Adaptive Virtual Resource Management with Fuzzy Model Predictive Control

Lixi Wang*, Jing Xu†, Ming Zhao*, Jose Fortes†
* Florida International University, {lwang007, mzhao}@fiu.edu
† University of Florida, {jxu, fortes}@ufl.edu

ABSTRACT

Virtualized systems such as utility datacenters and clouds are emerging as important new computing platforms with great potential to conveniently deliver computing across the Internet and efficiently utilize resources consolidated via virtualization. Resource management in virtualized systems remains a key challenge because of their intrinsically dynamic and complex nature, where the applications have dynamically changing workloads and virtual machines (VMs) compete for the shared resources in a convolved manner. To address this challenge, this paper proposes a new resource management approach that can effectively capture the nonlinear behaviors in VM resource usages through fuzzy modeling and quickly adapt to the changes in the virtualized system through predictive control. The resulting fuzzy-model-predictive-control (FMPC) approach is capable of optimizing the VM-to-resource allocations according to high-level service differentiation or revenue maximization objectives. A prototype of this proposed approach was implemented for Xen-based VM systems and evaluated using a typical online transaction benchmark (RUBiS). The results demonstrate that the proposed approach can efficiently allocate CPU resource to single or multiple VMs to achieve application- or system-level performance objective.

1. INTRODUCTION

Virtualized systems such as utility datacenters [27] and clouds [28][29] are emerging as promising new platforms that can significantly improve how resources are provisioned to applications and how computing is delivered to users. One the one hand, applications can be conveniently deployed via virtual machines (VMs) without being tied to any specific physical machine or constrained by any specific set of resources. On the other hand, resources can be consolidated and multiplexed across VM-hosted applications to increase utilization and reduce cost. The fundamental goal for resource management in such systems is that resources should be automatically and dynamically allocated to the applications' VMs according to application-level objectives (e.g., QoS—Quality of Service) and system-level objectives (e.g., service differentiation, revenue maximization).

In order to reach the above goal, resource management in virtualized systems needs to address the challenges raised by the intrinsically dynamic and complex resource usage behaviors in such systems. For example, when an application's workload changes over time in intensity and composition of requests, its VM's demands of different types of resources also change accordingly. As applications are consolidated to the same physical hosts via VMs, they also compete for the shared resources and interfere with each other. As a result, one application's performance depends on not only its own VM's resource usage but also others' behaviors. Even if the application workloads stay relatively steady, service-level objectives (SLOs) may change over time and as a result resources might need to be reallocated.

This paper proposes a new Fuzzy Model Predictive Control (FMPC) based approach to address these challenges in resource management. This approach is architected to answer two key questions: The first one asks how to accurately capture the complex relationship between resource allocation and application performance. The second asks how to adaptively optimize the VM resource allocation as changes occur dynamically in the system. Specifically in the approach described in this paper, a fuzzy-logic based modeling method is employed to learn the relationship between VM resource allocation and application performance, which can efficiently capture complex system behaviors without requiring any a priori knowledge. Then a predictive controller uses this model to predict the resource demand for all VMs and take the resource control actions that enable the system to quickly reach its optimization objective. These two phases work in a closed-loop manner where the model is constructed and updated online and resource allocations are adjusted dynamically in order to adapt to the changes in the system in a timely manner.

This proposed approach was prototyped on Xen-based VM environments and evaluated using a typical online transaction benchmark (RUBiS [14]). The results demonstrate that it can accurately estimate the resource demand for a VM running dynamically changing workload and quickly achieve the desired QoS target. The results also show that more complex behaviors of resource competing VMs can also be captured by the proposed approach and the system-level objective can be quickly achieved and sustained in such a scenario. Compared to a typical linear model based MPC approach, the FMPC approach can obtain 5% better overall QoS as well as faster adaption to the changes.

The rest of this paper is organized as follows. Section 2 describes the background and motivation. Section 3 discusses the detailed design and implementation of the proposed approach and Section 4 presents an experimental evaluation. Section 5 examines the related work and Section 6 concludes this paper.

2. BACKGROUND AND MOTIVATION

2.1 Adaptive Virtual Resource Management

Emerging virtualized systems such as utility datacenters and clouds promise to be important new computing platforms where applications could be executed efficiently and resources could be utilized efficiently. A key challenge to fulfilling this promise is to correctly understand an application's VM's resource demand based on its QoS target and effectively optimize the resource allocation across VMs based on resource-provider objectives. The major difficulty lies in the intrinsically dynamic and complex nature in the resource usage behaviors in such virtualized system.

First, the dynamics in an application's workload can lead to complex behaviors in its VM's resource usages as its intensity and composition change over time. For instance, a web workload's request rate varies depending on the time of day and the occurrence of events [26]; a database workload can also change in

terms of its composition of a wide variety of queries with different levels of CPU and I/O demands [18]. Second, interference among VMs hosted on the same physical machine can lead to complex nonlinear resource usage behaviors as they compete for various types of resources that cannot be strictly partitioned. For example, when co-hosted VMs compete for the shared last level cache or disk I/O bandwidth, the relationship between each VM's resource allocation and its application's performance is known to be nonlinear [11][25]. Finally, even if the application workloads stay relatively steady, their SLAs, which specify the QoS that they require and the cost that they are willing to pay, may change over time. Consequently, resources in the system need to be reallocated across different applications' VMs in order to sustain the system-level objective. As more applications become Internet-scale and resources become more consolidated, the above scenarios would also be increasingly common in a virtualized system.

