Evaluating Approaches for Performance Prediction in Virtualized Environments

Fabian Brosig*, Fabian Gorsler‡, Nikolaus Huber* and Samuel Kounev*
Karlsruhe Institute of Technology (KIT), Germany
*Email: {nikolaus.huber, fabian.brosig, kounev}@kit.edu, ‡fabian.gorsler@student.kit.edu

***Abstract*— Performance management and performance prediction of services deployed in virtualized environments is a challenging task. On the one hand, the virtualization layer makes the estimation of performance model parameters difficult and inaccurate. On the other hand, it is difficult to model the hypervisor scheduler in a representative and practically feasible manner. In this paper, we describe how to obtain relevant parameters, such as the virtualization overhead, depending on the amount and type of available monitoring data. We adapt classical queueing-theory-based modeling techniques to make them usable for different configurations of virtualized environments. We provide answers how to include the virtualization overhead into queueing network models, and how to take the contention between different VMs into account. Finally, we evaluate our approach in representative scenarios based on the SPECjEnterprise2010 standard benchmark and XenServer 5.5, showing significant improvements in the prediction accuracy and discussing further open issues for performance prediction in virtualized environments.**

I. Introduction

In application performance and resource management, service providers are faced with questions such as: How much resources should be allocated to meet service-level agreements (SLAs)? How should the system configuration be adapted to avoid performance problems arising from changing customer workloads? Answering such questions for non-virtualized execution environments is already a complex task [1]. In virtualized environments, this task is even more complicated because resources are shared. Moreover, since changes in the usage profiles of one service may affect services that are hosted on the same physical host, capacity planning has to be performed continuously during operation. *Proactive* application performance management requires the ability to predict performance of applications in the following two recurring scenarios: i) under varying service workloads in order to anticipate when performance SLAs will be violated based on workload forecasts ii) under different system configurations in order to find a suitable configuration that avoids SLA violations and ensures efficient resource usage. Given that computation details are abstracted by an increasingly deep virtualization layer, the following research questions arise: What is the performance overhead when virtualizing execution environments? How should the overhead be taken into account when conducting performance predictions? Can the performance-influencing factors be abstracted in a generic performance model?

Table I shows a motivating example illustrating the influence of the virtualization overhead on application-level performance metrics. We compare the performance of an operation `CreateVehicleEJB` of the SPECjEnterprise2010[1] benchmark in two different deployment scenarios. In the first scenario, the benchmark is deployed in a native application server without use of virtualization. In the second scenario, the benchmark is deployed in a virtualized application server on an identical physical machine. The application server virtual machine (VM) is the only guest VM hosted by the hypervisor. We investigate the operation's average response times and the utilization of the application server under four different load conditions (in a range from 15% to 60% utilization of the application server CPU). In the four load scenarios, the virtualized application server is slightly higher utilized than the native one. The average response times, however, are significantly higher in the virtualized setup. Obviously, the virtualization overhead plays an important role when investigating performance properties in virtualized systems.

Throughput	Native AppServer		Virtualized AppServer	
X	$U_{AppServer}$	R_{Avg}	$U_{AppServerVM}$	R_{Avg}
35	14.9%	26ms	15.8%	32ms
65	24.8%	27ms	27.2%	39ms
100	35.7%	28ms	40.0%	48ms
154	53.2%	31ms	60.7%	100ms

TABLE I: Native vs. Virtualized Setup: Response Times of `CreateVehicleEJB` in SPECjEnterprise2010

Existing approaches to quantify, model and predict the virtualization performance overhead often do this either in a black-box fashion (using for example constant overhead factors [2] or probabilistic models that do not capture the performance-relevant factors explicitly [3]) or focus on specific aspects such as storage I/O [4] or cache contention effects [5].

In this paper, we provide the following contributions: i) an approach to estimate virtualization overhead factors. ii) a generic approach how to consider virtualization overheads in classical queueing network models, and iii) an evaluation in a representative environment. In a set of evaluation scenarios based on the SPECjEnterprise2010 standard benchmark and XenServer 5.5, we show significant improvements in the prediction accuracy using our approach. At the same time, we discuss open issues and challenges of performance prediction in virtualized environments.

