QoS-driven Cloud Resource Management through Fuzzy Model Predictive Control

Lixi Wang, Jing Xu*, Hector A. Duran-Limon**, Ming Zhao,
Florida International University, Miami, FL, USA
* VMware, Inc., Palo Alto, CA, USA
** University of Guadalajara, Zapopan, Jalisco, México

***Abstract*—Virtualized systems such as public and private 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 system through predictive control. The resulting fuzzy-model-predictive-control (FMPC) approach is capable of optimizing the VM resource allocations to applications according to their QoS targets. This approach is incorporated in a two-level cloud resource management framework where at the VM host level the node controllers employ FMPC to optimize dynamic VM resource allocations within individual hosts, and at the cloud zone level the global scheduler coordinates the node controllers to optimize resource utilization across hosts through dynamic VM migrations. The proposed approaches were implemented for Xen-based virtualized systems and evaluated using typical benchmarks (RUBiS, FreeBench) on a testbed with over 100 concurrent VMs. The results demonstrate that FMPC can accurately model the resource demands for dynamic applications and optimize the resource allocations to VMs with complex contentions. It substantially outperforms the traditional linear modeling based predictive control approach. The two-level resource management can make effective use of VM migrations to further improve performance across hosts as the host-level loads vary over time.**

Keywords—QoS; cloud computing; resource management

I. INTRODUCTION

Virtualized systems such as public and private clouds [1][2] are emerging as promising new platforms that can significantly improve how resources are provisioned to applications and how computing is delivered to users. On 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 achieve 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 one another. 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 may change over time and as a result resources 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: 1) how to accurately capture the complex relationship between resource allocation and application performance; and 2) how to adaptively optimize the VM resource allocation as changes occur dynamically in the system. Specifically in the proposed approach, 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 with good speed. Then a predictive controller uses such a 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.

The paper also proposes a two-level resource management framework based on FMPC, including distributed host-level *Node Controllers* and a cloud zone level *Global Scheduler*. Each node controller uses FMPC to predict the resource demands of its local VMs and optimize the resource allocations according to their QoS targets. The global scheduler further improves performance across VM hosts by planning VM migrations based on the resource demand estimates from the node controllers. The node controllers in turn execute the VM migrations and transfer the performance models of the migrated VMs to minimize the impact of migrations on application performance.

This proposed approach was prototyped on Xen-based virtualized systems and evaluated using typical benchmarks (RUBiS [3], FreeBench [4]). The results demonstrate that FMPC can accurately estimate the resource demand for a VM running dynamically changing workload and quickly achieve the desired QoS target. FMPC can also capture the complex behaviors of resource-competing VMs and optimize the

resource allocations according to their QoS targets. It substantially outperforms the traditional linear model predictive control (LMPC) approach. Moreover, the proposed two-level resource management framework can effectively optimize the performance for more than 100 concurrent VMs running dynamic workloads across multiple hosts.

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

II. BACKGROUND

A. Adaptive VM Resource Management

Emerging virtualized systems such as public and private clouds promise to be important new computing platforms where applications can be executed cost-effectively using resources that are provisioned on demand. The key challenges to fulfilling this promise are how to accurately understand a VM's resource demand based on its hosted application's QoS requirement, and how to effectively optimize the resource allocations to the concurrent VMs in the system according to the service-level objectives. Without such *QoS-driven resource management*, cloud providers cannot support the more economical performance-based service-level agreements (SLA) and cloud users have to pay for resource capacity, instead of performance which is what the users really care about. Consequently, the users have to purchase unnecessary cloud resources to get their desired performance, and the providers cannot maximize their profits from offering the cloud services.

The main difficulty of achieving QoS-driven cloud resource management lies in the intrinsically dynamic and complex nature of the application and system behaviors in a highly consolidated, virtualized environment. 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 [5]; a database workload can also change in terms of its composition of a wide variety of queries with different levels of CPU and IO demands [6]. 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 processor cache or disk IO bandwidth, the relationship between each VM's resource allocation and its application's performance is known to be nonlinear [6][7][8]. 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, resource allocations to the applications' VMs need to be dynamically adjusted in order to sustain the system-level objective.

With the rapidly growing level of application and resource consolidation, the aforementioned scenarios are increasingly common in cloud systems and the management of such systems is in fact increasingly challenging. Different approaches have been studied for virtual resource management and they are examined in detail in Section VII. In particular, *machine learning* based techniques can be employed to automatically learn the relationship between an application's resource allocation and its performance; and *control theory* based techniques can be used to automate the control of resource allocations according to the application- and system-level objectives. This paper proposes a new resource management approach based on the combination of these two types of techniques that can effectively capture the nonlinearity in virtualized system behaviors and quickly adapt to the changes in such behaviors.

B. 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, because fuzzy modeling is particularly suited to efficiently model systems with complex behaviors [9]. The technique combines fuzzy logic with mathematical equations to discover and describe the patterns in system behaviors 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 mathematical model of the system behaviors. Based on such a fuzzy model, fuzzy inference can be applied to compute the output (y) for any given input (x).

Note that the fuzzy modeling approach differs fundamentally from traditional rule-based system management approaches [10][11]. 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.

C. Model Predictive Control

Model predictive control (MPC) [12] 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 controller can adapt to changes in the system behaviors in a timely fashion.

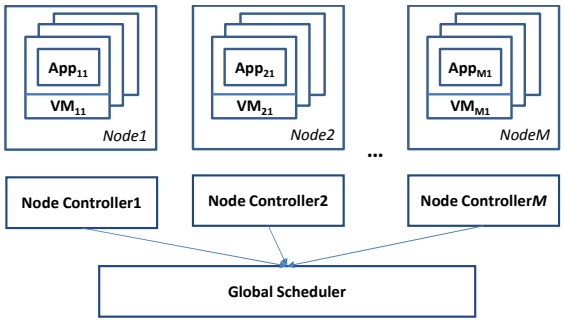


Figure 1. Two-level cloud resource management system

In contrast to an open-loop optimal control technique, the MPC system works in a closed-loop manner by feeding back 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 on one hand the controller is able to make more informative control actions based on the feedbacks, and 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 VM resource management work (examined in detail in Section VII) which adopts linear models which are accurate enough to model system behaviors within a small region of control operation. This paper proposes to use fuzzy modeling to build the model in MPC, which can capture the nonlinearity in system behaviors and perform optimal control over the entire operating space. The rest of this paper presents the details of this approach.