Different approaches have been studied for virtual resource management and they are examined in detail in Section 5. In particular, machine learning techniques can be employed to automatically learn the relationship between a VM's resource allocation and its application's performance; Control-theory techniques can be used to build a feedback loop into the resource management which can automatically adjust resource allocations and quickly reach the desired system objective. This paper proposes a new resource management approach based on the combination of these two types of techniques that can effectively capture the nonlinearly in virtualized system behaviors and quickly adapt to the changes in such behaviors, which are discussed in details in the following subsections.

2.2 Fuzzy-logic based System Modeling

This paper adopts a fuzzy-logic-based learning technique to model application performance and VM resource usage in a virtualized system such as utility datacenters and clouds, because fuzzy modeling is particularly suited to efficiently model systems with complex behaviors [7]. The technique combines fuzzy logic with mathematical equations to describe the discovered patterns of system behavior and to guide the control strategies of the system. A fuzzy model is a rule base which consists of a collection of fuzzy rules in the form of "If x is A then y is B", where A and B are linguistic values defined by fuzzy sets with associated membership functions. These rules are trained using the input (x) and output (y) data observed from the system and together they represent the model representing the system behaviors.

While building a fuzzy model, *data clustering* techniques (e.g., [13]) are often employed to discover the important features of the system and derive a concise representation of the system's behavior. Each cluster is treated as a fuzzy set and then each set is associated with a fuzzy rule. As a result, only a small number of fuzzy rules are needed in the fuzzy model. The mapping from a given input to an output on a fuzzy rule base is called *fuzzy inference*, which entails the following steps: 1) Evaluation of antecedents: the input variables are *fuzzified* to the degree to which they belong to each of the appropriate fuzzy sets via the corresponding membership functions, 2) Implication to consequents: implication is performed on each fuzzy rule by modifying the fuzzy set in the consequent to the degree specified by the antecedent; 3) Aggregation of consequents: the outputs of all the fuzzy rules are aggregated into a single fuzzy set which is then inversely translated into a single numeric value through a *defuzzification* method.

Note that the fuzzy modeling approach differs fundamentally from traditional rule-based system management approach [20][21]. The latter is based on the use of a set of event-condition-action rules which are triggered only when certain events happen and some preconditions are met. In such an approach, the rules are typically specified by system experts, which is often intractable to apply to a complex system because of the difficulty in defining thresholds and corrective actions for all possible system states. In contrast, a fuzzy model is built for the entire input space of the system and can be used for *continuous* control, where the fuzzy rules representing the model are created *automatically* from the observed input-output data.

2.3 Model Predictive Control

Model predictive control (MPC) [2] is an advanced control technique in which the controller takes control actions by optimizing an objective function that defines the objective of controlling the system. To enable the predictive capabilities of the control system, an explicit model that characterizes the system behaviors is leveraged to make predictions of system output over a specific future prediction horizon. Such modeling and optimization typically involved in MPC can be performed iteratively in an online fashion, where real-time data are used to update the model in the modeling phase and new optimal action is computed based on the model to adjust the system control. In this way, the system can adapt to the changes in the system behavior in a timely fashion.

In contrast to an open-loop optimal control technique, the MPC system works in a closed-loop manner by feeding back the information on previous inputs and outputs to the controller at the end of each control period in order to keep track of prediction errors and control variations, so that on one hand the controller is able to make more informative control actions based on the feedbacks, while on the other hand the system is able to be driven back to the set-point target appropriately without large oscillations even in the presence of noise.

MPC has been used by related work on VM resource management (examined in detail in Section 5), where most approaches adopt "black box" linear input-output models which are accurate enough to model nonlinear system behaviors within a limited region of control operation. In this paper, we propose to use fuzzy modeling to build the model in MPC which can capture the nonlinearity in system behaviors and perform optimized control over the entire operating space. We believe that such a fuzzy MPC approach has the potential to both capture the nonlinearity in a VM's resource usage behaviors effectively and adapt to the dynamic changes in these behaviors in a timely manner.

3. APPROACH

Figure 1 illustrates the architecture of our proposed system which consists of four key modules, *Application Sensors*, *Fuzzy Model Estimator*, *Optimizer*, and *Resource Allocator*. As the applications are running on their VMs, the *Application Sensors* monitor the performance $y_i(t)$ from each application i and then send them to *Fuzzy Model Estimator*. The estimator collects all necessary information including current and historical application performance and VM resource allocations to create the fuzzy model for performance prediction. Such a model, which represents the relationship between the control input (resource allocations to the VMs) and the measured output (performance of the applications), is updated every control period. Based on the model, the *Optimizer* produces a resource allocation scheme for

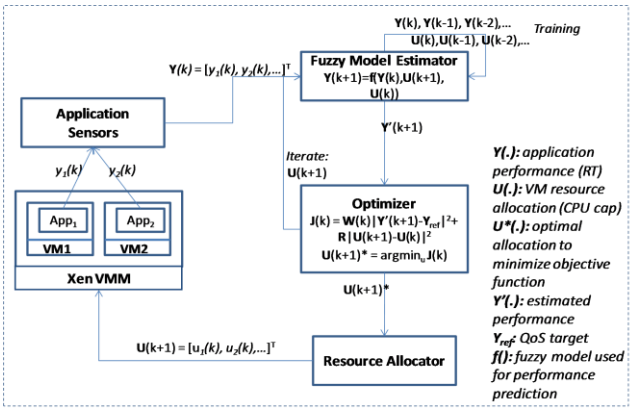

Figure 1 The architecture of the FMPC control system

the next time interval that optimizes the system according to a predefined objective function. Then the *Resource Allocator* adjusts the VM's resource allocations accordingly. Together, these modules form a continuous feedback loop for the virtual resource management.

