AppRAISE: Application-Level Performance Management in Virtualized Server Environments

Zhikui Wang, Yuan Chen, Daniel Gmach, Sharad Singhal, Brian J. Watson, Wilson Rivera, Xiaoyun Zhu, and Chris D. Hyser

***Abstract*—Managing application-level performance for multi-tier applications in virtualized server environments is challenging because the applications are distributed across multiple virtual machines, and workloads are dynamic in their intensity and transaction mix resulting in time-varying resource demands. In this paper, we present AppRAISE, a system that manages performance of multi-tier applications by dynamically resizing the virtual machines hosting the applications. We extend a traditional queuing model to represent application performance in virtualized server environments, where virtual machine capacity is dynamically tuned. Using this performance model, AppRAISE predicts the performance of the applications due to workload changes, and proactively resizes the virtual machines hosting the applications to meet performance thresholds. By integrating feed-forward prediction and feedback reactive control, AppRAISE provides a robust and efficient performance management solution. We tested AppRAISE using Xen virtual machines and the RUBiS benchmark application. Our empirical results show that AppRAISE can effectively allocate CPU resources to application components of multiple applications to meet end-to-end mean response time targets in the presence of variable workloads, while maintaining reasonable trade-offs between application performance, resource efficiency, and transient behavior.**

***Index Terms*—virtualization, performance model, performance control, resource allocation, workload consolidation.**

I. Introduction

Server virtualization enables consolidation of applications onto a shared hardware infrastructure, allowing IT to increase the agility of the overall IT infrastructure while reducing hardware, software, power, cooling, real estate, and other costs. As a result, virtualized server environments have drawn interest from many businesses. However, workloads for Internet-facing multi-tier applications can fluctuate considerably, and thus statically-configured virtual resources suffer much the same fate as dedicated physical resources: they are often either over-provisioned or over-loaded. To avoid this problem, popular modern virtualization technologies expose interfaces that permit rapid and frequent resource reallocations. In principle, such interfaces make it possible to adjust resource allocations to different applications in response to workload fluctuations, but it is difficult to do this well in practice due to the following challenges:

- Workload demand is volatile over both short and long timescales. Workloads frequently show diurnal and seasonal cycles, but also contain irregular patterns including "flash crowds". Even when transaction rates are steady, the mix of transaction types reaching the application can change over time. This characteristic is called *non-stationarity* [1] and is significant because it causes resource requirements of the application to change, since different transaction types can have different resource demands.
- Another difficulty lies in the fact that most Internet applications with multi-tier architectures can be distributed across multiple virtual machines (VMs) within a virtualized server environment. Since physical servers may run VMs hosting components from multiple applications, different applications may compete with one another for resources in complicated ways across many servers. Furthermore, these interactions may show complicated dynamics that vary across both time and space (servers) because changes in transaction types can cause resource demands to shift across application tiers.
- Finally, management goals are naturally stated in terms of application-level performance, such as transaction response time and throughput. Usually, the relationship between these performance metrics and the dynamic resource allocation controls exposed by virtualization technologies is non-trivial. While there has been substantial progress in recent years for application-level performance modeling of non-virtualized environments [2]–[4], virtualization makes performance modeling fundamentally harder by allowing available capacities to be changed over time.

It is thus difficult to properly tune resource allocations to virtual machines in response to dynamically changing workloads. While several existing management products [5], [6] can dynamically size VMs to maintain given resource utilization targets, unfortunately, the appropriate utilization target itself is application dependent and varies over time for a given application as the transaction mix changes.

In this paper, we present AppRAISE, a management system for application-level performance control in virtualized server environments. AppRAISE manages application-level objectives (e. g., response time thresholds) based on control theory

Manuscript received January 17, 2009; revised May 31, 2009 and August 17, 2009. The associate editor coordinating the review of this paper and approving it for publication was J. L. Hellerstein.

Z. Wang, Y. Chen, D. Gmach, S. Singhal, B. J. Watson and C. D. Hyser are with Hewlett Packard Laboratories (e-mail: {zhikui.wang, yuan.chen, daniel.gmach, sharad.singhal, brian.j.watson, chris_hyser}@hp.com).

W. Riversa is with University of Puerto Rico at Mayaguez (email: wilson.rivera@ece.uprm.edu).

X. Zhu is with VMware Inc. (email: xzhu@vmware.com).

This work was done while X. Zhu and W. Riversa were with Hewlett Packard Laboratories.

Digital Object Identifier 10.1109/TNSM.2009.04.090404

and performance modeling methodologies. When combined with VM placement and capacity planning tools [7], [8], AppRAISE can enable high utilization of server resources for VMs while meeting performance requirements of the managed applications. This paper focuses specifically on dynamic allocation of CPU resources and application mean response time guarantee, though the framework can be extended to the allocation of other types of resources (e. g., network bandwidth) and application performance metrics (e. g., throughput).

We make three contributions in this paper:

- First, we develop a hierarchical control architecture that adjusts CPU allocations of VMs to achieve end-to-end mean response time targets for the applications hosted by the VMs. The architecture incorporates an application controller for each application to tune the utilization targets of the VMs hosting the application components, and a node controller for each physical server that provides local resources to meet the time-varying demand of the workloads.
- Second, we propose a performance model that captures the effects of CPU capacity, time-varying workload intensity and transaction mix on application mean response time in virtualized server environments. We take queuing models and transaction mix models [1] originally proposed for physical servers, and extend them to model application performance for virtualized servers. Our models can be calibrated using only lightweight measurements that are routinely collected in today's production environments, without invasive instrumentation. We apply the models in the application controller which integrates proactive control and reactive feedback control for fast and robust control of the applications.
- Third, we validate the performance models and ensemble of controllers in a Xen-virtualized environment. Experiments were driven by workloads to mimic the realistic traces collected from production systems. Our results demonstrate that AppRAISE automatically adjusts performance-critical Xen parameters in response to workload fluctuations, and maintains application response time at specified targets without resource over-provisioning.

The remainder of this paper is organized as follows. The controller architecture is described in detail in Section II. We present the performance models in Section III. The implementation of AppRAISE is described in Section IV. The experimental validation is presented in Section V. We review related work in Section VI, and conclude the paper in Section VII.

II. Architecture and Controller Design

A. Problem Statement

We consider a server environment where multiple applications are hosted by a common pool of virtualized server resources. Each application consists of several interacting components, each of which runs in a virtual machine. Resources shared by virtual machines on a physical server, including CPU capacity, disk access bandwidth and network I/O bandwidth, are allocated to the virtual machines at run time through virtual machine monitors or hypervisors (e. g., Xen CPU scheduler).

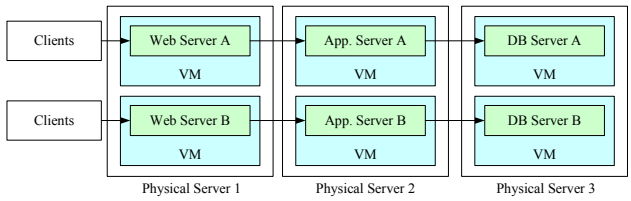


Fig. 1: An example of two 3-tier applications in virtualized server environments.

As an example, Figure 1 shows two 3-tier applications running on three physical servers. Each application consists of a web tier, an application tier and a database tier. Each tier runs in an individual virtual machine. In this example, the virtual machines hosting the same tier (e. g., web) are placed on the same physical server. In practice, the virtual machines can be distributed across servers using any reasonable placement.

Each application is assumed to have its own performance target or threshold. The actual performance is affected by the resource capacity available to the application, and the time-varying demands of the workload due to changes of workload intensity and/or workload transaction mix. Our goal is to meet the application-level performance targets for the multiple applications hosted in the virtualized environment through dynamic resource allocation, while not over-provisioning the total amount of resources to the applications.

To avoid confusion in terminology, we define a few critical concepts used throughout this paper. We use resource *capacity* to refer to the total amount of resources that a physical server provides, e.g., 100 CPU shares for a server with one CPU. Resource *entitlement* or *allocation* e refers to the resource capacity that is allocated to a virtual machine. Resource *consumption* c means the actual resource consumed by a virtual machine. We further define the resource *utilization* u of a virtual machine as the ratio between resource consumption of the virtual machine and resource entitlement to the virtual machine $u = c/e$. In contrast with physical servers, resource entitlement can be dynamically modified in virtualized servers. This results different performance models, which will be discussed more in detail in Section III.

B. Overall Architecture

Figure 2 shows the architecture of AppRAISE. The bottom part of the figure shows the managed infrastructure. Each physical server (node) runs multiple virtual machines. Each virtual machine hosts a component of an application, which can span multiple hosts. For instance, Figure 2 shows that components of App 1 are running on at least Server 1 and Server 2. In the management domain, there are layers of controllers, taking policies on the applications as inputs, and deciding the resource entitlement for each virtual machine in real time. More specially,

- The *configuration service* maintains information on the configuration of the system, such as the controller parameters and the location of the virtual machines hosting components of applications. The information can

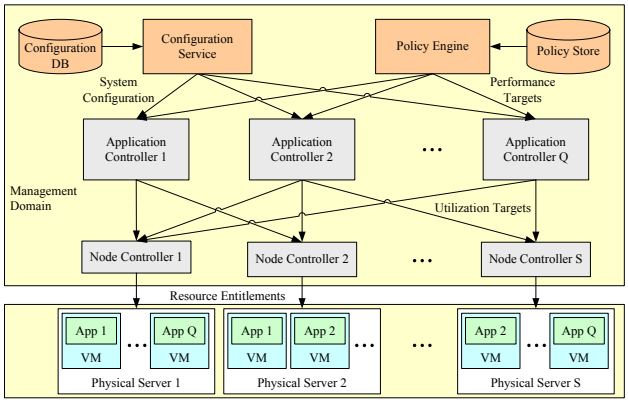


Fig. 2: AppRAISE architecture.