III. Two-level Resource Management Architecture

This paper considers the typical cloud environment where VM hosts are organized into zones: Within each zone, the hosts use shared storage servers to store the VM images so VMs can be quickly live-migrated across the hosts for load balancing; Across zones, VMs cannot be easily live-migrated so it happens only at rare occasions, e.g., when an entire zone is overloaded or under maintenance. Hence, the proposed resource management framework focuses on the dynamic resource allocations at the host level and dynamic VM migrations at the zone level. Nonetheless, the proposed two-level framework can also be used to balance loads across zones using non-live VM migrations according to the entire cloud system's service-level objectives.

Figure 1 illustrates the architecture of the proposed two-level cloud resource management framework which includes a *Node Controller* on every VM host and a *Global Scheduler* for the entire cloud zone. Specifically, a node controller is responsible for dynamically allocating resources to VMs and optimizing them using FMPC according to application QoS targets. The global scheduler dynamically adjusts VM placement through live migration in order to handle load variations on the VM hosts and to improve system-level performance. The node controllers and global scheduler cooperate with each other to complete the cloud resource management. When a node controller updates its predicted resource demands of its local VMs, it sends this information to the global scheduler for making VM migration decisions;

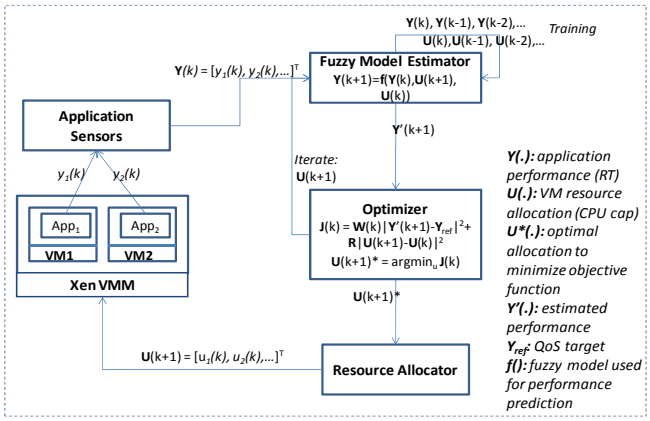


Figure 2. The architecture of the FMPC-based host-level resource management

when a global scheduler decides to migrate a VM, it coordinates with the node controllers on the source and destination hosts to execute the migration and then update their performance models and resource allocations based on the new VM placement.

These two levels of resource management operate at different granularity and time intervals. The node controllers allocate resources at a fine granularity (e.g., CPU cycles) and time scale (e.g., every 20 seconds), because of the low overhead of making such adjustments through the hypervisor interface and the fast speed of the proposed performance modeling and resource optimization techniques. The global scheduler adjusts the resource utilization across hosts in the units of VMs at a longer time scale (e.g., every minute) because of the relatively higher overhead and longer-term effect of VM migrations. Therefore, in this two-level architecture, fine-grained, frequent control actions occur only at the host level within the scope of the limited local VMs, whereas global control takes place at a coarse granularity and infrequently. It is thus easier to scale compared to the alternative one-level architecture that either employs a centralized manager to control the resource allocations to all the VMs across all the hosts, or completely decentralize the management so that a node controller has to communicate with all the other peers in order to obtain global knowledge and coordinate VM migration decisions.

IV. Host-level VM Resource Management

Figure 2 illustrates the architecture of the 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 the necessary information including the current and historical application performance and VM resource allocations to create the fuzzy model for the VMs. 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 the next time interval that optimizes the system according to a predefined objective function. Then the *Resource Allocator* adjusts the VMs' resource allocations accordingly. Together, these modules form a closed feedback loop that works periodically for cloud resource management.

A. Fuzzy Model Estimator

The proposed FMPC is a fuzzy-model-based predictive control approach. The major difference between FMPC and traditional MPC approaches [12] 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 *multi-input-multi-output* (MIMO) model in MPC described by the following equation is used to capture the time-varying relationship between VM resource allocations $\boldsymbol{u}(t)$ and application performance $\boldsymbol{y}(t)$,

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

where the input $\boldsymbol{u}(t) = [u_1(t), u_2(t), \cdots, u_N(t)]^T$ represents the allocation of p types of controllable resources to the q applications' VMs at time step t ($N = pq$), and the output $\boldsymbol{y}(t) = \left[y_1(t), y_2(t), \cdots, y_q(t)\right]^T$ represents the predicted performance of the applications at t. For example, if there are two applications which require on two types of resources, CPU and IO, $\boldsymbol{u}(t) = [u_{vm1}^{CPU}(t), u_{vm1}^{IO}(t), u_{vm2}^{CPU}(t), u_{vm2}^{IO}(t)]^T$. The m and n reflect the impact of the previous inputs and outputs to current prediction and are usually set to small values in order to reduce the complexity of the model, e.g., with $m = 0$ and $n = 1$, $\boldsymbol{y}(t) = \Phi(\boldsymbol{u}(t), \boldsymbol{y}(t-1))$.

In traditional MPC approaches, linear models are applied to approximate the nonlinear behaviors around the current operating point, so $\mathbf{y}(t) = \boldsymbol{a}\boldsymbol{u}(t) + \boldsymbol{b}\boldsymbol{y}(t-1)$, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are dynamically adapted as the system moves across different operating points. In the 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 fuzzy rules [13],

$$\begin{aligned} R_i\text{: } &If\ \boldsymbol{u}(t)\ is\ A_i\ and\ \boldsymbol{y}(t-1)\ is\ B_i, \\ &then\ \boldsymbol{y_i}(t) = \boldsymbol{a_i}\boldsymbol{u}(t) + \boldsymbol{b_i}\boldsymbol{y}(t-1) \end{aligned} \quad (1)$$

In the premise A_i and B_i are fuzzy sets associated with the fuzzy rule R_i. Their corresponding Gaussian membership functions $\mu_{Ai}(\boldsymbol{u}) = e^{-\frac{(\boldsymbol{u}-c)^2}{2\sigma^2}}$ and $\mu_{Bi}(\boldsymbol{y}) = e^{-\frac{(\boldsymbol{y}-c)^2}{2\sigma^2}}$ 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 $\boldsymbol{y_i}(t)$ of rule R_i is a linear function of the current control input and previous output with trainable parameter matrices $\boldsymbol{a_i}$ and $\boldsymbol{b_i}$.