3.1 Fuzzy Model Estimator

The proposed FMPC is a fuzzy-model-based predictive control approach [2]. The major difference between FMPC and traditional MPC approaches lies in the modeling part. In FMPC, the fuzzy model estimator is responsible for building models that can describe complex system behaviors using fuzzy logic based method. The strength of this approach includes the following aspects: 1) it simplifies the learning of the complex models by describing nonlinearity using a set of linear sub models captured by the fuzzy rules; 2) it can perform optimized control over the entire operating space; 3) it inherits the benefits of traditional predictive control that can guarantee dynamic performance in a closed-loop system and achieve desired target in a stable manner.

Consider a resource provider that hosts multiple applications by multiplexing multiple types of resources among them via VMs, a general *MIMO* model in MPC described by the following equation is used to build the time-varying relationship between resource allocations and application performance,

$$\boldsymbol{y}(t) = \Phi(\boldsymbol{u}(t), \dots, \boldsymbol{u}(t-m), \boldsymbol{y}(t-1), \dots, \boldsymbol{y}(t-n))$$

where the input vector $\mathbf{u}(t) = [u_1(t), u_2(t), \dots, u_N(t)]^T$ represents the allocation of p types of controllable resources to the q applications' VMs at time step t ($N = pq$), and the output vector $\mathbf{y}(t) = [y_1(t), y_2(t), \dots y_q(t)]^T$ is referred to as the predicted performance of q applications at time step t. For example, if there are two applications whose performance relies on two types of resources, i.e. CPU and disk I/O, then $\mathbf{u}(t)$ is a 4-dimensional vector, $[u_{CPU1}(t), u_{CPU2}(t), u_{IO1}(t), u_{IO2}(t)]^T$.

In traditional MPC approaches, linear models are applied to approximate the nonlinear behaviors around the current operating point, while m and n reflecting the impact of the previous inputs and outputs to current prediction are usually set to small values in order to reduce the complexity of the model, e.g., with $m = 0$, $n = 1$, $\boldsymbol{y}(t) = \Phi(\boldsymbol{u}(t), \boldsymbol{y}(t-1)) = \boldsymbol{a}\mathbf{u}(t) + \mathbf{b}\boldsymbol{y}(t-1)$.

In our proposed FMPC, the general Φ function from the control inputs to the system outputs is instantiated by a fuzzy model composed of a collection of Takagi-Sugeno fuzzy rules [7]

$$\boldsymbol{R}^i: If\ \boldsymbol{u}(t)\ is\ \boldsymbol{A}^i\ and\ \boldsymbol{y}(t-1) is\ \boldsymbol{B}^i,$$
$$then\ \boldsymbol{y}^i(t) = \boldsymbol{a_i}\boldsymbol{u}(t) + \boldsymbol{b_i}\boldsymbol{y}(t-1) \quad (1)$$

In the premise $\boldsymbol{A}^i$ and $\boldsymbol{B}^i$ are fuzzy sets associated with the fuzzy rule $\boldsymbol{R}^i$. Their corresponding membership functions μ_{Ai} and μ_{Bi} determine the membership grades of the control input vectors $\boldsymbol{u}(t)$ and $\boldsymbol{y}(t-1)$, respectively, which indicate the degree that they belong to the fuzzy sets. In the consequence, the output $\boldsymbol{y}(t)$ is a linear function of the current control input and the previous output with trainable parameter matrices $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$.

The *Estimator* adopts an efficient one-pass clustering algorithm, subtractive clustering [13], to build a concise rule base with a small number of fuzzy rules that can effectively represent the VMs' behaviors. Each cluster exemplifies a representative characteristic of the system behaviors and can be used to create a fuzzy rule accordingly. In this way, both the system structure and parameters are learned and adapted in real time from online data streams. The system model gradually evolves as opposed to having a fixed structure model, and the learning process is incremental and automatic. Owing to the speed of subtractive clustering and fuzzy modeling, this whole model updating process can be completed quickly within a fine-grained control interval.

The *Estimator* is invoked by the *Optimizer* discussed below in every control step t to predict the performance for specific input values and assist it to search for the optimal allocation solution across the input space. The *Estimator* applies *fuzzy inference* to predict the output $\boldsymbol{y}(t)$ for a given control input $< \boldsymbol{u}(t), \boldsymbol{y}(t-1) >$ based on a trained fuzzy rule base with S fuzzy rules. It entails the following steps: 1) Evaluation of antecedents: the input variables are *fuzzified* to the degree, $\boldsymbol{\delta}^i$, to which they belong to each of the fuzzy sets via the corresponding membership functions for each fuzzy rule $\boldsymbol{R}^i$;2) Implication to consequents: implication is performed on each fuzzy rule by computing $\boldsymbol{y}^i(t)$ based on the equation in the consequent of the rule; 3) Aggregation of consequents: the final prediction is performed as $\boldsymbol{y}(t) = \sum_{i=1}^{S} \boldsymbol{\delta}^i \boldsymbol{y}^i(t)$, where the outputs $\boldsymbol{y}^i(t)$ of all the fuzzy rules are aggregated into a single numeric value based on their corresponding membership grades $\boldsymbol{\delta}^i$.

