware (but with different versions of Xen) by some well-known benchmarks. Lu et al. [11] used the software BMC Performance Assurance for Servers to study the performance of Xen. All the above works use the relative performance metric *VM/native*, and all the systems are real systems.

Concerning on the analytic performance, it was Menascé [12] the first one to derive an analytic performance model for the general virtualized systems. He described analytic queueing models that can be used to model virtualized environments. Bolker and Ding [6] discussed the analytic queuing models for the virtualized system. They focused on the models on processor utilization and finally tested their model to study the VMware by a benchmark. Based on Xen's architecture, Benevenuto et al. [4] described performance models for predicting performance and designed a tool Xencpu to measure CPU time on Xen. A simple bound analysis was also given in that paper. In common, the hypervisor must be monitored in all above models.

Queue networks The basic entities in queueing network models are *service centers* (or resource center), which represent system resources, and *customers*, which represent users or jobs or transactions. We consider the multiple class models [9]. Multiple class model consists of total C classes. The workload intensity of each class c is characterized by λ_c, N_c or (N_c and Z_c), where λ_c is the arrival rate of class c, N_c is the population size of class c and Z_c is the think time of class c. The service time $S_{c,k}$ of class c at service center k is also an input of the model. The total number of class c visits the service center k is defined to be $V_{c,k}$. The service demand $D_{c,k}$ of a customer of class c at center k, is the total amount of time the customer requires in service at that center. Hence we have $D_{c,k} = V_{c,k} \cdot S_{c,k}$.

Usually, the performance metrics are throughput, response time, and average number in system. But we distinguish the outputs of a queue network by system measures or service center measures, or per-class and aggregate measures. For the detailed description of queueing model, we recommend to read the book [9].

Perhaps operational laws are the most famous queueing network theory result. For example, the utilization law of the open model [9] is as follows. Define $U_{c,k}$ to be the utilization of class c on service center k, we have

$$U_{c,k} = \lambda_c \cdot D_{c,k}. \tag{1}$$

From the utilization law, we have the processing capacity constraint $\max_k \sum_{c=1}^{C} U_{c,k} \leq 1$. One can obtain other desired performance metrics from the queue network models [9, 13].

The remainder of this paper is organized as follows: Section 2 gives the framework of analytic performance models. Section 3 provides a case study of above analytic performance model. Conclusions and future works are given in Section 4.

2. Analytic Framework

In this section we give a framework to analyze the performance of general virtual machines environment. The analytic queue network results are sometimes criticized for inaccurate performance evaluation and then it is invaluable during the development process. However, without analytic results, it is too cost for the designers to test every new design proposal using all kinds of experiments. Furthermore, the designer cannot get the performance guarantees for any new design. Consequently, it is important for the designer to get some analytic results though there is a gap between practice and theory.

The virtualized system consists total n virtual machines above the VMM (or hypervisor). Each virtual machine has its own operating system. The detail of the framework is

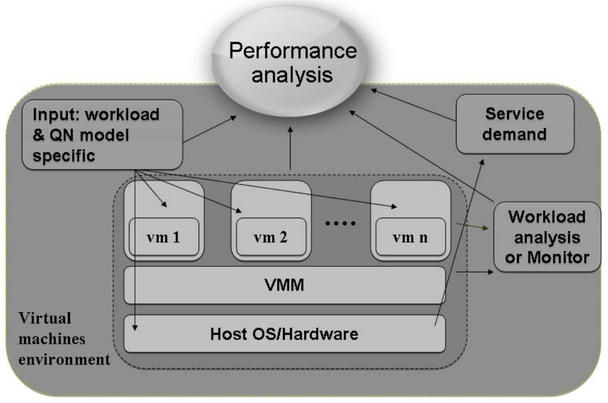
Performance analysis
Input: workload & QN model specific
Service demand
vm 1
vm 2
....
vm n
Workload analysis or Monitor
VMM
Host OS/Hardware
Virtual machines environment

Figure 1. Framework of analytic performance model

illustrated in Fig. 1. In the following, we will adopt the queue network theory to analyze the performance of the given framework. The framework consists of four parts (or components), *input, workload analysis or monitor, service demand*, and *performance analysis*. In the *input* part, we must specify the queue network model. At first we need to select the class type: open, closed and mixed product form [3]. Then the workload in each virtual machines are described. In the case of single class, the workload is described as the work intensity for each resource. In the case of multiple classes, a matrix of work intensive for every resources are required.