The *Estimator* adopts an efficient one-pass clustering algorithm, subtractive clustering [14], to build a concise rule base with a small number of fuzzy rules that can effectively represent the VMs' behaviors. Subtractive clustering takes a single parameter, radius, which can be adjusted to influence the number of clusters—a larger radius leads to a smaller number of clusters. 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 recursive least square (RLS) learning method is used to train the model parameters online. 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 RLS learning in fuzzy model construction, the model can be updated quickly (sub second), and can be used in every control interval to capture the current system behaviors.

The *Estimator* is invoked by the *Optimizer* discussed below in every control step t to predict the performance for different 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 $\langle \boldsymbol{u}(t), \boldsymbol{y}(t-1) \rangle$ 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 R_i; 2) Implication to consequents: implication is performed on each fuzzy rule by computing $\boldsymbol{y_i}(t)$ based on Equation (1); 3) Aggregation of consequents: the final prediction is performed as $\boldsymbol{y}(t) = \sum_{i=1}^{S} \boldsymbol{\delta_i}\boldsymbol{y_i}(t) = \sum_{i=1}^{S} \boldsymbol{\delta_i}(\boldsymbol{a_i}\boldsymbol{u}(t) + \boldsymbol{b_i}\boldsymbol{y}(t-1))$, where the outputs of all the fuzzy rules are aggregated into a single numeric vector based on their corresponding membership grades $\boldsymbol{\delta_i}$. The fuzzy model captures the system's nonlinearity in a way that each rule characterizes a certain aspect of the system using a simple linear model with low computation complexity, whereas with all the rules aggregated together the model can effectively capture nonlinear behaviors.

B. Optimizer

Generally, the objective function in MPC is formulated as

$$\begin{aligned} J(t) = &\sum_{i=1}^{P} \left\| \boldsymbol{y}(t+i|t) - \boldsymbol{y}_{ref}(t+i|t) \right\|^2 Q(i) \\ &+ \sum_{i=1}^{M} \|\boldsymbol{u}(t+i|t) - \boldsymbol{u}(t+i-1|t)\|^2 R(i) \end{aligned} \quad (2)$$

The first term in the summation represents *tracking error*, i.e., the difference between the predicted output $\boldsymbol{y}(t+i|t)$ and the reference output $\boldsymbol{y}_{ref}(t+i|t)$ of the next P steps. In FMPC, the predicted output is given by the fuzzy model described above. The second term represents the *control effort*, i.e., the amount of resource allocation changes in the next M steps. The importance of tracking accuracy in performance targeting

and maintaining stability in control operation can be tuned by the assigned $Q(i)$and $R(i)$ factors. Larger Q factor will make the controller react more aggressively to the tracking errors. Larger R factor will improve the stability of system by avoiding large oscillation in the resulting resource allocation.

To reduce the complexity of the problem, the *Optimizer* uses an objective function with $M = P = 1$, and $\boldsymbol{y}_{ref}$ can be simply set using the normalized QoS targets of the considered applications. In addition, in Equation (2), the performance of the q different applications, $\boldsymbol{y}(t) = [y_1(t), y_2(t), \cdots, y_q(t)]^T$ are treated with equal importance. In practice, applications hosted in the same cloud can be given different preferences, because they have different priorities or generate different amounts of revenues to the system. Without loss of generality, we use weights $w_i(t), (1 \le i \le q)$ to represent the preferences given to the applications. The objective function can be formulated as

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

The goal of the *Optimizer* is to find a resource allocation $\boldsymbol{u}(t+1)^*$ that can minimize the above objective function, i.e., $\boldsymbol{u}(t+1)^* = argmin_{\boldsymbol{u}} J(t+1)$, subject to the total resource capacity (e.g,. total available CPU time, total available memory capacity) of the host. By taking the resource allocation that minimizes the objective function at each time step, FMPC will be able to optimize the resource allocations to meet the applications' QoS targets, when it is not oversubscribed, or minimize the distance to the targets, when oversubscribed.

The fuzzy performance model in FMPC is rule-based and not differentiable; a minimization problem involving such models cannot be solved by any classical, derivative-based optimization algorithm. A genetic algorithm (GA) method is applied to solve this complex optimization problem [15]. This algorithm is well-known for tackling more general optimization problems in which the objective function is non-differentiable, discontinuous or highly non-linear, that are not well suited for standard optimization algorithm, e.g., quadratic or linear programming. In light of the natural selection process in biological evolution, the GA algorithm encodes a solution in the optimization search space as a gene in biological reproduction. By mimicking the gene combinations in biological reproduction, it iteratively operates on a population of candidate solutions as how a parent generation produces its children generation, by selecting the good parent candidates and performing randomly genetic operations (mutation and crossover) on them to produce the children for the next generation. The goodness of each candidate solution is computed by a predefined fitness function which is usually related to the objective function of the optimization problem. Finally, the population "evolves" towards a globally optimal solution over successive generations.

To implement a GA solver in the *Optimizer*, the control input $\boldsymbol{u}$ is specified as the variable vector in the optimization. The solver considers a fitness function based on the objective function defined in Equation 3, a model function based on the fuzzy model learned by the *Estimator*, and a constraint function based on the resource capacity bound. It then follows the genetic algorithm to search for the optimal resource allocation $\boldsymbol{u}(t+1)^*$. Since the GA solver may take a while to solve such a non-trivial optimization problem, a bound is set on the generations that the algorithm can produce, so that the optimization can finish within a small control interval. Although the solver may return only a suboptimal solution, given the time constraint, as FMPC operates iteratively, it can still steer the system to approach the optimal state.