3.2 Optimizer

Generally, the objective function in MPC can be formulated as

$$J(t) = \sum_{i=1}^{P} \|\Delta \boldsymbol{y}(t+i|t)\|^2\, Q(i) + \sum_{i=0}^{M-1} \|\Delta \boldsymbol{u}(t+i|t)\|^2\, R(i) \quad (2)$$

where P and M indicate the prediction and control horizon. $\boldsymbol{\Delta y}$ is the *predictive error* between $\boldsymbol{y}(t+i)$, the output of the next ith step predicted from the current time step t (using the fuzzy model produced by the Estimator), and the reference output $\boldsymbol{y_{ref}}(t+i)$ of the next ith step. $\boldsymbol{\Delta u}$ indicates the *control effort*. The importance of tracking accuracy in performance targeting and maintaining stability in control operation can be determined by tuning the $Q(i)$ and $R(i)$ factors for the two components of the equation. Larger Q factor will make the controller react aggressively to tracking errors in performance. Larger R factor will guarantee the stability of the system by preventing from large oscillation in the resulting resource allocation, but lead to slower response to the tracking error.

To reduce the complexity of the problem, we choose an objective function with $M = P = 1$. In addition, in Equation 2, the performance of the q different applications, represented in $\boldsymbol{y} = [y_1(t), y_2(t), \dots y_q(t)]^T$, are treated with equal importance. In practice, applications concurrently hosted in a virtualized datacenter or cloud are often given different preferences, because they have different priorities or they generate different amounts of revenue to the system. Without loss of generality, we use a weight vector $\boldsymbol{w} = [w_1(t), w_2(t), \dots w_q(t)]^T$ to represent the preferences

given to the applications. The objective function can be formulated as

$$J(t) = Q\left\|\boldsymbol{w}\cdot\left(\boldsymbol{y}(t+1)-\boldsymbol{y_{ref}}\right)\right\|^2 + R\left\|\left(\boldsymbol{u}(t+1)-\boldsymbol{u}(t)\right)\right\|^2$$
$$= Q\sum_{i=1}^{q}\left[w_i\left(y_i(t+1)-y_{refi}\right)\right]^2 + R\sum_{i=1}^{N}|u_i(t+1)-u_i(t)|^2 \quad (3)$$

$$\boldsymbol{u}(t)^* = argmin_{\boldsymbol{u}}J(t)$$

where $\boldsymbol{y}_{ref}$ is the desired QoS target that can be set manually; $\boldsymbol{u} \in \Re$ is the input space which specifies the allowable range for the input, particularly, the constraints on the total resource capacity; and *u** represents the optimal set of inputs that minimizes the objective function. To simplify the computation, this optimization problem can be approximately decomposed into *S* sub-problems [1]. Each of them is associated with a fuzzy rule in the rule base and represents a typical constrained linear least-squares problem that can be solved by a standard solver (e.g., *lsqlin* in MATLAB). Finally, all the sub-problem solvers are coordinated to derive the global optimization. Note that only a small number of rules will be produced by the clustering-based fuzzy modeling approach, so the computational effort is limited.

As described above, the Estimator and Optimizer work together in an online closed-loop. The input-output data pair <$\boldsymbol{u}(t)$, $\boldsymbol{y}(t)$> is measured and collected in every control period to train the fuzzy model. A MIMO fuzzy model can handle a coupled system with multi-input and multi-output to describe complex system behavior with implicitly contentions from system components. Once the model is established, it performs as a prediction tool for the controller to search for the optimal $\boldsymbol{u}(t+1)$ that promise the best $\boldsymbol{y}(t+1)$ at the end of each control period.

4. EVALUATION

4.1 Setup

This section evaluates our proposed FMPC-based virtual resource management using a comprehensive benchmark hosted on a typical VM environment. The testbed is a quad-core Intel Q6600 2.4GHz physical machine, which has 4GB RAM and 142GB SATA disk storage. Xen 3.3.1 is installed to provide the VMs, where the operating system for both Dom0 and DomU VMs is Ubuntu Linux 8.10 with paravirtualized kernel 2.6.18.8. Each DomU VM is configured with 2 virtual CPUs and 1.7G RAM. The FMPC controller is hosted on Dom0 with the remaining resources. In this evaluation we focus only on the management of CPU resource where the CPU allocation is done by setting CPU caps to VMs using Xen's Credit CPU scheduler [19].

The RUBiS benchmark used in our experiment models a multi-tier online auction site that supports the core functionalities such as browsing, selling, and bidding [14]. To evaluate our controller's accuracy and adaptability for modeling the complex behaviors of such a multi-tier application as a black box, the web-tier and database-tier of one RUBiS instance are deployed on the same DomU VM with Apache Tomcat 4.1.40 and MySQL 5.0. Four additional client VMs, each configured with 1 CPU and 512M RAM are hosted on another identical physical machine and they can launch up to 8000 emulated client sessions in total.