[1] SPECjEnterprise2010 is a trademark of the Standard Performance Evaluation Corp. (SPEC). The SPECjEnterprise2010 results or findings in this publication have not been reviewed or accepted by SPEC, therefore no comparison nor performance inference can be made against any published SPEC result. The official web site for SPECjEnterprise2010 is located at http://www.spec.org/jEnterprise2010.

Throughput X	Virtualized AppServer $U_{Domain0}$	$U_{AppServerVM}$	R_{Avg}
35	11.6%	15.8%	32ms
65	18.9%	27.2%	39ms
100	25.2%	40.0%	48ms
154	30.6%	60.7%	100ms

TABLE II: VM Utilization vs. Domain-0 Utilization when running `CreateVehicleEJB` in SPECjEnterprise2010

II. Background

Virtualization Technologies: The Xen hypervisor is an open source bare-metal hypervisor (type-I). With the Xen hypervisor, multiple para-virtualized or full-virtualized virtual machines (guest domains) can be executed on a single server sharing the physical resources. A scheduler integrated in the hypervisor schedules the access of all domains to the available physical CPUs. For access to other devices and for managing the guest domains, Xen uses a privileged control domain (Domain-0). Domain-0 contains the device drivers to access the physical devices. All communication of the guest domains with the physical devices goes through Domain-0. This causes additional management overhead in terms of CPU consumption. For example, if a guest domain sends a disk I/O request, Domain-0 requires CPU time to process the request on behalf of the guest domain.

The VMware ESX hypervisor uses another popular I/O model for VMs. In ESX, it is the hypervisor itself that contains device driver code and provides shared access to the physical devices. However, the problem of accounting the virtualization overhead induced by I/O accesses remains. In this paper, we demonstrate our findings and approaches using XenServer 5.5, although most of them can be applied in a straight-forward manner to other state-of-the-art virtualization platforms. Considering Xen as an example, we specifically address the additional challenge of modeling the virtualized application and the driver domain (Domain-0) separately [6].

Measuring and Accounting VM Overhead: We use Xenmon [7], [6] to obtain monitoring data from a Xen-virtualized system. Reported metrics include CPU utilization, network I/O and disk I/O accesses. Xenmon collects monitoring data both from the hosted guest VMs and from Domain-0, i.e., it reports resource usage of physical resources as well as resource usage at the virtualization layer issued by guest VMs and Domain-0.

When it comes to quantifying the virtualization overhead, the resource utilization attributed to Domain-0 becomes of interest. The monitoring data in Table II shows the same scenario as in Table I with added information on the utilization accounted to Domain-0, reported by Xenmon. The utilization of Domain-0 ranges from 12% to 31%, depending on the load level. At the highest load level, the utilization of Domain-0 is half of the utilization measured directly at the application server VM. This explains the highly increased response times of `CreateVehicleEJB`. The system is already heavily utilized, i.e., $> 90\%$. Recall that the application server VM is the only running guest VM and thus the only one responsible for the observed Domain-0 load. However, when different guest VMs are consolidated on one physical machine, the utilization accounted to Domain-0 is partitioned among the guest VMs. When considering the physical resource utilization caused by a specific VM, the corresponding Domain-0 partition has to be added to the utilization obtained at the VM level.

$D_{w_{i,j}}$	Mean service demand of workload class $w_{i,j}$ (*without* the virtualization overhead).
$O_{w_{i,j}}$	Virtualization overhead factor for workload class $w_{i,j}$.
$X_{w_{i,j}}$	Throughput of workload class $w_{i,j}$.

TABLE III: Performance Model Parameters

III. Problem Formulation

A hypervisor running on a physical machine hosts n virtual machines $VM_1, \ldots, VM_n$, where VM_i processes workload classes $w_{i,j}$. The virtual machines and the hypervisor share the available physical resources. The focus is on the CPU resource because other metrics such as disk and network I/O rates are not directly impacted by the virtualization layer, given that running an application in a virtualized environment will not cause more network packets to be sent or more disk requests to be generated. However, as explained in Section II, even when focusing on the physical CPU resource, the network and disk I/O traffic of the VMs cannot be neglected since they induce additional processing overhead.