As described above, the *Estimator* and *Optimizer* work together in an online closed-loop. The input-output data pair $\langle \boldsymbol{u}(t), \boldsymbol{y}(t) \rangle$ 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 behaviors with implicitly contentions from system components. Once the model is established, it serves as a prediction tool for the controller to search for the optimal $\boldsymbol{u}(t+1)$ that promises the best $\boldsymbol{y}(t+1)$ which will be applied to the VM resource allocation in the next control period. It can quickly recover from model inaccuracy (during bootstrapping or dynamic changes in the system), as the observed performance for a given allocation is immediately used to update the model and reflect the current behaviors.

V. CROSS-HOST CLOUD RESOURCE MANAGEMENT

Within a host's resource constraints, the FMPC approach allows the node controller to effectively optimize the host-level performance objective by allocating the resources to its local VMs. However, local optimality achieved at individual VM host level does not guarantee the global optimality of the entire cloud zone because resource utilization may be unbalanced across the hosts. The global scheduler in the proposed two-level cloud resource management addresses this issue and optimizes the zone-level resource utilizations by live-migrating VMs across the hosts. There is a good amount of related work on the use of VM migration to optimize for a variety of performance, energy, and thermal objectives (e.g., [16][17]). This paper focuses on the use of VM migration for cross-host load balancing and its integration with the FMPC-based node controllers.

To formulate the problem of VM consolidation, consider M VMs distributed among N nodes in a cloud zone with an initial placement $D_i = \{VM_{i1}, VM_{i2}, \ldots, VM_{ij}\}\ (1 \le i \le N)$, where $\sum_{i=1}^{N} D_i = M$. Then the necessary condition of VM migration is defined as when the total demands of a certain type of resource (e.g., CPU, memory, IO bandwidth), Res_{ij} from all the VM_{ij} on Host i exceeds its capacity C_i, i.e., $\sum_{VM_{ij} \epsilon D_i} Res_{ij} \ge C_i$.

The global scheduler detects these conditions on its managed hosts based on the VM resource demands estimated by the node controllers. It then uses this information to carefully make migration decisions for the entire system. The global scheduler periodically updates two lists based on the resource demands collected from the node controllers: *OutList*, the list of overloaded nodes which satisfy the

migration condition and need to move out some of its hosted VMs; and *InList*, the list of underutilized nodes with certain amount of residual resources and can be considered as the destination for other VMs to move in. The *OutList* and *InList* are both sorted based on the host-level total resource demand. At every migration interval, the global scheduler identifies the VMs that need to be migrated by iterating the VMs hosted on the nodes in *OutList*, starting from the node with the highest total resource demand. For a VM considered for migration, it chooses a destination node with the least amount of residual resources in *InList*. The new migration descriptor <*VM*, *source_host*, *dest_host* > is then be added to a *MigrationList*. The *OutList* and *InList* will be updated to remove nodes that are not overloaded and underutilized, respectively, anymore after the migration. The global scheduler iterates all nodes in the *OutList* until there is either no moveable VM or no available destination. It then sends the migration descriptors in the *MigrationList* to the node controllers of the involved source and destination hosts to start the migrations.

When a VM is migrated, it needs to be removed from the source host's fuzzy MIMO performance model and added to the destination host's MIMO model. If the migrating VM's performance model has to be retrained from scratch, it would have an impact on its performance as well as the performance of the other co-hosted VMs. To minimize this impact, the node controllers on the source and destination hosts work together and transfer the migrating VM's performance model from the source host and use it to bootstrap its model on the destination host. To facilitate this model transfer, the MIMO model is decomposed into a set of single-input-single-output (SISO) fuzzy models. It is done by zeroing out the components in the MIMO model that represent the correlation between a VM's performance and the allocations to other VMs. The SISO model of the migrating VM is transferred from the source host to the destination host. After migration, the MIMO models for both the source and destination hosts are reconstructed using their own SISO models. Only the correlations among the co-hosted VMs in these MIMO models need to be retrained, which has much less impact on the VMs because a VM's performance is mostly decided by its own resource allocation.

VI. Evaluation

A. Setup

This section evaluates the proposed FMPC-based two-level cloud resource management using representative benchmarks in a typical virtualized environment. The two level of controllers are implemented in C. The *Fuzzy Model Estimator* and the *Optimizer* are both implemented in MATLAB code with a C wrapper. The testbed is a cluster of Dell PowerEdge 2970 servers, each equipped with two six-core 2.4GHz AMD Opteron CPUs, 32GB of RAM, and 1TB SAS storage. Xen 3.3.1 is installed to provide the VMs, and the guest operating system is Ubuntu Linux 8.10 with paravirtualized kernel 2.6.18.8. To allocate CPU time to VMs, the *Resource Allocator* invokes Xen's credit-based CPU scheduler to set the caps on VM CPU usages, bounded by the total available CPU time on each physical host (a CPU core is reserved for Xen's control domain (Dom0) and the *Node Controller* on each host). The *Global Scheduler* runs in the control domain of one of the hosts.

The RUBiS benchmark used in the experiments models a multi-tier online auction site which supports the core functionalities such as browsing, selling, and bidding [3]. It represents the typical web applications commonly found in cloud computing systems. The evaluation also uses a real trace [18] collected from a departmental web server VM to drive the workload in RUBiS and make the experiments more realistic. To evaluate the FMPC approach's accuracy and adaptability for modeling the complex behaviors of such a VM as a black box, the web and database tiers of RUBiS are deployed on the same DomU VM using Apache Tomcat 4.1.40 and MySQL 5.0. The resource allocation to a RUBiS VM is dynamically controlled by the FMPC-based node controller. The client VMs, which generate workloads to the RUBiS VMs, are hosted on separate physical machines and they can launch up to 8000 emulated client sessions in total. To create high CPU contentions, another benchmark, FreeBench [4], which models computationally intensive jobs, was also used in the experiments.