The resource modeling and control period is set to 20 seconds in all the experiments. Here we consider the case when the predicted performance is only dependent on the current resource allocation. So Equation 1 is revised as $\boldsymbol{R^i}$: $If\ \boldsymbol{u}(t)\ is\ \boldsymbol{A}_i,\ then \boldsymbol{y^i}(t) = \boldsymbol{a_i}\boldsymbol{u}(t) + \boldsymbol{b_i}$. In Equation 3, both $\boldsymbol{u}$ and $\boldsymbol{y}$ are normalized by their maximum values that the system can achieve; and the *Q* and *R* factor are both set to 1 in order to balance the importance between tracking accuracy and controlling stability.

A Linear MPC (LMPC) based approach which leverages a linear auto-regressive-moving-average (ARMA) model [4] in the modeling part of MPC is used as a baseline. By comparing it to our FMPC-based approach, we can evaluate whether our proposed approach can estimate VM resource needs more accurately and achieve better level of service. For both approaches, as soon as the workload is launched, the controller starts with an initial resource allocation that is much less than the actual demand. The model is created from scratch once it collects the first few data points and afterwards it is updated in every control interval. In LMPC, the parameters for the linear model are estimated online using the recursive least squares method [15]; in FMPC, the fuzzy model is constructed by subtractive clustering where the parameters $\boldsymbol{a_i}$ and $\boldsymbol{b_i}$ in each fuzzy rule are trained by neuro-adaptive learning [16].

4.2 Experiments with Changing Workload

In the first experiment, we evaluate whether the FMPC approach can correctly allocate resource to a single VM according to its application's QoS target and whether it can deal with dynamic changes in the application's workload. We vary the RUBiS workload intensity by changing the number of concurrent client sessions, from 2400 to 3200 then to 4000. Each phase is kept for 15 control intervals (300s). The corresponding throughput targets for each phase are set to 400, 500 and 600 requests/s. The fuzzy model adapts as the workload changes: during the first phase, only 1 fuzzy rule is established in the rule base; by the end of the experiment, 2 rules are trained.

Figure 3 shows the performance (throughput in requests per second) of RUBiS measured every control interval, from using our proposed FMPC approach to manage the VM resources versus using the LMPC approach. As we can see both approaches are able to track the changes in the workload at periods 15 and 30 and meet the specified QoS targets pretty closely. However, FMPC outperforms LMPC in several important aspects. First, the FMPC based approach is more accurate in meeting the specified QoS target. The average *steady state error* throughout all three phases is 2.3% for FMPC and 2.9% for LMPC; particularly in the third phase, the steady state error is 1.7% for FMPC 3.3% for LMPC.

Second, the performance controlled by FMPC adapts faster than LMPC when a step change occurs in the workload intensity. The average *settling time* to within 5% of the steady state for all three phases is 3 control intervals in FMPC and 5 intervals in LMPC, where in each phase FMPC is 1 to 2 intervals faster than LMPC in settling time. This advantage is because that FMPC's fuzzy modeling is more accurate than LMPC's linear modeling when transition happens. Owing to the flexibility of FMPC, it tunes its model more adaptively than LMPC does. For example, instead of being restricted by a fixed linear shape mode of LMPC, FMPC can immediately add a new rule as soon as new data comes which cannot be fit into current model. As a result, LMPC suffers from more than 20% tracking error ($1-y/y_{ref}$) when the first transition occurs, whereas in FMPC there is almost no tracking error. Overall, the average of the performance across all three phases using FMPC is about 5% higher than using LMPC approach.

To better analyze the results, Figure 4 shows the corresponding CPU allocations. With an initial CPU allocation of 50% the FMPC controller is able to detect resource under-provision as soon as the first target miss is observed and converge to an