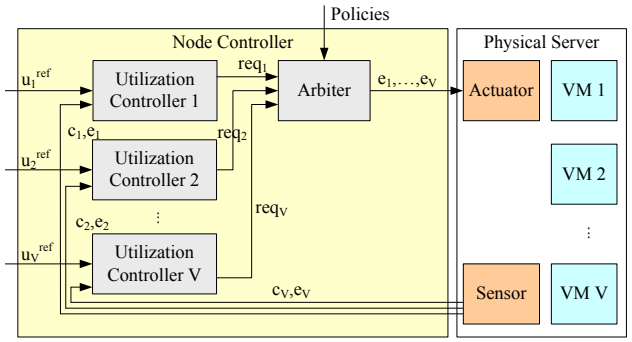


Fig. 3: Node controller architecture.

be updated dynamically by other services such as a VM placement service. Application controllers use that information for control and communication, given that the mapping between applications and physical servers can change over time.

- The *policy engine* provides high level policies to the controllers using a central policy store. Policy examples include performance targets for the applications, their priority or associated utility functions.
- Each application is managed by an *application controller* to maintain its performance target. The controller collects the application's performance and statistics on the workload, and then periodically determines resource utilization targets for the virtual machines hosting the application components.
- Each physical server is controlled by a *node controller*. It computes the resource entitlement for each VM to meet the resource utilization target as determined by the application controller. The needed entitlement can be dependent on the workload and utilization target. If the total demand exceeds the server capacity, the node controller allocates available resources to the VMs based on the predefined policies.

AppRAISE uses a *distributed* and *hierarchical* management framework. Each application controller only needs to manage the components of the associated application and each node controller addresses only the virtual machines on the associated node. This layered design allows the architecture to scale up to accommodate a large number of applications, servers and virtual machines in data centers. It also permits integration with migration controllers [9] that can live-migrate virtual machines to meet the total resource demand of the applications. Integration can be done through the configuration service, the policy engine, or other interfaces as defined in [7].

The remainder of this section provides more details on the architecture and algorithms of the node controller and application controller.

C. Node Controller

Figure 3 shows the architecture of the node controller. The role of a node controller is to maintain the utilization targets for all virtual machines on the node by dynamically adjusting their resource entitlement. It implements two main functions: the utilization controller and the arbiter. The per-VM *utilization controller* measures the VM's resource consumption c_v, and determines the ideal entitlement, or resource request req_v, that can satisfy the utilization target u_v^{ref} provided by the application controller. However, the total demand, or the aggregate of the resource requests may exceed the capacity of the server. The per-node *arbiter* then determines the actual entitlement e_v for each virtual machine, based on both the resource requests and the policies defined for the servers (e. g., whether or not to distribute the excessive capacity) and the virtual machines (e. g., priorities of the applications).

The CPU utilization controller uses an adaptive controller described in [10]. For each virtual machine v, the CPU request req_v in control interval k is set to

$$req_v(k) = e_v(k-1) - \frac{\gamma c_v(k-1)}{u_v^{ref}} \left(u_v^{ref} - u_v(k-1) \right), 0 < \gamma < 2, \quad (1)$$

where $u_v(k-1) = c_v(k-1)/e_v(k-1)$, and $c_v(k-1)$ and $e_v(k-1)$ are the measured CPU consumption and the CPU entitlement in the last interval $(k-1)$ and γ is the controller gain parameter. In this controller, the change of the resource request in the current interval is proportional to the observed utilization error, but the gain varies along with the resource consumption and the utilization target. When the virtual machine is more heavily loaded, CPU capacity is allocated to the VM more aggressively so that application performance does not suffer too much. On the other hand, the controller slowly reduces CPU allocation when the consumption is much less than the entitlement, providing a capacity buffer for bursty workloads having future demand spikes. The closed-loop system of utilization control has been shown in [10] to be stable. Experimental results in [10] also indicated that, for workloads with time-varying resource demand, this adaptive controller leads to lower response time and higher throughput with less amount of CPU resource, when compared with a fixed-gain integral controller.

The utilization controllers run independently of one another with all requests being sent to the arbiter. The arbiter decides how to allocate excess capacity when the aggregate request is less than capacity, and how resources are shared by the virtual machines when resources are under contention. As a special case, the excess capacity can be reserved to accommodate other applications that can be migrated to the server through through live virtual machine migration and workload consolidation. Alternatively, the capacity of the server can be

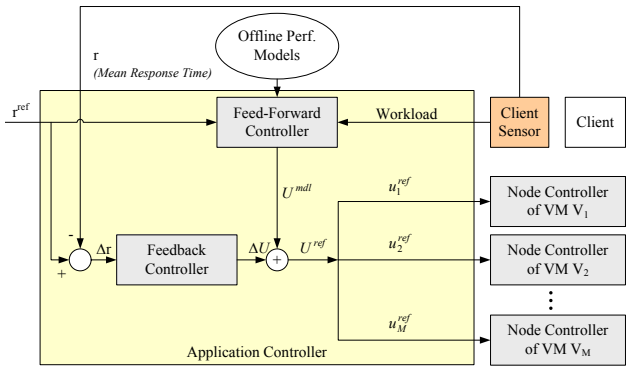


Fig. 4: Feedback and feed-forward application controller.

shared proportionally by the VMs during contention.

D. Application Controller

Figure 4 shows the architecture of the application controller. It controls a single application composed of a number of components, each individually hosted inside a virtual machine. The application controller is responsible for generating an appropriate utilization target for each virtual machine hosting an application component to ensure that the performance goal is met. Each application controller consists of a *feed-forward controller* and a *feedback controller*. The feed-forward controller estimates the expected utilization targets $U^{mdl} = [u_1^{mdl}, \ldots, u_V^{mdl}]$ for all of the application components based on a performance model, the performance target, and workload prediction derived from historical observations, as discussed in next section. The feedback controller, which typically operates at a shorter time-scale, tunes the utilization targets based on the error between the performance target and the measured performance.

The application controller can work in three different modes: feedback only, feed-forward only, and feedback plus feed-forward. In feedback-only mode, the feedback controller reacts to the performance error due to the fluctuation of the workload or other disturbances. There is delay between the time when performance goes above a threshold and the time when the controller takes action since it only reacts to the performance error. In feed-forward-only mode, the model-based predictive control proactively tunes the resource allocation based on predictions of workload and performance in the next interval. It works well if the models are accurate enough. However, inaccuracy in either performance or workload prediction can cause problems. Integration of feedback and feed-forward controllers provides a more robust solution than that with either feedback or feed-forward alone. The feed-forward controller complements the reactive feed-back controller by immediately adjusting the utilization targets upon detection of workload or policy changes. On the other hand, models have inherent limitations in their accuracy, and the workload may have unpredictable variations. Both can lead to bias in the prediction of utilization targets, which can be corrected by the feedback loop. With appropriate settings of the two controllers, the system can make reasonable trade-offs between responsiveness and stability over the entire operation range.

TABLE I: Notation for performance modeling

M	number of tiers (e. g., Web, APP, DB)
N	number of transaction types (e. g., Browse, Bid)
r_{cpu}	average resident time on CPU resources
r_{others}	average resident time on non-CPU resources
r	average user request response time
λn	average request rate of transactions type n
λ	aggregate request rate of all transaction types
β_{nm}	average CPU demand of transaction type n at tier m
α_n	average service time of non-CPU resources of transaction type n
e_m	CPU entitlement that is allocated to the virtual server at tier m
c_m	CPU consumption of the virtual server at tier m
u_m	CPU utilization of the virtual server at tier m

We briefly discuss the design of the feedback controller, leaving development of the performance model and the design of the feed-forward controller to the next section.

As shown in Figure 4, the feedback controller is driven by the error between the measured mean response time r and the response time reference r^{ref}. The output of the feedback controller provides adjustments ΔU to the utilization targets U^{mdl} determined by the feed-forward controller. In our implementations, we adopted an integral controller as follows:

$$\Delta U(j) = \Delta U(j-1) + G^{FB}\frac{r^{ref} - r(j-1)}{r^{ref}} . \qquad (2)$$

Here j represents the time index for the current interval, and G^{FB} is the gain in the controller. Note that because the utilization target should fall into the range of $[0, 1]$, the response time error is normalized by its reference r^{ref}.

III. Performance Model and Proactive Control

Queuing models have previously been applied to model application-level performance on physical servers. However, a virtual machine differs from a physical server in that its effective capacity varies with dynamic resource allocation (e. g., CPU shares). This distinguishes VM performance models from traditional queuing models, because model parameters such as service times can change significantly in virtualized servers if the effective server capacity is modified. As a result, performance models for physical servers cannot be directly applied to virtualized server environments. In this section, we extend a traditional queuing model to characterize the performance of multi-tier applications hosted on virtual machines. We show how the application performance can be modeled as a function of workload transaction mix and resource allocation to the application components. We then describe how to identify the model parameters using non-intrusive measurement and linear regression analysis. Finally we discuss how to use the proposed model to determine the CPU demand for virtual machines hosting an application in order to meet the response time target of the application, which is what the feed-forward controller does in the AppRAISE architecture.