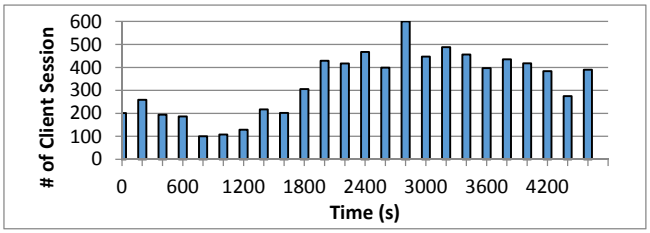
The control period of the *Node Controllers* is 20 seconds, during which a controller updates its local VMs' performance models (under 1s) and optimizes the resource allocations to VMs (under 15s). The control period of the *Global Scheduler* is one minute, during which it gathers the resource demands from all *Node Controllers*, decides VM migrations, and coordinates the involved controllers to execute the migrations.

B. Application-Level QoS Target Tracking

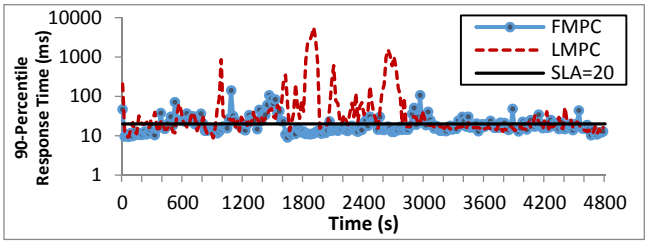
The first experiment evaluates the ability of the FMPC-based controller in tracking fine-grained, response-time-based QoS target for a multi-tiered application (RUBiS) that services a dynamic workload. The workload represents real-world workload patterns as its intensity (the number of concurrent client sessions) is varied following a real daily trace collected from a departmental web server at FIU [18]. The trace captures the number of requests per hour, which is scaled up to the range that the RUBiS setup can handle (Figure 3(a)). When replaying the trace on RUBiS, it is accelerated by 30 times so that a whole day trace can finish in 2880 seconds. The QoS target is set to 20ms in terms of the 90th-percentile response time which is a reliable metric for measuring the Internet service quality [19].

This experiment is challenging in that, first, the QoS target is the average request response time per control period, which is highly sensitive to the accuracy of VM performance model, and second, the workload is highly dynamic which requires good speed and adaptability in the performance modeling.

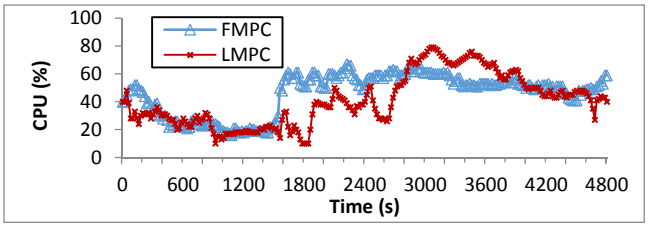
The experiment compares the proposed FMPC approach to the adaptive linear MPC (LMPC) approach studied in the related work [20]. In the FMPC approach, the predicted performance is assumed to be dependent on only the current resource allocation, so Equation (1) is simplified as $R_i: If\ \boldsymbol{u}(t)\ is\ \boldsymbol{A_i},\ then\ \boldsymbol{y_i}(t) = \boldsymbol{a_i}\boldsymbol{u}(t) + \boldsymbol{b_i}$. In Equation (3), both the input and output vectors $\boldsymbol{u}$ 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 to balance the importance between tracking accuracy and controlling stability. The



(a) A real-world trace used to generate the RUBiS workload

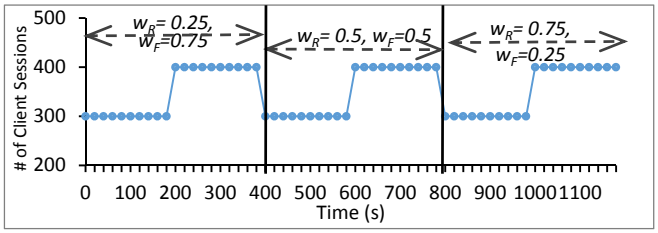


(b) Performance for the RUBiS workload

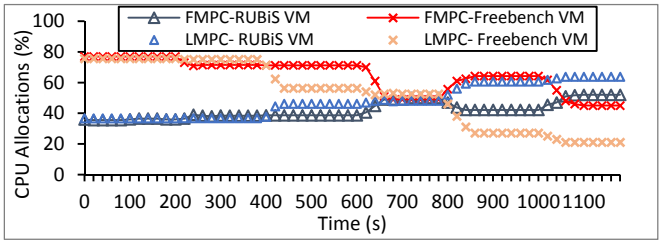


(c) CPU allocations to the RUBiS VM

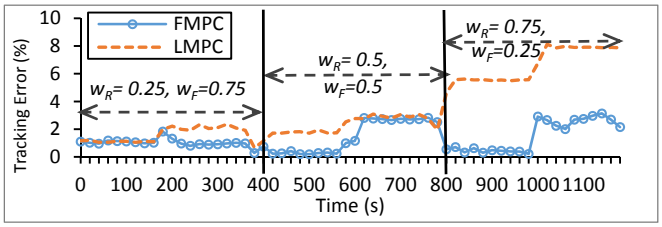
Figure 3. Meeting the QoS target of a RUBiS VM



(a) Changing workload and changing weights to the VMs



(b) Average CPU allocation to each group of VMs



(c) Level of QoS violation (weighted sum of the normalized performance errors) across all VMs

Figure 4. Optimizing resource allocation to several co-hosted VMs when both the workloads and weights are changing

baseline LMPC leverages a linear auto-regressive-moving-average (ARMA) model which automatically trains the linear VM performance model and is able to adapt the model based on the online training. For both approaches, once 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 with the first few data points and afterwards it is updated every control interval.