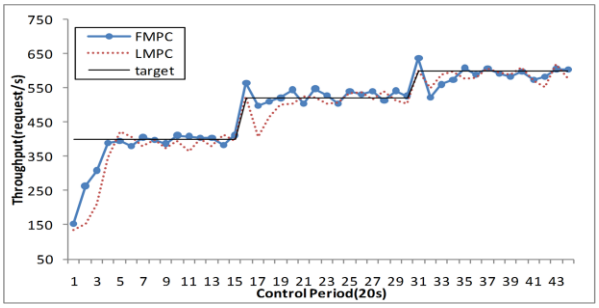

Figure 3 Performance for workload with changing intensity

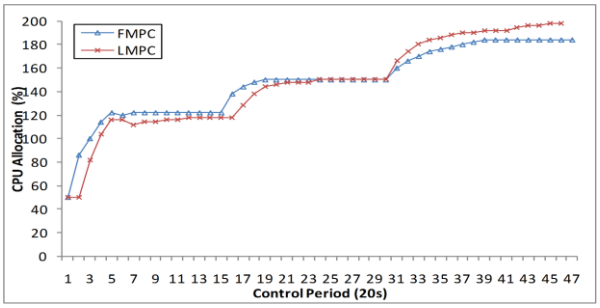

Figure 4 CPU allocation for workload with changing intensity

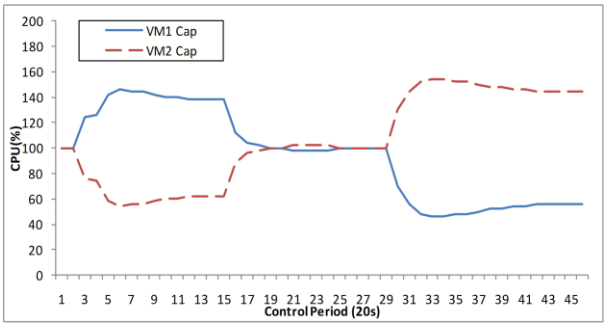

Figure 5 CPU allocations for interfering VMs

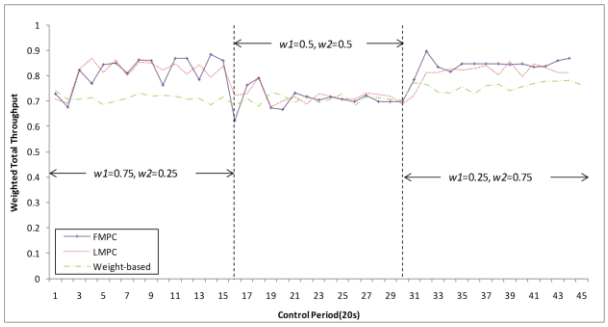

Figure 6 Weighted total throughput of interfering VMs

optimal allocation for meeting the target within a few control intervals. In comparison, the LMPC acts at least one interval slower than FMPC in the first phase and two intervals slower in the second phase. In the third phase, the LMPC approach also allocates 14% more CPU than the FMPC approach. Such over provisioning could lead to loss of performance for other co-hosted VMs and loss of revenue for the entire virtualized system.

In summary, the proposed FMPC controller can automatically track the reference QoS for an application by allocating the proper amount of resources to its VM. It also outperforms LMPC in terms of the adaptively and accuracy.

4.3 Experiments with Interfering VMs

In this experiment, we evaluate how the proposed FMPC controller manages the resource allocations to VMs that interfere with each other during their executions. This scenario is both interesting and challenging because the interference between the VMs result in correlation between their performance models while the level of interference is also dependent on one VM's workload intensity. By experimenting with the RUBiS workload, we notice that having 2400 concurrent users for one VM-hosted RUBiS application would create a total CPU demand of 100% on the single dual-virtual-CPU VM which hosts both the web and database tiers of RUBiS. However, if we run two independent RUBiS VMs concurrently and host both VMs on the same pair of physical cores (using CPU affinity), then neither VM can achieve the same level performance when serving the same workload even though each of them can still get 100% of CPU. This observation confirms the existence of performance interference across VMs which commonly exists on a highly consolidated virtualized system.

We evaluate the performance of our proposed FMPC approach in optimizing the system-level management objective and how it reacts to the dynamic changes in management policy. We use a two-input-two-output FMPC to control the resource allocations to the two VMs. The input variables are the CPU allocations to the two VMs and the outputs are the measured performance of the two RUBiS applications. As discussed in Section 3.2, we assign different weights w_1 and w_2, to the two VMs ($w_1 + w_2 = 1$), which represent the different priorities or impacts to revenue as determined by the application SLAs. So the objective function is:

$$J(t) = \left[w_1(t)\left(y_{ref1} - y_1\right)^2 + w_1(t)\left(y_{ref2} - y_2\right)^2 \right] + |\Delta \boldsymbol{u}(t)|^2$$

where $\boldsymbol{u} = [u_1, u_2]^T$ denotes the CPU caps set to the two VMs. Since they share the same two physical cores, the total available CPU is 200%. The QoS target y_{refi} is set to 400 request/s for both RUBiS instances, which is the performance that it can achieve with 100% CPU and no interference. We fix the workload intensity for each VM to 2400 client sessions, but we change their weights (as shown in Figure 6) to represent the scenario where application SLAs change over time, in order to evaluate whether the proposed FMPC approach can always achieve optimal total revenue and how quickly it adapts to such dynamic changes.

Figure 5 shows the CPU allocations to both application VMs made by our FMPC controller in the experiment. Initially, both VMs have equal CPU shares. In the first phase, *VM1* got more CPU resource (around 140%) than *VM2* (around 60%) because the former has a higher weight. Starting from the interval 16, as the weights change to 1:1, u_1 decreases and u_2 increases, both quickly converging to 100% of CPU as expected. During the third phase, *VM1* is assigned less CPU (around 60%) than *VM2* (around 140%) because VM2 now has a higher weight. Interesting, when one VM's weight is set to three times of the other one, it does not get three times of resource allocation, because of the nonlinear relationship between VM resource allocation and application QoS.

To demonstrate the effectiveness of the FMPC-based resource management, we compare it with the LMPC-based approach and another weight-based scheme which intuitively partitions the total resource to VMs based on their assigned weights (i.e., the CPU caps are set to 3:1, 1:1 and 1:3 for VM1:VM2 across the three phases). The *weighted total throughput* that is aggregated by the weighted throughputs from all applications in the system is used