In the *service demand* parts, the service demands of the input workload are requested, i.e. the service time of a class and the total number of visits at that resource center. Note that, in this part, the service demands of physical hardware devices for each class of workload are request to be known. In case the service demand is not known before hand, we can use a general approach introduced by Zhang et al. [21] to estimate it.

In the *workload analysis* or *monitor* part, one of them is selected for the final part *performance analysis*. In the *workload analysis* part, we suppose that the final output is a sequence of instructions. In the *monitor* part, we suppose to run a monitoring program to record down the utilizations of virtual devices in all the virtual machines. Unlike the *workload analysis* part, the virtual system in the monitor part must be runnable.

Finally, all the information from the first three parts pass to the component *performance analysis*. The performance metrics on each class and each device could be utilization, throughput, queue length, response time. We also concern on aggregate system's performance of each virtual machine, there are system throughput, system response time, average number of customers in the system.

Now we show in detail how the framework can provide the performance evaluation in the virtual machines environment.

2.1. Virtual service demands

To obtain the performance of a system, by the operational laws in queue networks [9], the key step is to get the workload intensity and the service demand. Since the workload intensity will be the same in classical computer system and virtual machines, we only need to compute the service demands in virtual machines. In the following, we compute the service demands based on the component of workload analysis or monitor.

2.1.1 Workload analysis

Firstly we define the following quantities:

I^{c,V_k^i}: total number of instructions that are required by class c for virtual resource center k in VM_i.

PI_j^{c,V_k^i}: total number of privileged instructions j by class c for the virtual resource center k in virtual machine VM_i.

EPI_j: average number of instructions required to emulate the privileged instruction j.

Many seminal papers addressed the workload analysis, see [7] for a survey. There were some tools for the workload characterization and analysis [5, 20]. By setting the smallest level of a workload to be an instruction, the final output of workload analysis is a sequence of instructions. That is to say, I^{c,V_k^i} is known as the output of the workload analysis. For a specific hypervisor or a VMM, we are also assumed to know whether an instruction is a privileged instruction, and if it is a privileged instruction, we can get the number of instructions to emulate it. Whereupon, the values of PI_j^{c,V_k^i} and EPI_j become available after workload analyzing. For example, it is estimated that the QEMU [15] without the module roughly need 5 or 10 native instructions to emulate a privileged instructions.

In [12] *slowdown* factor (S_v) of an application in a virtual machine is $S_v = f_p \cdot N_e + (1 - f_p)$, where f_p is the fraction of privileged instructions executed by a virtual machine and N_e is the average number of instructions required by the hypervisor to emulate a privileged instruction. Similar as the idea of [12], we define the *overhead* of an application of class c in virtual resource k of VM_i as follows:

$$OH_{c,k}^{VM_i} = \frac{\sum_j PI_j^{c,V_k^i} \cdot EPI_j + I^{c,V_k^i} - \sum_j PI_j^{c,V_k^i}}{I^{c,V_k^i}}. \quad (2)$$

Denote $S_{c,k}^{VM_i}$ to be the service time of virtual resource k in VM_i for class c. Thus, we define

$$S_{c,k}^{VM_i} = OH_{c,k}^{VM_i} \cdot S_{c,f(V_k^i)}, \tag{3}$$

where $S_{c,f(V_k^i)}$ is the service time of class c in the physical resource center $f(V_k^i)$. Let $V_{c,k}^{VM_i}$ be the average number of visits at that resource k for each interaction of class c in the virtual machine VM_i. Let $D_{c,k}^{VM_i}$ be the total service demand that a customer of class c spends in service at resource center k during one interaction with the system, i.e. it's complete execution. Denoted by B_{WA} the total time of workload analyzing in the workload analysis component. Then B_{WA} could be regarded as a preliminary processing time of each class of customer in any virtual machine. From the service demand component, for any c and k, we know $S_{c,k}$, hence $D_{c,k}^{VM_i}$ can be computed as follows:

$$D_{c,k}^{VM_i} = V_{c,k}^{VM_i} \cdot S_{c,k}^{VM_i} + B_{WA}. \tag{4}$$

2.1.2 Monitor model

In case we don't know any information about the VMM or the hypervisor, i.e., the information about privileged instructions are unknown to us. In this situation, we need the assumption that the virtual machines environment are executable. Consequently, we assume that one monitoring programming can run in all the virtual machines. It is worthy to notice that there exist lots of small tools for monitoring the performance of all kinds of resource centers. The quantities we obtained from the monitoring programming is the utilization of each virtual resource center, which implies that $U_{c,k}^{VM_i}$ is already given in our model. From the equation (1), we have the service demands as follows:

$$D_{c,k}^{VM_i} = U_{c,k}^{VM_i} / \lambda_c. \tag{5}$$

2.2. Performance analysis

In the general virtualization environment, each virtual machine runs several classes of jobs, our task is to compute the performance in each virtual machine. Though jobs running in each virtual machine don't know whether the other virtual machines exist, but in the analyze model, we could assume that all the jobs consist of total C classes. If one job class runs in two different virtual machines, they are regarded as different classes.

From the other three components, many quantities are known before hand now. On the other hand, we assume that the hypervisor adopts the first come first server (FCFS) scheduling policy to assign the incoming jobs for multiple classes model. Then the performance analysis component will compute the final performance according to the queue networks [9]. For example, in open model, the utilization of virtual resource k in VM_i is $U_{c,k}^{VM_i} = \lambda_c \cdot D_{c,k}^{VM_i}$, throughput $X_c = \lambda_c$, residence time $R_{c,k} = D_{c,k}$ in the delay model. The other metrics are also easy to obtain. Whereas we get that the system overhead of virtual resource k in VM_i is

$$\frac{\sum_{c \in VM_i} U_{c,k}^{VM_i}}{\sum_{c \in VM_i} U_{c,f(V_k^i)}}.$$

Remarks: The advantage of the *workload analysis* is able to analyze the performance overheads without really running the system. The disadvantage is that it gets additional time consuming costs from workload analyzing and the workload is only an approximate accuracy of the input workload. Furthermore, the service demands of each class must be known and it might be difficult to measure such service demands. The *monitor* component can bring some convenience to compute the system performance, but it must be a real system. Compared to previous works, the *monitor* component in this paper does not need to measure the hypervisor self.

2.3. Asymptotic bounds

All above we discussed is to derive the performance analysis in a given system. The queue network theory [9] also provided bounding analysis to find the bottleneck of a given system or how to configure a system such that they can work efficiently. The bounding analysis will especially benefit the virtual machines system, because, it is easy to configure or upgrade the system of a virtual machine.

Similar as [9], the performance indices are the system throughput and the system response time, respectively. The quantities as the input of bounding analysis are the following:

- K^i, the number of service centers in the virtual machine VM_i;
- $D_{\max}^i$, the largest service demand at any single center in the virtual machine VM_i;
- D^i, the sum of the service demands at the centers in the virtual machine VM_i;
- Z^i, the average think time (if the class is of terminal type) in the virtual machine VM_i.

Since, all the above parameters are known, take the inequality from [9] and the virtual service demands from Section 2.1, the asymptotic bounds are then updated accordingly after replacing all the $D_{\max}$ with our parameter $D_{\max}^i$. For example, in open model, the throughput of a virtual machine VM_i is at most $1/D_{\max}^i$, and the response time is at least D^i.

3. A case study

A case study is presented in this section. The network topology is illustrated as Fig. 2. The system consists of two virtual machines, VM_1 and VM_2. Each virtual machine has a virtual CPU and a virtual disk. There are one physical CPU and two physical disks.