Figures 3(b) and (c) show the performance of RUBiS and the CPU allocation to the VM every control interval. Both FMPC and LMPC can meet the QoS target eventually as workload changes, but FMPC is able to meet the QoS target more closely and be more responsive to the changes, especially when the workload intensity rises to its peak, from 1600s to 1800s. During this period, LMPC suffers from large fluctuations and the performance is substantially worse than FMPC; the average response is 109ms for LMPC and 13ms for FMPC. This large difference is due to the underestimation of CPU demand by LMPC during this period, while FMPC is able to correctly predict the increasing CPU demand when the RUBiS workload intensifies. Owing to the speed of fuzzy modeling, FMPC can quickly adapt to the workload changes and closely follow the QoS target. Overall, the average response time across the entire workload is 138.3ms for LMPC and 21.0ms for FMPC.

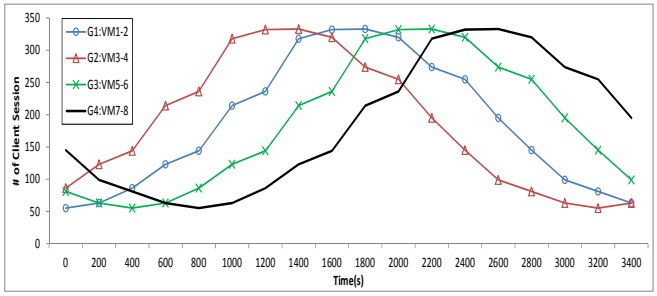
C. Host-level Resource Management

The second group of experiments evaluates how the proposed FMPC controller manages the resource allocations among multiple VMs on the same host in order to optimize host-level management objective and how it reacts to the dynamic changes in management policy.

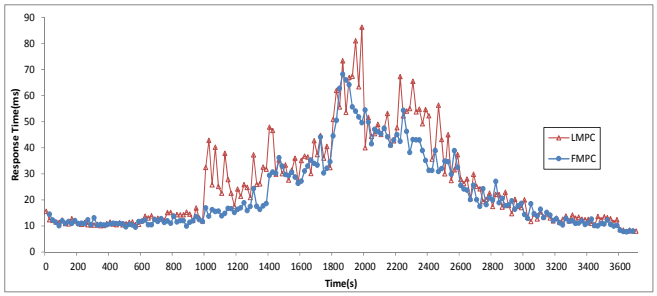
1) Changing Workloads with Changing Weights

Two different benchmarks are used in this experiment, including RUBiS and FreeBench. A total of 12 VMs share six physical CPU cores, and each VM is configured with one virtual CPU and 1G of RAM and runs one benchmark. Eight of the VMs run the multi-tiered RUBiS and the other four VMs run FreeBench. The entire experiment lasts for 1200s. All the RUBiS VMs service the same browsing mix workload with varying intensity as illustrated in Figure 4(a). All the FreeBench VMs run iterative, computationally intensive tasks continuously. In addition to varying the RUBiS workloads, the weights assigned to the VMs are also changed over time, as shown in Figure 4(a). The VMs that host the same application are treated equally; w_R is the weight for the RUBiS VMs, and w_F is the weight for the Freebench VMs. The experiment can be then divided into three phases according to the weight values, and for each phase the workload intensity of RUBiS VMs increases from 300 to 400 client sessions. The QoS target is 20ms response time for RUBiS and 0.8s loop time for FreeBench. To make the performance of different applications comparable, the actual performance measurements are normalized into the same range.

Figure 4(b) compares the online resource allocations made by FMPC vs. LMPC. For clarity, the average value of CPU allocations to VMs that run the same application is shown for every control interval. Figure 4(c) compares *the weighted sum*



(a) The workload trace for all eight RUBiS VMs



(b) Average 90th-percentile response time for all VMs

Figure 5. Optimizing resource allocation to several co-hosted VMs serving realistic workloads

of the normalized performance errors, $\sum_i w_i|y_i(t+1) - y_{refi}|$ achieved by FMPC and LMPC. This metric reflects the total performance discrepancy from the QoS target vector $\boldsymbol{y_{ref}}$, which should be minimized by the controller.

At the beginning of the first phase, FMPC and LMPC make similar allocation decisions, giving more CPU to the FreeBench VMs which have a higher weight than the RUBiS VMs. But as the RUBiS workload increases, FMPC increases the CPU allocations to the RUBiS VMs by shifting a total of 16% CPU allocations from the FreeBench VMs, while LMPC does not recognize this need and its allocation decision is almost unchanged. Consequently, LMPC has much higher performance errors, 63.7% in average, than FMPC. When the experiment transits to the second phase, both the weights and the RUBiS workloads are changed. FMPC handles these changes much better than LMPC, and results in 78.3% lower performance error in average for the first half of this phase. In the second half of the phase, both controllers enter the steady state, FMPC is still 8.9% better than LMPC in average. The difference between these two approaches is even more drastic in the third phase. At the beginning of this phase, both controllers favor the FreeBench VMs because of their higher weights. As the workload intensifies for the RUBiS VMs, FMPC increases their allocations which eventually surpass the FreeBench VMs, whereas LMPC continues to favor the FreeBench VMs. This opposite decision causes LMPC to perform substantially worse (up to 11 times higher performance errors) than FMPC.

2) Realistic workload

The second experiment evaluates both the scalability and stability of the proposed FMPC approach in managing more VMs under realistic workloads with more dynamic changes. In this experiment, eight VMs share four physical CPU cores, and they all run RUBiS using the same real-world web trace described in Section VI.B. To make the experiment more interesting, the VMs are divided into four groups, and each group starts the replay from a different offset of the trace, as shown in Figure 5(a). As a result, the four groups reach their peaks and values at different times in the experiment, and the total load of the VMs also varies over time. In this experiment, equal weight and QoS target (15ms) are set for all the VMs. Note that when the system is saturated, none of the VMs can meet its QoS target under equal resource allocations. However, this experiment focuses on how to optimize the overall performance by minimizing the distance to the VMs' QoS targets.

Figure 5(b) compares the overall performance achieved by FMPC vs. LMPC using the average 90th-percentile response time as the metric because all the VMs have the same QoS target and weight. At the beginning and the end of the experiment, the overall system load is low and as a result there is not much difference in performance between FMPC and LMPC. But when the system becomes more loaded, FMPC outperforms LMPC significantly. For example, from 1000s to 1600s, while LMPC achieves an average response time of 29.2ms and causes serious QoS violations (w.r.t. the 15ms target), FMPC still maintains a good performance (17ms average response time). From 1800s to 2600s, when the system is saturated, FMPC delivers a 15.6% better overall performance in average response time than LMPC.