We introduce many variables and parameters for performance modeling. For convenience, Table I summarizes the notation to be used this section.

A. Multi-tier Application

Modern Internet and e-business applications are usually structured into multiple logical tiers. Each tier provides certain

functionality to the preceding tier and uses the functionality provided by its successor to carry out its part of the overall request processing. In our model, we consider a general multi-tier application consisting of M tiers. We assume that each tier T_m runs on a separate virtual machine V_m $(m = 1, \ldots, M)$.

Multi-tier applications typically support a number of transaction types. For example, an online auction application has transaction types such as login, browse or bid. At any given time, a typical application workload is composed of a mix of different transaction types. In most cases, the transaction types have different demands for resources. For example, bid transactions in an auction site typically require more CPU time than browse transactions.

Our work considers a workload with N transaction types. Unlike workload models that assume a stationary transaction mix, and hence use an aggregate request rate to characterize the workload, we define the intensity of the workload as a vector $(\lambda_1, \ldots, \lambda_N)$, where λ_n is the average request rate of transaction type n during one time period. We use this vector model to capture the request rate per transaction type and thus to characterize non-stationary transaction mixes. This is important because transaction mixes in real production systems change over time [1]. We also define the aggregate rate of the transaction as

$$\lambda = \sum_{n=1}^{N} \lambda_n \ . \tag{3}$$

B. Performance Modeling

Multiple system resources (CPU, memory, disk, etc.) can become bottlenecks during operation. In this work, we focus on CPU resources, since business logic processing in multi-tier applications is often the bottleneck, and workloads are often processor intensive. In addition, adjusting a virtual server's CPU capacity is the most mature resource control knob available in today's virtualization technology. In the following discussion, we assume that CPU is the single bottleneck resource and that CPU is the only resource to be dynamically allocated among the virtual machines.

End-to-end response time of a multi-tier application can be calculated by aggregating the resident times over all resources (e. g., CPU, disk, network) across all tiers (e. g., web, application, and database tier). We break down the response time into two parts: the resident time on CPU resources and the resident time on non-CPU resources. We count both service and queuing times for CPU. We assume that non-CPU resources are adequately provisioned and hence the effect of contention for these resources on the response time (i. e., the queuing delay) is negligible.

Processor sharing (PS) approximates round-robin scheduling with small quantum size and negligible overhead, and is representative of scheduling policies in commodity operating systems [11]. Given the assumption that a Poisson process is a good approximation of request arrivals, we model CPU as an M/G/1/PS queue. According to queuing theory, the total CPU resident time by all the requests served in tier m is represented by $u_m/(1-u_m)$, where u_m is the CPU utilization of tier m. Then the mean CPU resident time of all the transactions with aggregate rate λ can be described as follows,

$$r_{cpu} = \frac{1}{\lambda} \sum_{m=1}^{M} \frac{u_m}{1-u_m} \ . \tag{4}$$

By definition, CPU utilization in Equation (4) is the ratio between the virtual machine's CPU consumption and its effective CPU capacity. Note that CPU consumption is independent of the CPU entitlement, as long as it does not exceed the entitlement. However, the utilization can change when the CPU entitlement is changed even for constant workloads. This is different from physical server environments where CPU capacity is not dynamically modified and hence CPU utilization for the same workloads remains the same. Let c_m and e_m denote respectively the CPU consumption and entitlement at tier m. The response time can then be represented in terms of CPU consumption and entitlement as follows, by replacing u_m with c_m/e_m in (4),

$$r_{cpu} = \frac{1}{\lambda} \sum_{m=1}^{M} \frac{c_m}{e_m - c_m} \ . \tag{5}$$

Given a workload with transaction mix $\lambda_1, \ldots, \lambda_N$, we can estimate the CPU consumption of the workload based on the following observation. The resource demands of different transaction types are usually different, but the resource demand of a single transaction type is relatively fixed irrespective of the transaction mix of the workload and CPU entitlement of the virtual machines, since each transaction type usually has a relatively fixed code execution path and hence a stable resource demand [1]. In typical multi-tier applications, each transaction issued by the users can trigger differing numbers of requests at different tiers. However, for a given transaction type, the number of requests generated in each tier is proportional to the request rate of this transaction type issued by users. Hence the CPU consumption c_m at tier m can be defined as a linear function of the transaction mix as follows,

$$c_m = \sum_{n=1}^{N} \beta_{nm} \lambda_n \ , \tag{6}$$

where β_{nm} is the average CPU demand of transaction type n at tier m. We call (6) the *utilization model*.

Replacing c_m in (5) by (6), the resident time on CPU has the following form

$$r_{cpu} = \frac{1}{\lambda} \sum_{m=1}^{M} \frac{\sum_{n=1}^{N} \beta_{nm} \lambda_n}{e_m - \sum_{n=1}^{N} \beta_{nm} \lambda_n} \ . \tag{7}$$

Let r_{others} denote the mean resident time spent on all non-CPU resources, and α_n represent service times of transaction type n on all non-CPU resources of all tiers on the execution path of that transaction type. Then the mean resident time on non-CPU resources can be approximated by the weighted sum of each transaction type's service time,

$$r_{others} = \frac{1}{\lambda} \sum_{n=1}^{N} \alpha_n \lambda_n \ . \tag{8}$$

Let r denote the end-to-end mean response time of the application. Combining (7) and (8), the mean response time can be represented by a function of the transaction mix

and the resource entitlement as follows, which we call the *performance model*:

$$r = r_{cpu} + r_{others} = \frac{1}{\lambda}\left(\sum_{m=1}^{M} \frac{\sum_{n=1}^{N} \beta_{nm}\lambda_n}{e_m - \sum_{n=1}^{N} \beta_{nm}\lambda_n} + \sum_{n=1}^{N} \alpha_n \lambda_n\right) . \quad (9)$$

Note that the CPU demand β_{nm} for each transaction does not vary with the CPU entitlement e_m. The CPU entitlement has no effect on the service time of non-CPU resources either, assuming that it depends on the capacity of non-CPU resources only. Thus once we obtain α_n and β_{nm} values, the model (9) can be used to predict the response times for given transaction mixes and CPU entitlements as discussed in the next section.

C. Model Parameter Identification

Multi-tier applications from enterprise applications to large e-commerce sites share a crucial characteristic: the relative frequencies of transaction types in their workloads are non-stationary [1]. That is, the frequencies of the transaction types vary over time. This property makes possible a non-intrusive approach to identify the parameters of the performance model.

We divide time into non-overlapping intervals and measure the mean response time r, transaction mix $\lambda_1,\ldots,\lambda_N$, the CPU entitlement e_m and CPU consumptions c_m $(m = 1,\ldots,M)$ of the tiers for each time interval. Since the transaction mix changes over time in a non-stationary way, and the relationship between c_m and λ_n are linear, β_{nm} can be estimated through linear regression using (6) over multiple measurement intervals. The model (9) can then be used to obtain α_n using linear regression analysis.

D. Proactive Control

In the architecture shown in Figure 4, the feed-forward controller provides proactive control to meet the specified response time target by dynamically adjusting the CPU utilization targets of virtual machines hosting the application components. This is achieved by first translating the response time target to a utilization target based on the performance model (9).

By combining Equations (4) and (8), we re-formulate the performance model as follows,

$$r = \frac{1}{\lambda}\left(\sum_{m=1}^{M} \frac{u_m}{1-u_m} + \sum_{n=1}^{N} \alpha_n \lambda_n\right) , \quad (10)$$

from which the utilization targets u_m $(m = 1,\ldots,M)$ for each tier can be derived for a given response time target r^{ref} and transaction mix λ_n $(n = 1,\ldots,N)$. Note that β_{nm} are required to derive α_n though they are not used directly in (10).

When the number of tiers is greater than one, we have more than one variable u_m in (9), and the solution becomes indeterminate. In that case, we can define an objective function (e. g., a cost function of CPU capacity allocated to the virtual machines) and cast the problem into an optimization problem with a constraint defined by (10). Other constraints (e. g., $u_m < 80\%$) may also be specified by operator policy.

It is also possible to assume that the utilization targets of all tiers are the same. This does not imply resource entitlements are the same across the tiers, since the resource consumptions of the tiers can be significantly different. In that case, (10) can be solved directly to obtain the utilization target for each tier as follows,

$$u_m^{ref} = \frac{r^{ref}\lambda - \sum_{n=1}^{N} \alpha_n \lambda_n}{M + r^{ref}\lambda - \sum_{n=1}^{N} \alpha_n \lambda_n} . \quad (11)$$

We use this approach in the evaluation experiments discussed in next section.

Note that, the utilization targets necessary to meet the response time threshold also depend on the transaction mix, which is not available until the end of the measurement interval. A number of prediction algorithms such as those proposed to capture the trends or patterns of the workloads [12] can be used for this purpose. In this paper, we use a simple one-step predictor to predict the transaction mix in the current interval from the transaction mix in the previous interval. Integration of our model with more advanced workload prediction techniques is left for future work.