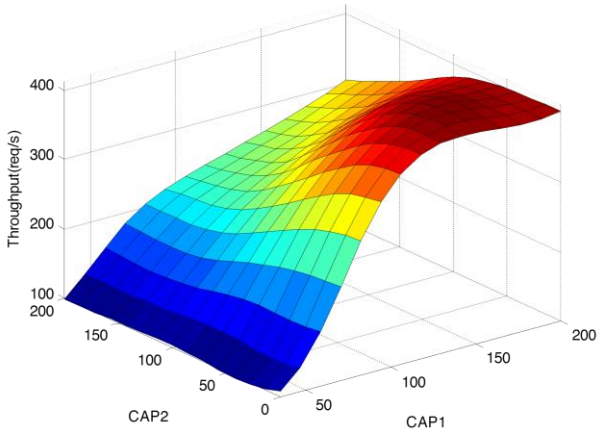

Figure 7 The 3-D fuzzy model for VM_1

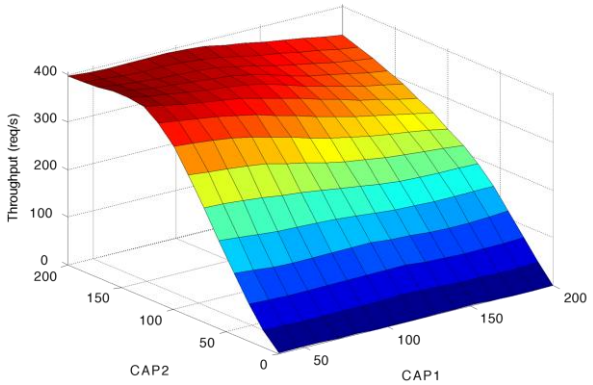

Figure 8 The 3-D fuzzy model for VM_2

as the performance metric in this experiment. The results in Figure 6 illustrate that the allocation decisions made by the FMPC controller substantially outperform the weight-based scheme across all three phases. During the first two phases, LMPC works as well as FMPC. However, in the third phase, FMPC generates about 4.7% more throughput in average than LMPC does. From the results, we can see that FMPC can achieve higher weighted total throughput, particularly in the first and third phases. Nonetheless, the FMPC approach can correctly capture these nonlinear behaviors and produce much better resource allocations.

To further understand the impact of interference on VM performance, we use fuzzy modeling to build a global two-input two-output non-linear model given the entire input space for the two competing RUBiS VMs, where the two control inputs are the CPU allocations to the VMs and the two control outputs are the measured performance for the individual RUBiS instances hosted on the VMs. The model is created in the following way: while keeping the workloads concurrently running against the two VMs, the CPU cap set to each VM is varied from 0% to 200%. The model is trained offline based on a total of 350 data points collected from a set of evenly distributed cap values in this range. Each data point is 4-element tuple $< cap_1, cap_2, y_1, y_2 >$. The fitting error is 7.4%.

For better illustration, we split this model into two 3-D models and illustrate them separately in Figure 7 and 8 each representing the behavior of one VM under the interference from the other. From the models, we can see that for each application, the performance is not only dependent on the CPU allocation to its hosting VM but also affected by the CPU cap set to the other VM. With the same value of cap set to one VM, its application's performance will drop as the cap value of the other VM increases. Nonetheless, the fuzzy logic based modeling technique is able to capture more complex relationship between resource allocation and performance with the presence of interference resulted from co-hosted VMs.

In summary, the FMPC approach is able to optimize the resource allocation for two interfering VMs and sustain it by quickly adapting to the changes in service-level policies. Meanwhile, the CPU usages for running FMPC and LMPC are both low (10.7% and 11.5% respectively). It takes about 10ms for FMPC to complete the modeling and optimization in each control loop.

5. RELATED WORK

Various solutions have been studied in the literature to address the problem of resource management in virtualized systems. Due to the limited space, here the discussion focuses on the two most relevant types of approaches. One type of approaches considers machine learning techniques to automatically learn the complex resource model for a virtualized system based on data observed from the system. For example, the CRAVE project employs simple regression analysis to predict the performance impact of memory allocation to VMs [22]; Wood et al. also use regression method to map a resource usage profile obtained on a physical system to one that can be used on a virtualized system [23]; The VCONF project has studied using reinforcement learning to automatically tune the CPU and memory configurations of a VM in order to achieve good performance for its hosted application [24]; Kund *et al.* employ artificial neural networks to build performance models that consider both resource allocation to VMs and resource interference between VMs [25].

Another type of approaches applies control theory [30] to automatically adjust VM resource allocation in order to achieve the desired system-level objective. In particular, linear MIMO MPC has been studied for datacenter resource management where multiple applications are sharing a common pool of resources. For example, Liu *et al.* consider the complex interactions and dependencies among different application tiers hosted on VMs and optimize their CPU allocations in order to achieve QoS differentiation among the multi-tier applications. Its follow-up work [4] builds an online ARMA model for each application to represent the relationship between the allocations of multiple resources and normalized performance when the application tiers are hosted on VMs spanning across physical nodes. Linear MPC has also been studied to capture the last-level cache interference between concurrent VMs and compensate its performance impact [10]. In the related work on other aspects of system management, Wang *et al.* uses MPC to optimize the power consumption for multiple servers [8]; Lu *et al.* applies MPC to the control of CPU utilization in a highly coupled distributed real-time system [3].

Our proposed FMPC approach combines the strengths of machine-learning and control-theory techniques in virtual resource management. Compared to other modeling based approaches, the FMPC approach can be effectively applied online and quickly adapt to changes in system behaviors. Typical model-based approaches require substantial data for training the model which is difficult to do online. Even if a model can be built offline, it is difficult to adapt it online when the system behavior changes. Compared to other MPC-based approaches, the FMPC approach can well capture nonlinear system behavior without

much learning overhead. In a typical linear-model-based MPC approach, a linear model is assumed to approximate the nonlinear behavior within a limited region of an operation point while it can be updated adaptively as the system moves from one operating point to another. However, as demonstrated by our experiment results, the FMPC approach can more accurately capture the system behavior with a nonlinear fuzzy model and it can perform optimized control continuously over the entire operating space.