D. System-level Resource Management

The last group of experiments evaluates the scalability of the proposed two-level cloud resource management framework using a larger testbed. The setup used in the previous experiment is extended from single host to six hosts, each initially running eight RUBiS and nine FreeBench VMs. There are a total of 102 VMs under the management of a global scheduler and six node controllers. An additional six client VMs are used to generate the workloads for the RUBiS VMs. The traces for the RUBiS VMs are created similarly to the previous experiment, where all the RUBiS VMs are divided into six groups and each group starts the replay from different offset of the trace.

This experiment is designed to evaluate the ability of the two-level resource management to use dynamic VM migrations to optimize the overall performance across hosts. The baseline uses only the FMPC-based node controllers but without VM migrations. Figure 6(a) shows the level of QoS violations—the weighted sum of the normalized performance errors—occurred on every host over time using heat map. The *x*-axis shows the time in seconds and the *y*-axis shows the host ID. The gray shades represent different levels of QoS violations (the darker the worse), whereas the white color indicates when all the VMs' QoS targets are met. The results show that the use of VM migration substantially improves the performance of the VMs across the entire system. Overall, the average performance across all the VMs in the system is improved by 23.7% compared when migration is not used. This improvement is made possible by the *Global Scheduler* which decides VM migration based on the resource demands estimated using FMPC, and by the node controllers which

cooperate to migrate the VMs and their performance models. Figure 6(b) also uses a heat map to illustrate the distribution of the VMs over time when migration is employed to balance the load across hosts. The gray shades in the legend represent the number of VMs on a host.

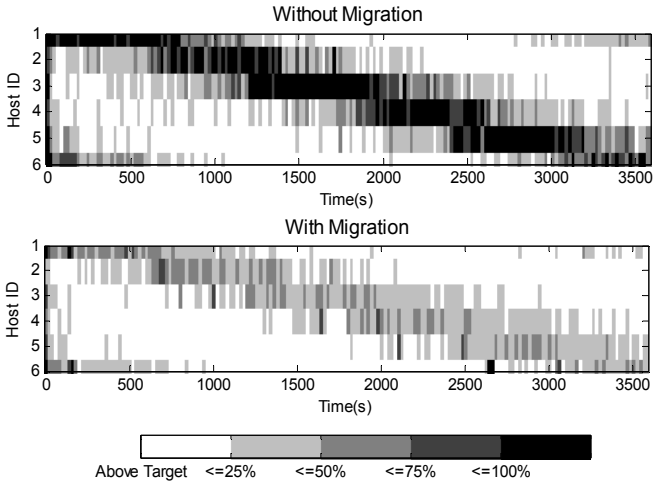
VII. Related Work

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 approach considers machine-learning-based techniques to automatically learn the complex resource model for a virtualized system based on data observed from the system. For example, Xu *et al.* studied the use of fuzzy modeling to predict the CPU demand of a single VM [21][22]; the CRAVE project used regression analysis to predict the performance impact of memory allocation to VMs [23]; The VCONF project studied using reinforcement learning to automatically tune the CPU and memory configurations of a VM in order to achieve a good performance for its hosted application [24]; and Kund *et al*. used artificial neural networks and support vector machines to build offline performance models considering both resource allocation to VMs and resource interference between VMs [7][8].

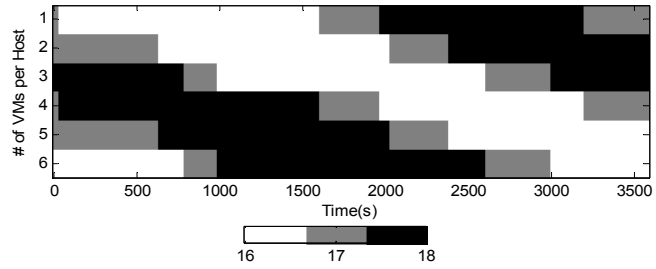
Another type of approach applies control theory [13] to automatically adjust VM resource allocation in order to achieve the desired system-level objective. In particular, linear MIMO-model-based MPC (LMPC) has been studied for resource management where multiple applications share a common pool of resources. For example, Padala *et al*. [20] used an online-trained linear MIMO model to capture the relationship between the resource allocations to multi-tiered applications and their performance, and designed controllers to optimize the resource allocations by minimizing the estimated distance to the QoS targets; and Gupta *et al*. studied the use of LMPC to capture the process cache interference among VMs and compensate its performance impact [25].

The FMPC-based approach proposed in this paper combines the strengths of machine-learning and control-theory techniques in cloud 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 behaviors change. Compared to traditional LMPC-based approaches, FMPC can well capture nonlinear system behaviors without much learning overhead. In LMPC, 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. As demonstrated in Section VI, FMPC can more accurately model the system with a nonlinear fuzzy model and it can perform optimized control continuously over the whole operating space.

While this paper focuses on the problem of how much resources to provision to an applications' single VM, a.k.a., *vertical scaling*, there is also related work studying how many



(a) Level of QoS violation (weighted sum of the normalized performance errors) across hosts



(b) Placement of VMs across hosts

Figure 6. Optimizing resource allocation to over 100 VMs across multiple hosts using both dynamic resource allocations and VM migrations

VMs to provision to an application, a.k.a., *horizontal scaling*. Gandi *et al*. studied a workload discretization technique for predicting workloads, and then used a queuing model to predict the number of needed servers [26]. But queuing models are often inadequate to model the complexity of cloud systems, so they had to also use a feedback controller to compensate for the error in the prediction. AGILE [27] used wavelets to directly predict the number of needed application server instances in the medium term, and to capture the near-term resource demand they used curve fitting by trying polynomials with different orders—which could be more efficiently and accurately captured by a fuzzy model, as shown in another work [6]. AutoScale [28] used the number of requests in the system to infer the number of needed servers based on a model that maps the number of requests on a single server to the load at the server, which was shown to be nonlinear and could also be well captured by a fuzzy model. Moreover, the above related works do not address how to optimize the resource allocation to competing applications, which can be solved by the predictive controller proposed in this paper's FMPC approach.