E. Discussion

Our performance model has several desirable features. First, our model can predict performance of multi-tier applications running on virtualized servers with variable CPU entitlements. Once we obtain a set of model parameters, we can use them for different configurations and scenarios. Second, our modeling approach is non-intrusive in the sense that the process of model parameter identification requires no additional instrumentation and the data used in our approach is readily available from standard system and application monitoring. Thus our model parameter identification does not require changes to existing applications and systems for instrumentation purposes and hence can avoid expensive benchmarking. Finally, our model captures the resource demand at per-transaction type and per-resource levels and can be used to derive resource demand for different transaction mixes. Hence this approach can be applied to realistic workloads where both the frequencies of transaction types and request volume rates can change over time.

The performance model also has several limitations. First, although CPU is often the key resource in multi-tier applications, in reality multiple system resources can become bottlenecks. The model presented in this paper currently handles only a single bottleneck resource, and does not capture the utilization of multiple resources. It is possible to enhance our model to include other temporal resources (e. g., network bandwidth) using queuing networks. This will be the subject of future work. Modeling spatial resources such as virtual memory or buffer caches requires different approaches. Another limitation is that we only calculate the mean response time, which can be misleading. We are interested in exploring the enhancement of the current model for handling bursty workloads and estimating probability distributions of response time in our future work. We can use Markov's Inequality to estimate a loose upper-bound for the tail distribution of response time. An alternative approach is percentile regression analysis [13]. Finally, the model assumes that resources are held at exactly one tier at a time as a request proceeds through the system. Thus, our model does not handle applications

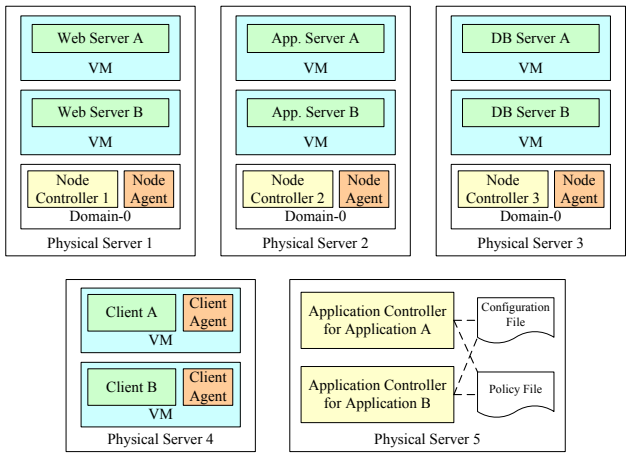


Fig. 5: Testbed design for experiments.

such as video streaming, where a request can hold resources simultaneously at multiple tiers or servers.

IV. Implementation

To evaluate the effectiveness of AppRAISE, we implemented the architecture and algorithms on a testbed that hosted multiple multi-tier applications in a virtualized server environment. Experimental results showed that our proposed models are valid in such an environment and that AppRAISE provides a solution for robust application level performance control for the hosted applications.

A. Testbed

As shown in Figure 5, our testbed consists of five HP ProLiant servers. Each has 2 CPUs, 4 GB of RAM, one Gigabit Ethernet interface card, two local SCSI disks, and a SUSE Linux distribution with a Xen-enabled Linux 2.6 kernel.

Three of the servers were used to host two three-tier applications, called *Application A* and *Application B*, respectively. Both applications were comprised of a front-end Apache Web server (v. 2.2.9), a middle-tier JBoss application server (v. 4.0.2), and a back-end MySQL database server (v. 5.0.26). Each application component was run inside one virtual machine. In our experiment, each physical server hosted two virtual machines containing one tier from the two applications.

Inside the management domain (Domain-0) of each physical server, the “Node Agent" exposed APIs for monitoring the host and the virtual machines, and allocating the resources to the virtual machines. To get CPU statistics, the agent collected the hypervisor counters that accumulated the CPU time (or cycles) consumed by each domain. These counters were sampled at fixed intervals, yielding a sensor metric of the CPU consumption. The Xen hypervisor also provides interfaces that allow runtime adjustment of scheduling parameters such as CPU entitlement to each domain. In our experiments, the Xen Credit Scheduler was employed in the *capped mode*, which assures a straightforward guarantee on the CPU entitlement to domains while the maximum capacity available to the virtual machines is also capped by the entitlement.

The workload generators of the two applications were run in the fourth server. The “Client Agent" collected data on the workload, including the URL and the end-to-end response time of each transaction. The agent also exposed APIs for the controllers to access the application data. In production environments, the performance data may not be measured directly from the clients but a proxy can be used for that purpose in the front end.

The fifth server hosted the application controllers, each managing one application. Each physical server was managed by a node controller, running inside Domain-0. Communication among the agents and the controllers was implemented using the XML-RPC protocol.

We did not implement the configuration service and the policy engine shown in Figure 2. Instead, XML files were used to configure the topology (i. e., the placement of VMs), define the applications (i. e., their component VMs, performance metrics and targets), and configure the controllers (i. e., control and sampling intervals, parameter values).

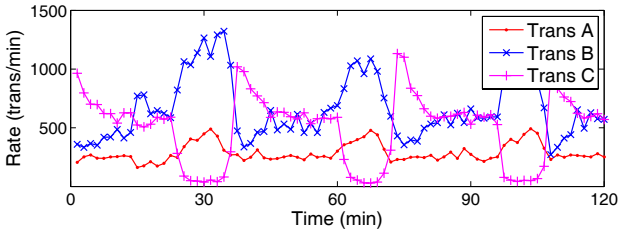
B. Workload

Our test application is a *modified* version of the Rice University Bidding System (RUBiS) [14], [15], which is an online auction benchmark with 22 transaction types, such as browsing for items and viewing user information. In our testbed, the EJB_BMP version of the application server is deployed, which has higher CPU demand than other implementations. We employ a RUBiS database with 500000 users, 55000 items, 550000 comments, and 3.7 million bids.

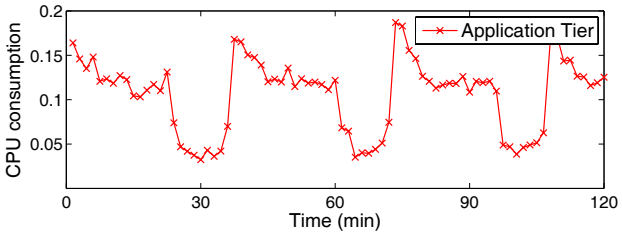
Like other benchmark workload generators, the default RUBiS client produces stationary workloads, that is, the relative frequencies of the different transaction types remain constant over time. To emulate real applications with highly non-stationary transaction mixes, we used a custom workload generator that allowed us to replay transaction traces collected from production systems.

The transaction traces that we used in our experiments were collected from a globally-distributed business-critical internal HP application called “VDR". The transactions of this enterprise application involved case and data management for both external customers and HP users. The RUBiS application was used in our experiments to emulate the VDR application as follows. The RUBiS transactions were first ranked based on their popularity in the workload generated by the default RUBiS generator. Synthetic traces were then created by replacing VDR transactions in the original traces with the RUBiS transaction of the same popularity rank. These traces were then fed to the custom workload generator, which sent out the requests to the RUBiS application with exponential inter-arrival times. More information on the workload can be found in [1].

To illustrate the non-stationarity of the synthetic workload of RUBiS that mimicked the VDR application, Figure 6a shows the individual intensities plotted versus time for the top three transaction types while the aggregate arrival rate of all the transactions was hold constant at 2,400 transactions per minute or 40 transactions per second. In our tests, the transaction mix data were collected every 90 seconds. In this experiment, 100% of CPU capacity was available to the application. The time series show fluctuation in the rates of the



(a) Time-varying frequencies of transaction types



(b) Time-varying CPU demand in the application tier

Fig. 6: Non-stationarity of the workload.

individual transaction types. Moreover, the relative intensity of the transactions significantly changed over time. The transaction types also consume different CPU resources. Figure 6b shows the aggregate CPU consumption of the transactions on the application tier. Even with constant aggregate rate, the workload has time-varying demand on CPU.

The CPU consumption shown in Figure 6b follows a pattern similar to the intensity of Transaction C that is shown in Figure 6a. This is mainly due to the very high frequency and significant CPU demand of that transaction type. In fact, Transaction C accounts for 21% of all the transaction types and is the third most expensive transaction in terms of CPU demand. Transaction A and B are slightly more popular than Transactions C, accounting for 23% and 28% respectively, but they have much lower CPU demand. Two other transactions types, not shown in the figures, consume more CPU than Transaction C on the application tier , but account for only 3% each of the total number of transactions, while all the remaining transactions have very low CPU demand.

C. Control Algorithms for AppRAISE

Figure 7 shows the pseudo code of the algorithms for the feed-forward controller, the feedback controller and the utilization controller utilized in AppRAISE, as described in Section II and Section III.