The major performance metrics of interest are the response times $R_{w_{i,j}}$ of a service belonging to a certain workload class $w_{i,j}$, the CPU resource utilization of the physical machine (U_{total}), and the utilization measured inside a virtual machine VM_i (U_{VM_i}) without the incurred virtualization overhead. To motivate the proposed approaches for taking the virtualization overhead into account, we shortly describe the native base scenario with one application running on one physical machine. According to classical performance modeling techniques, as described for example in [1], the physical machine can be modeled as a queue. The service demand D_w of workload class w can be obtained using the Service Demand Law [1], [8] $D_w = U^w_{total}/X_w$, where U^w_{total} is the portion of the utilization caused by workload class w, and X_w is the request throughput of the workload class. Observations X_w and U^w_{total} have to refer to the same timeframe.

When running the application in a virtualized environment, the following issues need to be addressed: i) How to obtain the virtualization overhead of workload class $w_{i,j}$? ii) How to include the virtualization overhead in the service demand of workload class $w_{i,j}$? iii) How and where to schedule the virtualization overhead? The following sections describe how we handle these issues.

IV. Performance Model Parameterization

Table III shows the relevant model parameters when modeling application performance in virtualized infrastructures. The service demands $D_{w_{i,j}}$ can be derived using existing techniques that are based on utilization U_{VM_i} and throughput measurements $X_{w_{i,j}}$ inside the VM [1], [8]. The challenge is to estimate the virtualization overhead factors $O_{w_{i,j}}$. There are different ways to obtain the relevant overhead, depending on the amount and type of available monitoring data.

At design-time and deployment-time, typically (micro-)benchmarks are used to characterize the target virtualization infrastructure [9], [6], [7], [10], [11]. By comparing benchmark results on the native and the virtualized system, benchmark-specific virtualization overheads are derived. A virtualization overhead factor $O_{w_{i,j}}$ is then estimated according to the most similar mix of benchmark workload types corresponding to workload class $w_{i,j}$.

In the following, we present how we combine the work of [12] and [6] as an approach to estimate virtualization overhead factors based on *run-time* Domain-0 utilization measurements. We compute

$$O_{w_{i,j}} = 1 + U^{w_{i,j}}_{Dom0} / U^{w_{i,j}}_{VM_i},$$

where $U^{w_{i,j}}_{VM_i}$ is the measured utilization of VM_i due to $w_{i,j}$ and $U^{w_{i,j}}_{Dom0}$ is the measured Domain-0 utilization partition induced by $w_{i,j}$. However, when running more than one VM, the latter Domain-0 partition has to be approximated. The authors in [12] describe a method that partitions the Domain-0 utilization in different blocks, where each block can be assigned to a guest VM that caused the utilization. As input, the method requires VM monitoring data of disk I/O, network I/O and CPU usage for each guest VM as well as for Domain-0. The output is an approximation of the per-VM physical resource utilization. The method uses a regression model based on micro-benchmark results (obtained in a native and a virtualized setup) as a starting point, and calibrates the model using run-time monitoring data of disk I/O and network I/O metrics as well as resource utilization metrics of Domain-0 and the guest VMs. The regression model estimation error is continuously observed and triggers a calibration process when a certain threshold is reached. This way, the model reflects workload dynamics that may be caused by changes in workload patterns. The authors propose a guided regression as a calibration approach, for details we refer to [12]. Note that the evaluation scenarios for partitioning CPU utilization presented in [12] exhibit a relative error of lower than 10%, mostly around 5%.

V. Performance Model Structure

In this section, we describe how to construct performance models that take the virtualization overhead into account. The idea is to adapt classical performance modeling techniques making them usable for performance prediction in virtualized environments. In this paper, we use queueing networks to model system performance. We assume that the relevant model parameters listed in Table III are available including the number of VMs, the VM-specific CPU demands and the VM-specific overhead in terms of induced CPU demand on Domain-0.