6. CONCLUSION AND FUTURE WORK

This paper presents a new fuzzy modeling based predictive control (FMPC) approach that can automatically manage the resources in a virtualized system based on the system-level objective. This approach is based on the combination of fuzzy-logic based modeling for capturing complex system behaviors and MPC-based resource control for agile system optimization and adaption to changes in the system. Experiments based on a typical online transaction benchmark demonstrate that this approach can accurately estimate the resource allocation for a VM hosting changing workload and achieve the desired QoS. The results also show that it can capture the complex behaviors of resource competing VMs and optimize their resource allocations in such a scenario. Compared to traditional linear modeling based MPC, the FMPC approach is shown to be better in terms of the obtained QoS and the speed to achieve the application or system target.

Currently we are evaluating this approach on the management of both CPU cycles and I/O bandwidth for VMs with more complex resource usage and contention. We will evaluate the effectiveness of our approach for applications with SLOs specified by response times, instead of throughput, and more interesting nonlinear behaviors. We also plan to evaluate our prototype on larger virtualized systems and test its scalability. In our future work, we will extend this work to provide cross-physical-host resource management that considers VM migration as an additional means of resource control and support VMs with correlated performance requirements.

7. ACKNOWLEDGEMENT

This research is sponsored by National Science Foundation under grant CCF-0938045, OCI-0910812, IIP-0932023, CNS-0855123, and IIP-0758596 and Department of Homeland Security under grant 2010-ST-062-000039. The authors are also thankful to the anonymous reviewers for their useful comments. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

8. REFERENCES

[1] Y. L. Huang et al., "Fuzzy Model Predictive Control," IEEE Transactions on Fuzzy Systems, Vol. 8, No. 6, 2000.
[2] J. Maciejowski, "Predictive Control with Constraints," Prentice Hall, 1 edition, 2002.
[3] C. Lu et al., "Feedback Utilization Control in Distributed Real-Time Systems with End-To-End Tasks," *TPDS*, 2005.
[4] P. Padala et al., "Automated Control of Multiple Virtualized Resources," Proceedings of EuroSys, 2009.
[5] J. D. Morningred et al., "An Adaptive Nonlinear Predictive Controller," *Chem. Eng. Sci.*, vol. 47,pp. 755–762, 1992.
[6] L. A. Zadeh, "Fuzzy Sets", Information and Control, 1965.
[7] T. Takagi, M. Sugeno, "Fuzzy identification of systems and its application to modeling and control," *TSMC*, 1985.
[8] Xiaorui Wang et al., "MIMO Power Control for High-Density Servers in an Enclosure," TPDS, 2010.
[9] X. Liu et al, "Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform," CDC, 2007.
[10] R.Nathuji et al., "Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds," EuroSys, 2010.
[11] Diwaker Gupta et al., "Enforcing Performance Isolation Across Virtual Machines in Xen," Middleware, 2006.
[12] J.N. Matthews et al., "Quantifying the Performance Isolation Properties of Virtualization Systems," ExpCS, 2007.
[13] S. Chiu, "Fuzzy Model Identification Based on Cluster Estimation," Journal of Intelligent and Fuzzy Systems, 1994.
[14] C. Amza et al., "Specification and Implementation of Dynamic Web Site Benchmarks," WWC, 2002.
[15] K. Astrom, B. Wittenmark, "Adaptive Control," 1995.
[16] Neuro-adaptive Learning, URL: http://www.mathworks.com/help/toolbox/fuzzy/fp715dup12.html.
[17] Jing Xu et al., "Autonomic Resource Management in Virtualized Data Centers Using Fuzzy-logic-based Control", Cluster Computing, Vol. 11, No. 3, Pages: 213-227, 2008.
[18] L. Wang et al., "Autonomic Resource Management for Virtualized Database Hosting Systems," Tech. Report, 2009.
[19] Credit-Based CPU Scheduler, URL: http://wiki.xensource.com/xenwiki/CreditScheduler.
[20] HP-UX Workload Manager, http://docs.hp.com/en/5990-8153/ch05s12.html
[21] J. Rolia et al., "Configuring Workload Manager Control Parameters for Resource Pools," NOMS, April 2006.
[22] J. Wildstrom et al., "CARVE: A Cognitive Agent for Resource Value Estimation", ICAC, 2008.
[23] T. Wood et al., "Profiling and Modeling Resource Usage of Virtualized Applications," Middleware, 2008.
[24] J. Rao et al., "VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration", ICAC, 2009.
[25] S. Kundu et al., "Application Performance Modeling in a Virtualized Environment," HPCA, 2010.
[26] 1998 World Cup Web Site Access Logs, URL: http://ita.ee.lbl.gov/html/contrib/WorldCup.html.
[27] M. Kallahalla et al., "SoftUDC: A Software-based Data Center for Utility Computing," Computer, 2004.
[28] Amazon Elastic Compute Cloud (Amazon EC2), URL: http:// aws.amazon.com/ec2/.
[29] Windows Azure Platform, URL: http:// www.microsoft.com/windowsazure/.
[30] T. Abdelzaher et al., "Introduction to Control Theory and its Application to Computing Systems, Performance Modeling and Engineering ", Springer, 2008.