The three controllers worked continuously, each with their own sampling and control intervals notated as T_{FF}, T_{FB}, and T_{Util}, respectively. It is challenging to determine appropriate values for the sampling intervals for guaranteed convergence of the system with nested loops. However, the rule of thumb is to set $T_{Util} \ll T_{FB} \ll T_{FF}$ so that the three loops can work at different time scales. In that case, the inner loops can be deemed to be in their respective steady states for stability analysis of the outer loops. In our experiments, T_{Util}, T_{FB} and T_{FF} were set to 10, 30 and 90 seconds respectively. These intervals were used to balance the overhead associated with resource actuation and the tracking properties of the controllers

//Feed-forward Control
For the i^{th} control interval {
1) Get transaction mix $\lambda_1(i-1), \ldots, \lambda_N(i-1)$
2) Estimate current transaction mix
$\hat{\lambda}_n(i) = \lambda_n(i-1), \quad n = 1, \ldots, N$
3) Predict utilization target
$U^{mdl} = f(\hat{\lambda}_1(i), \ldots, \hat{\lambda}_N(i), \alpha_1, \ldots, \alpha_N, r^{ref})$
}
// Feedback Control
For the j^{th} control interval {
1) Get mean response time $r(j-1)$
2) Calculate utilization adjustment
$\Delta U(j) = \Delta U(j-1) + G^{FB}\left(r^{ref} - r(j-1)\right)/r^{ref}$
3) Update utilization targets
$U^{ref}(j) = U^{mdl} + \Delta U(j)$
with $U^{ref}(j) = u_1^{ref}(j) = \ldots = u_M^{ref}(j)$
}
// Utilization Control
For the k^{th} control interval of virtual server v of tier m {
1) Get consumption $c_v(k-1)$ and entitlement $e_v(k-1)$
2) Calculate new entitlement
$e_v(k) = e_v(k-1) - \gamma c_v(k-1)\left(u_v^{ref} - u_v(k-1)\right)/u_v^{ref}$
with $u_v(k) = c_v(k-1)/e_v(k-1)$
}

Fig. 7: Pseudo code of AppRAISE controllers.

as the workload fluctuates. First, in a Xen environment, the response time of the application can suffer if the scheduler parameters are modified too frequently. As Xen matures and improves its efficiency, utilization control can be executed at shorter intervals. Second, since the workload fluctuated significantly, the predictability of the feed-forward controller can be compromised if longer time intervals are used for the predictive controller. We plan to experiment in the future with other intervals to improve control performance.

Two additional parameters were needed for the controllers: the gain for the utilization controller and that for the feedback controller. Note that there is a non-linear relationship between response time and utilization. A larger feedback control gain can cause larger deviation of the response time. On the other hand, a larger gain can help the controller to react more quickly to sharp changes in workload. The gain parameter γ of the utilization controller was set to 1.0, which is in the middle of the stability range. The gain of the feedback controller G^{FB} was 0.1 based on our earlier experience [16].

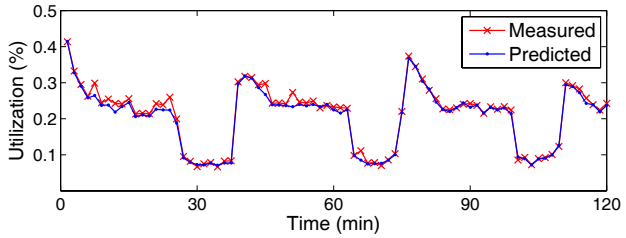
For comparison, we also evaluated scenarios in which one or two of the three controllers were disabled, but the parameters of the controllers were otherwise kept the same. This will be described in more detail in the next section.

V. Evaluation

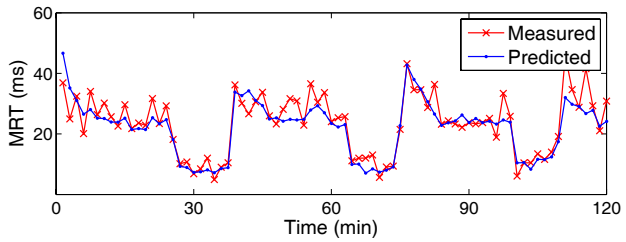
We ran extensive experiments to evaluate AppRAISE. In this section, we focus on modeling and dynamic control of the system. By comparing different controller options and control algorithms, we demonstrate how AppRAISE can achieve a reasonable tradeoff between tracking a performance target and reducing resource consumption.

A. Model Parameter Identification and Validation

To validate the transaction-mix-based utilization model (6) and performance model (9), we ran experiments on the



(a) CPU utilization prediction from transaction mix



(b) Performance prediction from transaction mix

Fig. 8: Model validation.

testbed using a standard "training and testing" process. In these experiments, the applications were driven by the traces, and the virtual machines that hosted application components were statically allocated constant shares of CPU capacity. The parameters of the models were identified through linear regression, using the time series of the metrics, including consumption, entitlement, transaction mix, and response time. The robust linear regression tool in MATLAB, which has been shown to be less sensitive to outliers than traditional least-square regression, was used to obtain the model parameters. The models were then validated using data sets collected in other experiments run under different conditions (e. g., using different resource entitlements and/or workload intensity).

Figure 8 shows part of the data collected from an experiment, from which the model parameters including α_n and β_{nm} for all the transactions were identified. The experiment was run for 10 hours, and 100% of the CPU was available to each of the three virtual machines that hosted the three tiers of the application. The model was then validated using the data collected from another experiment, in which only 50% of CPU was available.

Figure 8a shows the validation results where the experimental data were compared with those predicted using models (6) and (9) and exact transaction mix values. Note that only two hours of the time series are shown. In Figure 8a, the time series of the utilization metric of the application tier were compared with those predicted using the utilization model (6). Figure 8b shows the mean response times collected from the same experiment, versus those predicted by the performance model (9). Not surprisingly, the prediction error for response time appears much higher than that for the utilization. To evaluate how accurate the models are, we introduce the normalized error ε that is defined as

$$\varepsilon = \sum_i |\hat{y}_i - y_i| / \sum_i |y_i| \ , \qquad (12)$$

where $\hat{y}_i$ and y_i are the estimated and measured values respectively of the metric y in the i^{th} interval. The normalized errors

TABLE II: Statistics on utilization and performance prediction

CPU caps of tiers (Web, App, DB)	Norm. utilization error (app tier)	Norm. MRT error
(100, 100, 100)	2.6%	9.3%
(90, 90, 90)	4.0%	11.9%
(70, 70, 70)	9.1%	12.8%
(50, 50, 50)	3.1%	14.9%
(40, 40, 40)	4.0%	12.3%
(30, 60, 40)	4.9%	15.1%

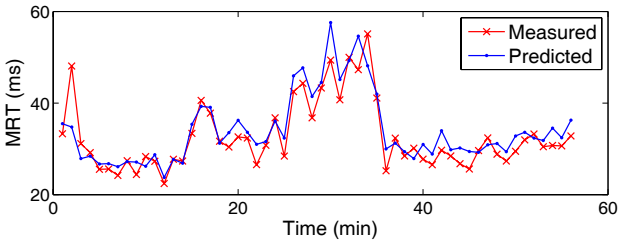


Fig. 9: Response time under high utilization.

for the utilization and response time shown in Figure 8 were 3.1% and 14.9% respectively. From its cumulative distribution, 90 percent of response time prediction errors were actually below 25%.

Table II shows statistics on the prediction errors of the models in a series of experiments when the CPU entitlement to the application was varied. In all six cases, the model parameters, α's and β's, were the same as that derived from the first experiment where 100% of capacity was available to all the three tiers, notated as (100, 100, 100). The models were then tested in the other five cases where the CPU entitlements were varied as in the first column. Overall, the utilization model can predict the utilization level with errors less than 10%. For prediction of MRT, which is more challenging than that of utilization, the errors are no more than 15%.

We note that the trends versus time of both utilization and response time shown in Figure 8 are similar as that of the CPU consumption shown in Figure 6b. For utilization, this is due to the workload intensities being the same in the two experiments. However, the amplitude of the utilization in Figure 8a is almost doubled since only half of CPU capacity was allocated to the application in the experiment. If we decompose CPU resident time into CPU service time and queuing time, the first part is actually proportional to the CPU demand since the resource entitlement was constant. Given the relatively low utilization (25% most of the time), the queuing time only contributed approximately 10% of the total response time. That explains why the response time followed similar trend as that of the CPU demand.

To validate our model under relatively high utilization, we created another trace with transaction mixes different from those of VDR and ran another experiment driven by that trace. Compared with VDR, which had high demand on the application tier, the new workload put more demand on the database tier. In the experiment, the database server utilization varied between 30% and 65% with a mean utilization of around 50%. From queuing theory, the queuing time in this experiment contributes 50% of the resident time for CPU. Figure 9 shows the response time measurement, and those

predicted using the same models as in previous experiments. The normalized prediction error is only 7.7%.

B. Experimental Configuration for Dynamic Control

We focus on the dynamic behavior of the system in this section. We ran two series of experiments. In the first series, only one application was used with the three tiers hosted in the three virtualized servers. In the second series, two applications were run in parallel as shown in Figure 1. All the applications were driven by the VDR traces. For each series, four control options were tested.

Case 1: Fixed utilization targets ("FixedUtil"). In this case, CPU capacity was dynamically allocated to meet the specified utilization target, which was held constant over time. This approach is widely used in production systems. The controller is similar to the functionality offered by some commercial workload management products, such as HP's gWLM [5] and IBM's Enterprise Workload Manager [6]. In our experiment setup, the feedback and feed-forward controllers were disabled so that the utilization controller tracked a fixed utilization target configured through the configuration file.