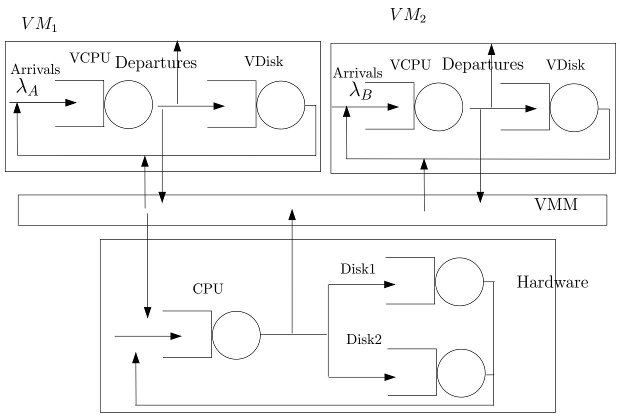

Figure 2. Case study – network topology

The virtual CPU in both virtual machines are mapping to the Physical CPU, virtual disks in VM_1 and VM_2 are mapping to the different physical disk. We adopt the following example and some data is taken from the user manual of JMT [5]. The workload in each virtual machine is an *open class*. It is the *queueing model* for the workload in service center. The arrival rate of class A in VM_1 and of class B in VM_2 are $\lambda_A = 0.15$ jobs/s, $\lambda_B = 0.32$ jobs/s, respectively. The service time and visits for service centers are given in Table 1.

Table 1. Service time and visits

		CPU	Disk1	Disk2
Class A	Service time	0.006	0.038	0.03
	Visits	101	60	0
Class B	Service time	0.014	0.062	0.08
	Visits	44	0	27

We assume that workload analyzing is timeless ($B_{WA} = 0$) and the overheads from the workload analysis component are as follows: $OH_{CPU}^{VM_1} = 1.15$, $OH_{disk}^{VM_1} = 1.2$, $OH_{CPU}^{VM_2} = 1.1$, $OH_{disk}^{VM_2} = 1.3$. Now we give the performance indices as stated in Section 2 and the optional law [9]. The utilization of service centers in VM_1 and VM_2 are as follows:

$$\begin{aligned}
U_{CPU}^{VM_1} &= \lambda_A \cdot D_{A,CPU} \cdot OH_{CPU}^{VM_1} \\
&= 0.15 * 0.006 * 101 * 1.15 = 0.1045, \\
U_{Disk}^{VM_1} &= \lambda_A \cdot D_{A,Disk} \cdot OH_{Disk}^{VM_1} \\
&= 0.15 * 0.038 * 60 * 1.2 = 0.4104, \\
U_{CPU}^{VM_2} &= \lambda_B \cdot D_{B,CPU} \cdot OH_{CPU}^{VM_2} \\
&= 0.32 * 0.014 * 44 * 1.1 = 0.2168, \\
U_{Disk}^{VM_2} &= \lambda_B \cdot D_{B,Disk} \cdot OH_{Disk}^{VM_2} \\
&= 0.32 * 0.08 * 27 * 1.3 = 0.8986.
\end{aligned}$$

The throughput of service centers in VM_1 are

$$\begin{aligned}
X_{A,CPU} &= \lambda_A \cdot V_{A,CPU} \\
&= 0.15 * 101 = 15.15 \ \ jobs/s \\
X_{A,disk} &= \lambda_A \cdot V_{A,disk} = 0.15 * 60 = 9 \ \ jobs/s.
\end{aligned}$$

Response time of service centers in VM_1 are

$$\begin{aligned}
R_{A,CPU} &= \frac{D_{A,CPU}}{1 - \sum_{j=1}^{2} U_{CPU}^{VM_j}} \\
&= \frac{0.006 * 101 * 1.15}{1 - (0.1405 + 0.2168)} = 1.03 \ \ sec. \\
R_{A,Disk} &= \frac{D_{A,Disk}}{1 - U_{Disk}^{VM_1}} = \frac{0.028 * 60 * 1.2}{1 - 0.4104} = 4.64 \ \ sec.
\end{aligned}$$

Then the aggregate response time of class A is obtained by $R_A = R_{A,CPU} + R_{A,Disk} = 5.67$ seconds. The queue length of CPU and disk in VM_1 is

$$\begin{aligned}
Q_{A,CPU} &= \frac{U_{A,CPU}}{1 - \sum_{j=1}^{2} U_{CPU}^{VM_j}} \\
&= \frac{0.1405}{1 - (0.1405 + 0.2168)} = 0.154 \ \ jobs. \\
Q_{A,Disk} &= \frac{U_{A,Disk}}{1 - U_{Disk}^{VM_1}} = \frac{0.4104}{1 - 0.4104} = 0.696 \ \ jobs.
\end{aligned}$$

4. Conclusions & Future Work

In this paper we addressed an analytic framework and performance models for the performance analysis of virtual machines system. This framework works if we know the information of privileged instructions and how to emulate a given privileged instruction of a given hypervisor. The provided framework is especially useful in the system design period or the simulation period. We also gave the analytic models if the system is a real system. In this case, we need to measure the utilizations of virtual service centers, but we do not need to know any information about privileged instructions and do not need to monitor the hypervisor itself, i.e., the performance can be computed with a monitor programming running in the virtual machines.