In a virtualized environment there are n VMs sharing the same physical machine which has m cores. Each VM_i has an allocated number of vCPUs ranging from 1 to m. To describe the mapping of queues to processing resources in an abstract fashion, we first introduce a notation for the resource allocations: The available processing resources, i.e., the number of cores, are represented by the set $\text{Res}(total) := \{0, \dots, m-1\}$. The set of resources assigned to VM_i is denoted as $\text{Res}(VM_i) \subseteq \text{Res}(total)$. $\text{Res}(Dom0)$ is defined accordingly. Figure 1 illustrates an example of possible resource allocations. There are four VMs $VM_1, \dots, VM_4$ and Domain-0, each assigned a portion of the available

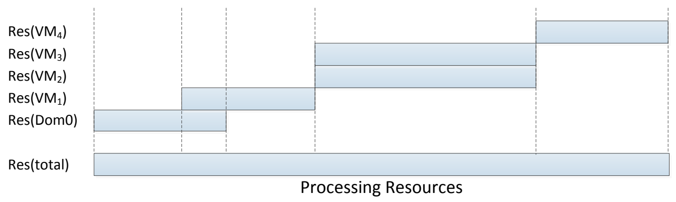

Fig. 1: Assigned Processing Resources in a Virtualized Environment

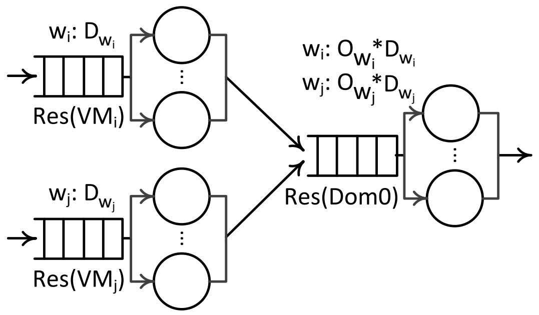

Fig. 2: Disjoint Resource Sets

processing resources $\text{Res}(total)$. Resource sets $\text{Res}(VM_2)$ and $\text{Res}(VM_3)$ are *identical*. Resource sets $\text{Res}(VM_2)$ and $\text{Res}(Dom0)$ are *partially overlapping*. The remaining pairs of resource sets are *disjoint*.

Figures 2 and 3 show how different types of resource sets can be mapped to queueing networks taking into account the resource sharing. The virtualization overhead $O_{w_{i,j}} * D_{w_{i,j}}$ is induced on Domain-0 *after* processing the VM-internal service demand $D_{w_{i,j}}$, i.e., the virtualization overhead O for a service demand D is assumed to *not* be executed in parallel. This is because VMs are typically blocked due to I/O wait when using Domain-0.

Figure 2 shows how a disjoint pair of resource sets $\text{Res}(VM_i)$ and $\text{Res}(VM_j)$ is mapped to queues. Non-overlapping resource sets are modeled as separate (multi-server) queues. The overhead is scheduled on a separate dedicated queue representing Domain-0.

Figure 3(a) and Figure 3(b) show how pairs of identical resource sets are mapped to queues. In case one of the

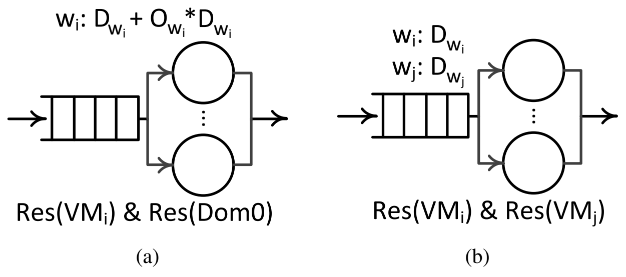

Fig. 3: (a) Equal Resource Sets $\text{Res}(VM_i)$ and $\text{Res}(Dom0)$, (b) Equal Resource Sets $\text{Res}(VM_i)$ and $\text{Res}(VM_j)$

resource sets represents Domain-0, i.e., the pair is $\{\text{Res}(VM_i), \text{Res}(Dom0)\}$, the resource sets are mapped to one (multi-server) queue with a service demand for workload class w_i of $D_{w_i} + O_{w_i} * D_{w_i}$ (see Figure 3(a)). For a pair $\{\text{Res}(VM_i), \text{Res}(VM_j)\}$, the resource sets are mapped as shown in Figure 3(b). There is one (multi-server) queue with two workload classes w_i, w_j having VM-internal service demands D_{w_i}, D_{w_j}.