Case 2: Feedback plus utilization control ("FB+Util"). The feed-forward controller was disabled in this case. The feedback controller was modified to directly tune the utilization targets, rather than adjusting output from the feed-forward controller.

Case 3: Feed-forward plus utilization control ("FF+Util"). In this case, the feedback controller was disabled. With accurate enough performance models and workload predictors, the feed-forward controller should track the performance target upon predictable changes of workload demand faster than the feedback controller does.

Case 4: Integrated control ("FB+FF+Util"). With integrated feedback and feed-forward control, this option should provide fast and stable tracking of the response time target as workload demands change.

In each experiment, time series of the response times, CPU consumption and allocation of each tier of the applications were collected and analyzed. By comparing the results from the four options, we expect to understand the effects of feedback and/or feed-forward control on the resource usage and performance guarantee of the applications, and show how AppRAISE can achieve a reasonable tradeoff between tracking a performance target and reducing the required resources.

C. Experiment Results for a Single Application Running in Three Virtual Machines

In the first series of experiments, each of the four experiments ran 90 minutes with a mean arrival rate of 30 transactions/second. For all four cases, the sampling intervals of the utilization controller and feedback controller were 10 and 30 seconds respectively. In the case "FF+Util", the feed-forward controller updated the utilization targets every 30 seconds, based on the workload history in the past 30 seconds. In the case "FB+FF+Util", the feed-forward controller reset the utilization targets every 90 seconds, but only based on the history of the past 30 seconds. In all the cases, the minimum CPU share of each virtual machine is set to 20% of a CPU.

TABLE III: Steady-state performance for one application

Control cases	MRT (ms)			Mean entitlement (shares of CPUs)			
	Mean	Std	Below target	Web tier	App tier	DB tier	Total
FixedUtil	103.4	60.4	23%	20	21.0	22.6	63.6
FF+Util	33.9	14.6	76%	20	35.5	32.1	87.6
FB+Util	38.5	19	59%	20	35.3	32.1	87.5
FB+FF+Util	37.1	16	60%	20	35.0	31.7	86.7

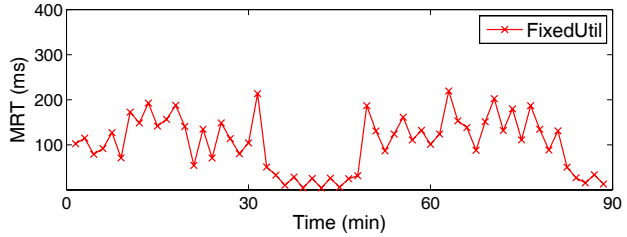
One question is how to set up the utilization target in the first case, and the response time targets in the other three cases. In production environments, the operators of the systems usually do not know how to set up the utilization target of workload managers, and often use the default settings provided by the manufactures, for instance, 75% in the HP gWLM. Previous analysis has also shown that it's highly undesirable for interactive applications to operate above 70% [1]. An ideal utilization target depends on the performance target of the application, how bursty the workload demand could be, and the capacity of the system available to the application. In our experiments, we set the utilization target to 50% in the "FixedUtil" case, without considering the properties of the applications and the systems.

Setting the response time target is a more challenging issue. It has been found that the response time requirement for real user-interactive Internet services is very demanding. For example, Amazon found that every additional 100ms of latency cost it 1% loss in sales; Google noticed that an extra 0.5 seconds in search page generation time reduced the access traffic by 20% [17]. In our experiments, most of the transactions of our workload have low CPU demands. We thus set the response time target to 40ms. Given the smaller testbed and simpler applications than that in production environments like Google and Amazon, the response time target we set should be reasonable.

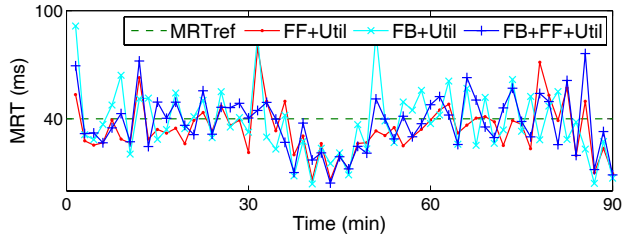
Table III shows the statistics of the experimental results in the four cases. For performance, we show the mean and standard deviation of the MRT (mean response time as measured every 90 seconds) samples, and the percentage of the MRT samples with values less than the target. The table also shows the mean CPU entitlement to each of the three tiers, and the total entitlement to the application. Overall, best performance was achieved in "FF+Util" case: the lowest mean response time and comparable total resource entitlement.

To better understand the results, we show in Figure 10 the time series for MRT and the CPU entitlement in the four cases. Detailed discussion follows based on the data in the table and the information in the time series.

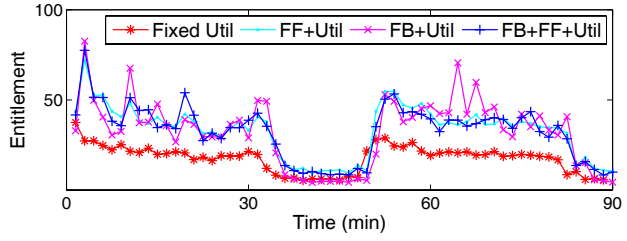
"FF+Util" achieves best tradeoff between performance and resource entitlement. Compared with the other three cases, the least CPU was allocated to the application under control of the utilization controller ("FixedUtil"), but the application experienced the worst response time. Although the time series for response time in Figure 10a follows a similar pattern as for the other cases in Figure 10b, the magnitude of the samples is much higher than the others. As argued before, in general the highest utilization threshold to maintain a given response time reference depends on many factors. It



(a) MRT of the application with utilization controller only



(b) MRT of the application under control of FF, FB or FB+FF



(c) CPU entitlement to the application tier in the four scenarios

Fig. 10: Transient effect of the controllers.

may be derived with good models and workload predictor. In our case, a utilization target of about 25% is actually needed for the 40ms response time target.

Among the other three cases, the application under control of "FF+Util" has the lowest response time, in terms of both mean and standard deviation. "FB+Util" has the largest mean response time, and also the largest variance of MRT. Although the total CPU shares allocated to the applications are very close to one another in the three cases, the tradeoff between CPU entitlement and response time is still maintained in the three cases as shown in the table.

The table shows that the mean response times in the three cases with FB and/or FF controllers are all below the target 40ms. This is mainly because the response time is much lower than the target, as shown in Figure 10b for about 10 minutes in the middle of the experiments and for a few intervals at the end. During those time periods, the workload had very light CPU demand, but the CPU capacity was over-provisioned because the virtual machines were maintained at 20 shares even when the requested entitlement was lower.

If we remove the samples in the time periods when CPU was significantly over-provisioned, the mean response times for the three cases become 39ms, 43ms, and 42ms, and the standard deviations become 13ms, 16ms, 13ms, respectively. That means the response time target is still tracked the best in the case "FF+Util". In the "FB+Util" case, the controller is too aggressive in compensating for the MRT error. Oscillation exists in the time series for MRT in the case "FB+Util", as shown in Figure 10b. This also explains why its response time metric has the largest variance among the three cases. Using a smaller control gain should alleviate the issue.

Similar results were observed in a second experiment driven by the traces utilized in the modeling experiment for data shown in Figure 9. In that experiment, the mean utilization of the application server was 40%, and that of database server was 48%. Both were relatively higher than that in the above experiments. In the case "FF+Util", the average MRT was 55ms for a target of 60ms, and 92% of the samples were below the target.

Prediction errors can be compensated by feedback controller. The results in Table III imply that the models are reasonable, and that the model based feed-forward controller can maintain performance at its target level. However, these results do not mean that the feedback controller is useless. In the work presented in [16], we have validated the benefits of feedback control through experiments run on another testbed. In those experiments, the "FF+Util" controller was shown to be sensitive with respect to modeling errors, which can result in over or under provisioning of CPU capacity, and poor response time in the latter case. However, the integrated controllers were shown to be more robust with respect to modeling errors, and maintained the performance at its target even in the presence of prediction errors.

Proactive control can improve the transient process of feedback loop significantly. What is the effect of introducing feed-forward control into the feedback loop? In general, it should help the controller to respond to transient changes more quickly and more smoothly because of the proactive action. From Table III, we know that the variance of response time in the case "FB+FF+Util" is less than that in the case "FB+Util". As seen in Figure 10b, even with the same gain of the feedback controller, the integrated control results in a much smoother response time curve than "FB+Util" does.

To study the problem further, note that in the case "FB+FF+Util", resetting the utilization target by the feed-forward controller moves the operation of the feedback loop closer to the equilibrium. Even with the same feedback gain, which is too aggressive in the case "FB+Util", the integrated controller resulted in less variance. As another example, shown in Figure 10b and 10c, the demand of the workload sharply increased at around the 50^{th} minute. The "FB+Util" controller responded with delay, causing a large overshoot of the response time. With integrated proactive control, as in the "FB+FF+Util" case, the response time was driven closer to the reference with much less overshoot. More examples are shown in [16] to demonstrate the effect of proactive control on transient processes.