To give more intuition of our framework, we will do a series of experimental results by running real world benchmarks on the VMM environment. The detail of experimental results will be given in the full version of this paper.

References

[1] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, 2–13, 2006.

[2] 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. *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 164–177, 2003.

[3] F. Baskett, K.M. Chandy, R.R. Muntz, and F.G. Palacios. Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM (JACM)*, 22(2):248–260, 1975.

[4] F. Benevenuto, C. Fernandes, M. Santos, V. Almeida, J. Almeida, G. Janakiraman, and J.R. Santos. Performance Models for Virtualized Applications? *LNCS*, 4331:427–439, 2006.

[5] M. Bertoli, G. Casale, and G. Serazzi. Java Modelling Tools: an Open Source Suite for Queueing Network Modelling and Workload Analysis. *Proceedings of the International Conference on Quantitative Evaluation of Systems*, pages 119–120, 2006. Tools homepage, http://jmt.sourceforge.net.

[6] E. Bolker and Y. Ding. Virtual performance won't do: Capacity planning for virtual systems. *Proc. 31th Int. Computer Measurement Group Conf*, 1:39, 2005.

[7] M. Calzarossa and G. Serazzi. Workload characterization: a survey. *Proceedings of the IEEE*, 81(8):1136–1150, 1993.

[8] L. Cherkasova and R. Gardner. Measuring CPU overhead for I/O processing in the Xen virtual machine monitor. *Proceedings of the USENIX Annual Technical Conference*, 24–24, 2005.

[9] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc. 1984.

[10] Joshua LeVasseur, Volkmar Uhlig, Matthew Chapman, Peter Chubb, Ben Leslie, and Gernot Heiser. Pre-virtualization: soft layering for virtual machines. Technical Report 2006-15, Fakultät für Informatik, Universität Karlsruhe (TH), 2006.

[11] J. Lu, L. Makhlis, and J. Chen. Measuring and Modeling the Performance of the Xen VMM. *Proc. 32th Int. Computer Measurement Group Conf*, 2:621, 2006.

[12] D.A. Menascé. Virtualization: Concepts, applications, and performance modeling. *Proc. 31th Int. Computer Measurement Group Conf*, 407–414, 2005.

[13] D.A. Menascé and V.A.F. Almeida. *Performance by Design: Computer Capacity Planning by Example*. Prentice Hall PTR, 2004.

[14] A. Menon, J.R. Santos, Y. Turner, G.J. Janakiraman, and W. Zwaenepoel. Diagnosing Performance Overheads in the Xen Virtual Machine Environment. *Proc. of Virtual Execution Environments*, 13–23, 2005.

[15] Qemu. http://fabrice.bellard.free.fr/qemu/.

[16] L.H. Seawright and R.A. MacKinnon. VM/370 - A Study of Multiplicity and Usefulness. *IBM Systems Journal*, 18(1):4–17, 1979.

[17] R. Uhlig, G. Neiger, D. Rodgers, AL Santoni, FCM Martins, AV Anderson, SM Bennett, A. Kagi, FH Leung, and L. Smith. Intel virtualization technology. *Computer*, 38(5):48–56, 2005.

[18] I. VMware. A Performance Comparison of Hypervisors, 2007.

[19] I. XenSource. A Performance Comparison of Commercial Hypervisors, 2007.

[20] L. Zhang, Z. Liu, A. Riabov, M. Schulman, C. Xia, and F. Zhang. A Comprehensive Toolset for Workload Characterization, Performance Modeling, and Online Control. *Computer Performance Evaluation: Modelling Techniques and Tools*, 63–77, 2003.

[21] L. Zhang, C. Xia, M. Squillante, and W. Mills III. Workload service requirements analysis: a queueing network optimization approach. *Proc. of 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, 23–32, 2002.