For partially overlapping resource sets $\text{Res}(VM_i)$ and $\text{Res}(VM_j)$, the service demands D_{w_i}, D_{w_j} may be scheduled in three different ways. There are demands D_{w_i} that are scheduled on the resource set $\text{Res}(VM_i) \setminus \text{Res}(VM_j)$, respectively demands D_{w_j} that are scheduled on the resource set $\text{Res}(VM_j) \setminus \text{Res}(VM_i)$, and demands of both D_{w_i} and D_{w_j} that are scheduled on the resource set $\text{Res}(VM_i) \cap \text{Res}(VM_j)$. The probability of each of the described scheduling paths depends on the hypervisor scheduler. Such complex scheduling strategies are not supported in standard queueing network models.

VI. Evaluation

We installed the SPECjEnterprise2010 Java EE benchmark in a system environment we describe in the following. The benchmark application is deployed in VMs hosted on XenServer 5.5. The VMs are running CentOS 5.3 and we use Oracle WebLogic Server (WLS) 10.3.3 as application server. The virtualized blade server has a 4-core Intel Xeon CPU. As a database server (DBS), we use Oracle Database 11g, running on four 6-core AMD CPUs. The benchmark driver is running on a separate blade server. The benchmark workload is generated by an application that is modeled after an automobile manufacturer. The benchmark driver executes five benchmark operations. A dealer may `Browse` through the catalog of cars, `Purchase` cars, or `Manage` his dealership inventory, i.e., sell cars or cancel orders. In the manufacturing domain, work orders for manufacturing vehicles are placed, triggered either through WebService or RMI calls (`CreateVehicleWS` or `CreateVehicleEJB`).

For the evaluation, we considered different resource allocation scenarios while varying the number of VMs, the VM configuration, amount of resources as well as the type and intensity of the workload. We selected the two SPECjEnterprise2010 benchmark operations `CreateVehicleEJB` and `Browse` as workload types. We injected different load levels, ranging from low to high load. For performance modeling, we used classical queueing network models, which were directly mapped to Queueing Petri Nets (QPNs) and solved using the simulation engine SimQPN [13]). Furthermore, note that in our setup the database server is clearly under-utilized. For the considered injection rates, the response times of database requests are almost constant. This is important because our study is focussed on evaluating the performance of the application server when running it to a virtualized environment.

Scenario 1: In this scenario, we considered a VM with one vCPU running next to Domain-0, i.e., without any overlap in the resource allocations. Note that in XenServer 5.5, Domain-0 has one vCPU allocated by default. Given that there is only one VM, we can obtain the virtualization overhead factor by considering U_{Dom0} and $U_{VM_{WLS}}$. For the operation `CreateVehicleEJB`, the overhead factor is 0.34, for

Throughput X	-Baseline- Rel. Prediction Error U_{WLS}	U_{Dom0}	R_{Avg}	Rel. Prediction Error U_{WLS}	U_{Dom0}	R_{Avg}
`CreateVehicleEJB`, $\text{Res}(VM_{WLS}) = \{1\}, \text{Res}(Dom0) = \{0\}, m = 2$						
17.8	-30%	n.a.	-15%	-3%	-5%	-2%
32.6	-26%	n.a.	-16%	-2%	-1%	-5%
50.0	-24%	n.a.	-17%	+3%	+6%	-6%
76.8	-22%	n.a.	-13%	+3%	+17%	-4%
`Browse`, $\text{Res}(VM_{WLS}) = \{1\}, \text{Res}(Dom0) = \{0\}, m = 2$						
23.1	-25%	n.a.	-23%	-4%	-5%	-7%
46.4	-21%	n.a.	-15%	-1%	-1%	-2%
69.3	-21%	n.a.	-13%	-1%	+2%	-3%
92.7	-20%	n.a.	-4%	+1%	+6%	+5%