To summarize the results shown in Table III and Figure 10, and those presented in [16], our performance model is reasonable and is able to predict the performance based on the transaction history of the workload. Although the feed-forward controller works well given good models, the integrated controller can be more robust. On one side, introducing proactive control into a feedback loop can help the system to respond quickly to changing workload demands. On the other side, introducing feedback to predictive control can compensate for errors in the feed-forward loop due to modeling biases or

TABLE IV: Steady-state performance for two applications. (Application I, 30 transaction/sec, MRT target 40ms)

Control cases	MRT (ms)			Mean allocation (shares of CPUs)
	Mean	Std	Below	Total
FF+Util	34.4	9.8	78%	91.8
FB+Util	38.3	19.1	58%	87.5
FB+FF+Util	36.8	13.3	66%	88.9

TABLE V: Steady-state performance for two applications. (Application II, 20 transaction/sec, MRT target 60ms)

Control cases	MRT (ms)			Mean allocation (shares of CPUs)
	Mean	Std	Below	Total
FF+Util	50.3	19.6	74%	54.4
FB+Util	58.0	23.3	50%	54.1
FB+FF+Util	55.6	22.8	60%	53.8

workload prediction inaccuracies.

D. Experimental Results for Two Applications Running in Six Virtual Machines

The objective of these experiments was to demonstrate that the performance models we proposed are still valid when the physical resources are shared by multiple applications, and that AppRAISE is still able to provide robust performance control. In the experiments, there were two 3-tier applications running in parallel, as shown in Figure 1. They were driven by the same VDR traces, but with different rates and performance targets. Application I, the same as that in the previous section, was driven by the client with a rate of 30 transactions/sec, and its MRT target was set to 40 ms. The rate of Application II was set to 20 transactions/sec, and its MRT target was 60ms, higher than that of Application I. The cases with FF and/or FB controllers were tested under this new scenario, each running for 90 minutes.

The statistics on the two applications are shown in Table IV and V, respectively. We note that the models are still valid, the performance targets of both applications are met, and the "FF+Util" case achieves the lowest response time with slightly higher resource entitlement. By comparing the entries in Table III and Table IV, we can see that Application I has almost the same statistics under the two scenarios, especially in the cases using the "FF+Util" controller.

E. Discussion

A few additional issues about the implementation of AppRAISE and the experiments are worth discussion. First, the prediction of the workload, including the transaction mix and aggregate intensity, is critical to the feed-forward controller. Inappropriate prediction can cause relatively large errors in the output of the feed-forward controller. A better predictor than the "one-step" approach may be adopted, for instance, to exploit the seasonal pattern of the transaction rate using predictors such as those in [12].

Second, the experimental results can be affected by the control parameters, including the control intervals and the feedback controller gains. For instance, shorter sampling periods can maintain the utilization target better when the consumption changes very quickly, but a shorter interval could cause the utilization controller to change the entitlement too fast resulting in unstable behavior. For the feed-forward controller, the performance model works better for larger intervals, but a larger interval also makes the prediction of the workload and the CPU consumption less accurate. The performance overhead of sampling and control actions has to be considered as well. Our future work includes understanding these trade-offs and optimizing the parameters.

VI. Related Work

A. Feedback Control in Computing Systems

Feedback control theory has been applied to solve a number of performance and quality of service (QoS) problems in computing systems in the past several years (see [18], [19] and the references therein for a detailed survey). In these applications, we see two major challenges for appropriate system modeling and effective controller designs: the complex behaviors of computing systems themselves, and the time-varying demands placed on these systems by stochastic and sometimes bursty workloads.

In [9], Wood *et al.* considered black and gray box approaches for managing VMs in a Xen-based testbed. They only considered resource utilization for the black box approach, and added OS and application log information for the gray box approach. They found that the additional information helps to make more effective decisions.

Most early work in this area assumed that the system under control is linear, and that the parameters can be identified offline. However, due to the wide variation of demands observed in computing systems, the parameters or even the structure of the models could change over time. To deal with time-varying parameters in linear models, adaptive control theory has been applied to systems in the context of, for instance, caching services [20] and resource containers [21]. This approach allows the parameters of the model to automatically adapt to changes in operating conditions using online system identification.

In [10], the system's nonlinear and bimodal behavior in different operating regions was studied quantitatively, and controllers that could adapt to the bimodal behavior were developed. In [22], a queuing model based feed-forward predictor was used to capture the non-linear relationship between system response time and resource allocation. They evaluated this idea by controlling the number of web server processes allocated to a class of transactions, whereas we have extended it to the more general-purpose control of CPU allocations for virtualized, multi-tier applications. To manage the increased complexity, we have introduced the conceptual architecture of application and node controllers. We have also incorporated a more sophisticated queuing-theoretic performance model that considers non-stationary transaction mixes.

Other researchers have studied potential conflicts that can arise when running multiple automation policies independently without coordination. In [23], Kephart *et al.* studied the scenario where a performance manager, that dispatches workloads to a set of blades, runs in parallel with a power manager that controls processor frequency. They demonstrated that oscillations can occur in both autonomic managers. The

paper also showed how this problem can be fixed by explicit communication between the two managers. In [24], Heo *et al.* identified the incompatibility between a DVFS adaptation policy and a server on/off policy when they are not coordinated in a server farm, and they presented a co-adaptation approach that can resolve such conflicts. Neither study dealt with resource management in virtualized data centers, as is considered in this article.

In [25], Xu *et al.* presented a two-layered approach to managing resource allocation of virtual containers sharing a server pool in a data center, and evaluated the scheme in a testbed running VMware ESX Server [26]. Their local and global controllers together offer a solution similar to the management approach studied in this paper using fuzzy logic instead of feedback control.

In previous work [7] we provided an integrated capacity and workload management concept for next generation data centers. In that work we did not consider applications that span multiple VMs as the present application controller does. The application and node controllers presented in this work can be easily integrated with those higher level controllers.

Utilization controllers are relatively straightforward to implement. Maintaining the utilization at a target level only requires that the resource consumption is collected in real time and is generally available through local system-level instrumentation. Entitlement actuation can be done inside the server as well, using interfaces available from the virtual machine monitors or hypervisors. Utilization controllers have also been implemented in a few workload management products [5], [6] for dynamic sizing of virtual containers like process groups and virtual partitions of processors, although they generally use standard integral controllers unlike the adaptive controller used in our work.

For most workload managers, a default utilization target, e. g., 75%, is used. However, this default is rarely ideal for individual customer environments when application-level performance guarantees are expected. For instance, depending on how bursty a workload might be, this default utilization level can result in bad application-level performance or over provisioning of resources. It remains challenging for operators to determine the appropriate values for a given application. AppRAISE addresses this issue by automatically determining the appropriate utilization target values for virtual machines

B. Performance Models

In [3], Stewart & Shen used finer-grained resource consumption profiles of components within an application, in conjunction with a queuing model to guide component placement decisions. Urgaonkar *et al.* employed a closed network of queues to model multi-tier applications under processor sharing (PS) scheduling in [2]. They assumed that service times and think times are exponentially distributed so that mean value analysis can be used to solve the model. In [27], Bennani & Menasce demonstrated how analytic performance models can be used to design controllers for the live migration of servers. They used a combination of a multi-class open queuing network model to track transactional workloads and a closed queuing network model, similar to that in [2], to model batch workloads.

Transaction mix models are a new approach to modeling performance in transaction-oriented applications, e.g., modern enterprise applications [1]. Important attractions of transaction mix models for our present purposes are that they are simple, applicable to open network models, readily able to compose with queuing-theoretic extensions, and easy to calibrate using only lightweight, passive measurements that are routinely collected in today's production environments. An extensive evaluation of transaction mix models has shown that they yield remarkably accurate application-level performance predictions in real production applications and in a wide range of challenging lab test scenarios [1]. Transaction mix models leverage naturally-occurring workload irregularities for model calibration, thereby circumventing the need for controlled benchmarking or invasive instrumentation.

The performance models described in Section III combine transaction mix models with novel queuing-theoretic extensions. Our approach is noteworthy in several respects:

- It explicitly considers the transaction mix, recognizing that different types of transactions may incur very different resource consumption rates. Since real production work-loads have time-varying transaction mixes, incorporating transaction mix into the model enhances accuracy.
- Our performance model is relatively straightforward to apply. It uses open-queuing results, with a simple formula for queuing time that applies to both M/M/1 and M/G/1/PS queues. Simplicity allows for quick computation (in comparison to the closed-queuing network approach); it also allows us to invert the equation easily to solve for the target utilization of the application components.

VII. Conclusions and Future Work

In this paper we presented AppRAISE, a distributed management system for application performance control and dynamic resource allocation in virtualized server environments. We evaluated the queueing models for applications hosted on virtualized servers, and used the models for prediction of resource demand to meet application-level performance requirement based on workload transaction-mix history. Empirical results showed that AppRAISE integrates the benefits of both feedback and feed-forward control techniques, and provides a reasonable trade-off between performance guarantee and resource efficiency for applications under control.

We are continuing to study the performance models for virtualized servers for more complicated cases and different virtualization technologies. As future work, further improvements could be achieved through more robust and accurate prediction for the workload, online modeling, and parameter optimization.

References

[1] C. Stewart, T. Kelly, and A. Zhang, "Exploiting nonstationarity for performance prediction," *ACM SIGOPS Operating Syst. Rev.*, vol. 41, no. 3, 2007.

[2] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi, "An analytical model for multi-tier internet services and its applications," *ACM SIGMETRICS Performance Evaluation Rev.*, vol. 33, no. 1, pp. 291-302, 2005.