VIII. Conclusions 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 according to the

application-level QoS targets and 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. Based on FMPC, the paper also proposes a two-level cloud resource management framework. The node controllers work on the VM host level to estimate VM resource demands and optimize each host's resource allocations. The global scheduler works at the cloud zone level to optimize resource utilization across hosts through dynamic VM migrations. The global scheduler cooperates with the node controllers to understand the host-level resource demands and carry out the VM migration plans. The node controllers involved in a migration also work together to share the performance models for the migrated VMs and minimize the migrating impact.

Future work will be explored along the following directions: (1) consider the holistic management of different types of VM resources based on this paper's approach and the authors' earlier work which considered the management of memory and IO resources [6][29]; (2) study the coordinated management of VMs belonging to the different tiers of the same application, a more common setup used by cloud applications; and (3) apply this paper's approach to the management of horizontal scaling of VMs serving the same tier of a cloud application.

ACKNOWLEDGEMENT

The authors thank the anonymous reviewers and Giuliano Casale for their helpful comments, and Eric Johnson for assisting the collection of traces. This research is sponsored by National Science Foundation under the National Science Foundation CAREER award CNS-1253944, the Department of Defense award W911NF-13-1-0157, and the Department of Homeland Security award 2010-ST-062-000039.

REFERENCES

[1] Amazon Elastic Compute Cloud (Amazon EC2), URL: http://aws.amazon.com/ec2/.

[2] Windows Azure Platform, URL: http://www.microsoft.com/windowsazure/.

[3] C. Amza, A. Chanda, A.L. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet and J. Marguerite, "Specification and Implementation of Dynamic Web Site Benchmarks," Proceedings of the IEEE International Workshop on Workload Characterization, 2002

[4] Freebench, URL: https://code.google.com/p/freebench/

[5] 1998 World Cup Web Site Access Logs, URL: http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[6] L. Wang, J. Xu, M. Zhao, Y. Tu, and J. Fortes, "Fuzzy Modeling based Resource Management for Virtualized Database Systems," Proceedings of the 19th Annual Meeting of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS2011), July 2011.

[7] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao, "Application Performance Modeling in a Virtualized Environment," Proceedings of the 16th IEEE International Symposium on High-Performance Computer Architecture (HPCA-16), 2010.

[8] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta, "Modeling Virtualized Applications using Machine Learning Techniques," Proceedings of the 8th Annual International Conference on Virtual Execution Environments (VEE2012), March 2012.

[9] T. Takag and M. Sugeno, "Fuzzy identification of systems and its application to modeling and control," IEEE Transactions on Systems, Man and Cybernetics, (1), 116-132, 1985.

[10] HP-UX Workload Manager, http://docs.hp.com/en/5990-8153/ch05s12.html

[11] J. Rolia, L. Cherkasova, and C. McCarthy, "Configuring Workload Manager Control Parameters for Resource Pools," Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium, 2006.

[12] J. Maciejowski, "Predictive Control with Constraints," Prentice Hall, 1 edition, 2002.

[13] T. Abdelzaher, Y. Diao , J. Hellerstein , C. Lu, and X. Zhu, "Introduction to Control Theory and its Application to Computing Systems, Performance Modeling and Engineering", Springer, 2008.

[14] S. Chiu, "Fuzzy Model Identification Based on Cluster Estimation," Journal of Intelligent and Fuzzy Systems, 1994.

[15] D. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning," Kluwer Academic Publishers, Boston, MA, 1989.

[16] J. Xu and J. Fortes, "A Multi-objective Approach to Virtual Machine Management in Datacenters," Proceedings of 8th International Conference on Autonomic Computing (ICAC), 2011.

[17] J. Xu and J. Fortes, "Multi-objective Virtual Machine Placement in Virtualized Data Center Environments," Proceedings of 2010 IEEE/ACM International Conference on Green Computing and Communications (GreenCom), 2010.

[18] FIU SCIS website trace. URL: https://visa.cs.fiu.edu/traces

[19] M. Kallahalla, M. Uysal, R. Swaminathan, D. Lowell, M. Wray, T. Christian, N. Edwards, C. Dalton, and F. Gittler, "SoftUDC: A Software-based Data Center for Utility Computing," Computer, 2004.

[20] P. Padala, K. Hou, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated Control of Multiple Virtualized Resources," Proceedings of ACM European conference on Computer systems, 2009.

[21] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, "On the Use of Fuzzy Modeling in Virtualized Data Center Management," Proceedings of the 4th International Conference on Autonomic Computing (ICAC2007), June 2007.

[22] 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, Vol. 11, No. 3, Pages: 213-227, September 2008.

[23] J. Wildstrom, P. Stone, and E. Witchel, "CARVE: A Cognitive Agent for Resource Value Estimation", Proceedings of International Conference on Autonomic Computing, 2008.

[24] J. Rao , X. Bu, C. Xu, L. Wang, and G. Yin, "VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration", Proceedings of International Conference on Autonomic Computing, 2009.

[25] R.Nathuji and A. Kansal, "Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds," Proceedings of the 5th ACM European conference on Computer systems, 2010.

[26] A. Gandhi, Y. Chen, D. Gmach, M. Arlitt, and M. Marwah, "Minimizing Data Center Sla Violations and Power Consumption via Hybrid Resource Provisioning," Proceedings of International Green Computing Conference and Workshops (IGCC), 2011.

[27] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "Agile: Elastic Distributed Resource Scaling for Infrastructure-As-A-Service," Proceedings of the USENIX International Conference on Automated Computing (ICAC'13), 2013.

[28] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. Kozuch, "AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers," ACM Transactions on Computer Systems, 30, 4, Article 14, November 2012.

[29] L. Wang, J. Xu, and M. Zhao, "Application-aware Cross-layer Virtual Machine Resource Management," Proceedings of the 9th International Conference on Autonomic Computing (ICAC2012), September 2012.