TABLE IV: Results: Scenario 1 with a single VM

Throughput X_{CVE}, X_{BRO}	Measured U_{total}	R^{CVE}_{Avg}	R^{BRO}_{Avg}	Rel. Prediction Error U_{total}	R^{CVE}_{Avg}	R^{BRO}_{Avg}
`CreateVehicleEJB` (`CVE`) and `Browse` (`BRO`): $\text{Res}(VM_{\text{BRO}}) = \text{Res}(VM_{\text{CVE}}) = \{0, \ldots, 3\}, \text{Res}(Dom0) = \{?\}, m = 4$						
35.7, 42.4	0.23	34ms	16ms	-11%	-41%	-11%
64.6, 77.3	0.37	39ms	19ms	-0%	-48%	-23%
100.1,119.8	0.54	54ms	34ms	+5%	-61%	-52%

TABLE VI: Results: Scenario 3 with two VMs

`Browse`, it is 0.29. We evaluate the prediction accuracy for two kinds of performance models. First, we build a *baseline* model where we obtain service demand $D_{\texttt{CreateVehicleEJB}}$ using $U_{VM_{WLS}}$, i.e., without considering the virtualization overhead. Second, we build a model following the approach described in Section V, explicitly taking into account the virtualization overhead. Table IV shows the corresponding prediction errors. The utilization and response time predictions are clearly of much higher accuracy compared to the baseline model.

Scenario 2: Here, we consider two VMs, the first one running `CreateVehicleEJB`, the second one running `Browse`. While the guest VMs are sharing physical cores 1 to 3, Domain-0 is pinned to core 0. The performance model is parameterized with measurement data obtained at the medium load level, taking the virtualization overhead into account as described in Section V. Table V shows the measurement results and prediction errors. The total utilization as well as Domain-0-specific utilization is predicted with an error of less than 10%. The response time predictions are of lower accuracy (with error up to 40%) due to low level synchronization effects in Domain-0 that are not reflected in our model. We intentionally refrain from modeling such low level details of the hypervisor scheduler in order to strike a balance between model accuracy and model compactness. Prediction errors of up to 40% for system response time are typically considered acceptable for capacity planning purposes [1].

Scenario 3: In this scenario, we again run two VMs (`CreateVehicleEJB` and `Browse`). In contrast to the previous scenario, the guest VMs are this time sharing four vCPUs and Domain-0 is *not* pinned to a certain core. Since Domain-0 is expected to be executed on the four cores as they become available, we can consider $\text{Res}(Dom0) = \{0, \ldots, 3\}$. Table VI shows the corresponding results. While for the utilization, the prediction error rates are acceptable, the predicted response times have a high error.

Throughput	**Measured**				**Rel. Prediction Error**			
X_{CVE}, X_{BRO}	U_{total}	U_{Dom0}	R_{Avg}^{CVE}	R_{Avg}^{BRO}	U_{total}	U_{Dom0}	R_{Avg}^{CVE}	R_{Avg}^{BRO}
`CreateVehicleEJB` (`CVE`) and `Browse` (`BRO`): $\text{Res}(VM_{\text{BRO}}) = \text{Res}(VM_{\text{CVE}}) = \{1, \dots, 3\}, \text{Res}(Dom0) = \{0\}, m = 4$								
35.5, 42.5	0.21	0.20	29ms	14ms	+1%	-11%	-28%	+7%
65.1, 77.7	0.37	0.33	33ms	16ms	+4%	+0%	-34%	+0%
100.6,119.5	0.58	0.45	42ms	29ms	+5%	+12%	-41%	-33%

TABLE V: Results: Scenario 2 with two VMs

VII. Related Work

Approaches to predict the performance overhead based on machine learning techniques include for example [14], [15]. Kousiouris et al. use Artificial Neural Networks (ANNs) to predict the impact of real-time scheduling parameters, VM deployment and workload type on the system performance based on measurements using six different Matlab benchmark tests [14]. Kundu et al. also use benchmark results to train ANNs to predict the performance overhead for CPU, memory and I/O [15]. Watson et al. [3] use quantile regression to obtain a probabilistic performance model making it possible to predict the response time of the RUBiS benchmark depending on the allocated resources at the VM level. However, in all these approaches, a performance model that makes the performance-relevant factors explicit is not provided.

Jung et al. [2] use layered queueing networks (LQN) for performance prediction. They claim to explicitly consider the performance-relevant influences of the hypervisor as part of their modeling approach, which is evaluated in the context of the RUBiS benchmark. Unfortunately, they do not describe or quantify the specific parameters they use, making it difficult to generalize this approach and apply it to other performance models. The approach of Menascé [16] uses analytical queueing models to quantify the slowdown of virtualized applications in server consolidation scenarios. However, in [6], [12] it is shown that using the total CPU time as apportionment factor to derive workload-specific virtualization overhead, as done in [16], typically results in a rough approximation at best.