[3] C. Stewart and K. Shen, "Performance modeling and system management for multi-component online services," in *Proc. 2nd USENIX Symp. Netw. Syst. Design Implementation (NSDI)*, Boston MA, USA, 2005, pp. 71-84.

[4] X. Liu, J. Heo, and L. Sha, "Modeling 3-tiered web applications," in *Proc. 13th Annual Meeting IEEE Int. Symp. Modeling, Analysis, Simulation Computer Telecommun. Syst. (MASCOTS)*, Atlanta, Georgia, USA, 2005, pp. 307-310.

[5] "HP Global Workload Manager (gWLM)." [Online]. Available: http://mslweb.rsn.hp.com/gwlm/index.html.

[6] "IBM Enterprise Workload Manager." [Online]. Available: http://www-03.ibm.com/servers/eserver/zseries/zos/ewlm.

[7] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova, "1000 islands: integrated capacity and workload management for the next generation data center," in *Proc. 5th IEEE Int. Conf. Autonomic Comput. (ICAC'08)*, Chicago, IL, USA, June 2008.

[8] J. Rolia, L. Cherkasova, M. Arlitt, and A. Andrzejak, "A capacity management service for resource pools," in *Proc. 5th International Workshop Software Performance (WOSP)*, Palma, Illes Balears, Spain, 2005, pp. 229-237.

[9] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. 4th USENIX Symp. Netw. Syst. Design Implementation*, Cambridge, MA, USA, Apr. 2007, pp. 229-242.

[10] Z. Wang, X. Zhu, and S. Singhal, "Utilization and SLO-based control for dynamic sizing of resource partitions," in *Proc. 16th IFIP/IEEE Int. Workshop Distributed Syst.: Operations Management (DSOM)*, Barcelona, Spain, 2005, pp. 133-144.

[11] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Upper Saddle River, NJ: Prentice-Hall, Inc., 1984.

[12] W. Xu, X. Zhu, S. Singhal, and Z. Wang, "Predictive control for dynamic resource allocation in enterprise data centers," in *Proc. IEEE/IFIP Netw. Operations Management Symp. (NOMS)*, Apr. 2006.

[13] P. Bodik, C. Sutton, A. Fox, D. Patterson, and M. Jordan, "Response-time modeling for resource allocation and energy-informed SLAS," in *Workshop Statistical Learning Techniques Solving Syst. Problems (MLSys)*, Whistler, Canada, 2007.

[14] C. Amza, A. Chanda, A. L. Cox, S. Elnikety, R. Gil, K. Rajamani, and W. Zwaenepoel, "Specification and implementation of dynamic web site benchmarks," in *Proc. 5th IEEE Workshop Workload Characterization (WWC)*, Austin, TX, USA, 2002, pp. 3-13.

[15] "Rubis rice university bidding system." [Online]. Available: http://www.cs.rice.edu/CS/Systems/DynaServer/rubis.

[16] Z. Wang, X. Liu, A. Zhang, C. Stewart, X. Zhu, and T. Kelly, "Autoparam: automated control of application-level performance in virtualized server environments," in *Proc. 2nd IEEE Int. Workshop Feedback Control Implementation Design Computing Syst. Netw. (FeBID)*, Munich, Germany, 2007.

[17] G. Linden, "Make data useful." [Online]. Available: http://www.scribd.com/doc/4970486/Make-Data-Useful-by-Greg-Linden-Amazoncom, 2006.

[18] G. F. Franklin, D. J. Powell, and M. L. Workman, *Digital Control of Dynamic Systems*. Upper Saddle River, NJ: Prentice-Hall, Inc., 1997.

[19] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. New York: John Wiley & Sons, 2004.

[20] Y. Lu, T. Abdelzaher, C. Lu, and G. Tao, "An adaptive control framework for QoS guarantees and its application to differentiated caching services," in *Proc. 10th IEEE Int. Workshop Quality Service*, 2002, pp. 23-32.

[21] X. Liu, X. Zhu, S. Singhal, and M. Arlitt, "Adaptive entitlement control of resource containers on shared servers," in *Proc. 9th IFIP/IEEE International Symp. Integrated Netw. Management*, Nice, France, 2005, pp. 163-176.

[22] L. Sha, X. Liu, Y. Lu, and T. F. Abdelzaher, "Queueing model based network server performance control," in *Proc. 23rd IEEE Int. Real-Time Syst. Symp.*, Austin, Texas, USA, 2002.

[23] J. O. Kephart, H. Chan, R. Das, D. Levine, G. Tesauro, F. Rawson, and C. Lefurgy, "Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs," in *Proc. 4th IEEE Int. Conf. Autonomic Computing (ICAC)*, Washington, DC, USA, June 2007.

[24] J. Heo, D. Henriksson, X. Liu, and T. Abdelzaher, "Integrating adaptive components: An emerging challenge in performance-adaptive systems and a server farm case-study," in *Proc. 28th IEEE Int. Real-Time Syst. Symp. (RTSS)*, 2007.

[25] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, "Autonomic resource management in virtualized data centers using fuzzy logic-based approaches," *Cluster Computing J.*, vol. 11, no. 3, pp. 213-227, 2008.

[26] "VMware." [Online]. Available: http://www.vmware.com/.

[27] M. N. Bennani and D. A. Menasce, "Resource allocation for autonomic data centers using analytic performance models," in *Proc. ICAC '05: 2nd Int. Conf. Automatic Computing*, Washington, DC, pp. 229-240.

Zhikui Wang is a senior research scientist in Sustainable IT Ecosystem Lab of HP Labs, Palo Alto. He is generally interested in application of control theory and optimization techniques to IT systems. In HP Labs, his main research areas are power and cooling management in servers and data centers, workload management and application performance control. He received his Ph.D. degree from UCLA, majored in Control Systems/Electrical Engineering. Dr. Wang co-authored more than thirty referred papers in journals and conferences, has ten more pending patent applications. He is a member of IEEE and ASME.

Yuan Chen is a senior researcher at HP Labs in Palo Alto, CA. His general research interests are in the design, implementation and management of distributed and enterprise systems. Yuan received a B.S. from University of Science and Technology of China, a M.S. from Chinese Academy of Sciences, and a Ph.D. from Georgia Institute of Technology, all in Computer Science. Yuan's current research has been in the area of IT service management with a focus on efficient SLA and energy management. His past work includes policy-based change management, adaptive distributed systems and middleware, publish-subscribe systems, Internet storage systems, parallel data mining algorithms, and constraint programming. Yuan has published over 30 technical papers in leading conferences and journals in these fields. He is a member of IEEE.

Daniel Gmach is a post-doctoral researcher at HP Labs, Palo Alto. Before that he was a Ph.D. student at the database group of the Technische Universität München where he graduated in 2009. He studied computer science at the University of Passau. His current research interests are in adaptive resource pool management of virtualized enterprise data centers, performance measurement and monitoring, hosting large-scale enterprise applications, database systems, and software engineering principles.

Sharad Singhal is a Distinguished Technologist at Hewlett Packard Laboratories, where he has led teams that have developed techniques for monitoring and managing service level agreements; methods for controlling service quality in multi-tier applications, resource allocation and assignment algorithms; as well as architectures for management of large-scale data centers. His current research interests include application of control theory to computing, policy-based system and services management, and large-scale data center management architectures.

Prior to HP, Sharad spent two years at Bell Laboratories where he worked on speech coding, and 12 years at Bellcore (now Telecordia) where he both conducted and managed research in a number of areas including speech and video processing, neural networks, middleware and personal communications services. Sharad obtained his B. Tech degree from the Indian Institute of Technology, Kanpur, and his MS. and Ph.D. degrees from Yale University. He is a member of the IEEE and the Acoustical Society of America.

Brian J. Watson is a research scientist in Hewlett-Packard's Sustainable IT Ecosystem Laboratory. His diverse career has spanned computer science, aerospace engineering, and physics. His current research interests include machine learning techniques and their application to control systems. Other interests include travel, running, and learning to fly. He is a member of IEEE.

Wilson Rivera is an Associate Professor at the University of Puerto Rico Mayaguez Campus (UPRM). He received his PhD degree in Computational Engineering from Mississippi State University. He leads the Parallel and Distributed Computing Laboratory (PDCLab) at UPRM. His current funded projects address research problems in the area of distributed systems including automated Grid deployment, dynamic resource management and parallel performance. Dr. Rivera is also executive director for the Institute for Computing and Informatics Studies at UPRM.

Xiaoyun Zhu is a Staff Engineer at VMware, Inc., specializing in resource management for virtualized data centers and cloud computing environments. Her general interests are in applying control theory, optimization, algorithms, statistical analysis and simulation to IT systems and services management. She received her Ph.D. in Electrical Engineering from California Institute of Technology in 2000, and her dual B.S. in Automation and Applied Mathematics from Tsinghua University in 1994. From 2000 to 2008, she worked as a senior research scientist at Hewlett Packard Labs. Dr. Zhu has co-authored over 40 refereed papers in journals and conference proceedings. She holds 6 patents and has over 20 pending patent applications. She has been a program committee member for a number of leading management conferences, including IM, NOMS, DSOM, ICAC, and MASCOTS, and an associated editor for the American Control Conference. She is a member of ACM.

Chris D. Hyser is a senior researcher at Hewlett Packard Labs based in Rochester, New York.