VIII. Concluding Remarks

In this paper, we evaluated how to predict the performance of virtualized environments. We described different ways to obtain relevant model parameters, such as the virtualization overhead, depending on the amount and type of available monitoring data. We provided a generic approach how to integrate the virtualization overhead in queueing network models.

We evaluated different virtualization setups in a representative experimental environment. We used the industry-standard SPECjEnterprise2010 benchmark application and deployed it on a XenServer 5.5 virtualization platform. In single VM setups, the model predictions reach a satisfying accuracy and clearly outperform the baseline where the virtualization overhead is neglected. In consolidated virtualization environments with more than one guest VM, the modeling approach based on classical queueing networks is less accurate for response times and still quite accurate for predicting VM utilization and physical machine utilization. Since the response times are typically underestimated, Domain-0 seems to induce additional delays and contention effects that are not yet captured by our queueing model. We intentionally refrained from modeling such low level details of the hypervisor scheduler since we aimed to come up with a compact model that is practically usable and provides a balance between accuracy and complexity. As part of our future work, we plan to consider further VMM configuration options resulting in more complex resource allocation scenarios.

References

[1] D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy, *Capacity Planning and Performance Modeling˜- From Mainframes to Client-Server Systems*. Prentice Hall, Englewood Cliffs, NG, 1994.

[2] G. J. G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu, "Generating adaptation policies for multi-tier applications in consolidated server environments," in *ICAC*, 2008.

[3] B. J. Watson, M. Marwah, D. Gmach, Y. Chen, M. Arlitt, and Z. Wang, "Probabilistic performance modeling of virtualized resource allocation," in *ICAC*, 2010.

[4] G. Casale, S. Kraft, and D. Krishnamurthy, "A model of storage i/o performance interference in virtualized systems," in *Int. Conf. on Distributed Computing Systems Workshops*, 2011.

[5] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell, "Modeling virtual machine performance: challenges and approaches," *SIGMETRICS Performance Evaluation Review*, 2010.

[6] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy, "Profiling and modeling resource usage of virtualized applications," in *Middleware*, 2008.

[7] D. Gupta, R. Gardner, and L. Cherkasova, "XenMon: QoS monitoring and performance profiling tool," HP Labs, Tech. Rep. HPL-2005-187, 2005.

[8] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi, "Cpu demand for web serving: Measurement analysis and dynamic estimation," *Performance Evaluation*, vol. 65, no. 6 - 7, pp. 531 – 553, 2008.

[9] N. Huber, M. von Quast, M. Hauck, and S. Kounev, "Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments," in *Int. Conf. on Cloud Comp. and Services Science*, 2011.

[10] Y. Koh, R. C. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *Int. Symp. on Performance Analysis of Systems and Software*, 2007.

[11] K. Ye, X. Jiang, S. Chen, D. Huang, and B. Wang, "Analyzing and modeling the performance in xen-based virtual cluster environment," in *Int. Conf. on High Perf. Computing and Communications*, 2010.

[12] L. Lu, H. Zhang, G. Jiang, H. Chen, K. Yoshihira, and E. Smirni, "Untangling mixed information to calibrate resource utilization in virtual machines," in *Int. Conf. on Autonomic Computing*, 2011.

[13] S. Spinner, S. Kounev, and P. Meier, "Stochastic Modeling and Analysis using QPME: Queueing Petri Net Modeling Environment v2.0," in *Int. Conf. on Application and Theory of Petri Nets and Concurrency*, 2012.

[14] G. Kousiouris, T. Cucinotta, and T. Varvarigou, "The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks," *Journal of Systems and Software*, vol. 84, no. 8, 2011.

[15] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta, "Modeling virtualized applications using machine learning techniques," in *International Conference on Virtual Execution Environments*, 2012.

[16] D. A. Menascé, "Virtualization: Concepts, applications, and performance modeling," in *CMG-CONFERENCE*, 2005.