Fuzzy Modeling Based Resource Management for Virtualized Database Systems

Lixi Wang*, Jing Xu†, Ming Zhao*, Yicheng Tu‡, José A. B. Fortes†
* Florida International University, {lwang007, mzhao}@fiu.edu
† University of Florida, {jxu, fortes}@ufl.edu, ‡ University of South Florida, ytu@cse.usf.edu

***Abstract* — The hosting of databases on virtual machines (VMs) has great potential to improve the efficiency of resource utilization and the ease of deployment of database systems. This paper considers the problem of on-demand allocation of resources to a VM running a database serving dynamic and complex query workloads while meeting QoS (Quality of Service) requirements. An autonomic resource-management approach is proposed to address this problem. It uses adaptive fuzzy modeling to capture the behavior of a VM hosting a database with dynamically changing workloads and to predict its multi-type resource needs. A prototype of the proposed approach is implemented on Xen-based VMs and evaluated using workloads based on TPC-H and RUBiS. The results demonstrate that CPU and disk I/O bandwidth can be efficiently allocated to database VMs serving workloads with dynamically changing intensity and composition while meeting QoS targets. For TPC-H-based experiments, the resulting throughput is within 89.5–100% of what would be obtained using resource allocation based on peak loads; For RUBiS, the response time target (set based on the performance under peak-load-based allocation) is met for 97% of the time. Moreover, substantial resources are saved (about 62.6% of CPU and 76.5% of disk I/O bandwidth) in comparison to peak-load-based allocation.**

I. INTRODUCTION

Virtual machines (VMs) [1][2] can be powerful platforms for hosting database systems. For database service providers, VMs allow fine-tuned databases to be encapsulated along with their execution environments and easily deployed as appliances on different systems. For resource owners, VMs support flexible resource allocation to both meet database demands and share resources with other applications. Virtualization is also the enabling technology for the emerging cloud computing paradigm [3][4], which further allow highly scalable and cost-effective database hosting leveraging its elastic resource availability and pay-as-you-go economic model. However, due to the highly complex and dynamic nature of database systems, it is still challenging to efficiently host them using virtualized resources. Typical databases have to serve dynamically changing workloads consisting of a variety of queries and consuming different types and amounts of resources. This makes it difficult to host databases on shared resources without compromising Quality of Service (QoS) or wasting resources.

This paper addresses this problem through an *autonomic* resource management system for virtualized databases. The goal of this system is two-fold. First, it should be able to automatically learn a database VM's need of multi-type resources for servicing a complex query workload, so that resources can be efficiently allocated while satisfying the desired QoS. Second, it should be able to automatically adapt to dynamic changes in the database VM's behavior and timely adjust the resource allocation, so that both resource efficiency and query QoS can be sustained.

To achieve this goal, this paper proposes to use *fuzzy modeling* to learn and predict a database VM's multi-type resource need based on the observed query workload and resource usages. This method does not require any a priori knowledge of the system's internal structure and it can efficiently describe complex and nonlinear system behaviors. To effectively capture a complex query workload and supply it as the input for accurate modeling and prediction, this paper proposes a novel *workload characterization* method to capture both the intensity and composition of a workload by leveraging the database's internal query cost estimation. Both the workload characteristics and the VM's fuzzy model can be learned and updated *online*, adapting to the dynamic changes in the system in a timely fashion.

This proposed system is prototyped on Xen-based VM environments and evaluated by experiments using typical database workloads based on TPC-H [5] and RUBiS [6] benchmarks. The results show that the system can accurately and efficiently allocate *CPU* and *disk I/O bandwidth* at *fine granularity* (every 10s) for a database VM that is serving CPU and I/O intensive queries while delivering the same level of QoS as using peak-load-based resource allocation. The results also show that the system can quickly adapt to dynamic changes in the database VM's resource needs caused by changing workload intensity and composition. For the TPC-H based experiments, the obtained throughput is within 89.5–100% of the peak-load-based allocation. For the RUBiS-based experiments, the system can meet the QoS target (set based on the performance under peak-load-based allocation) for 97% of the time. In the meantime, it saves substantial resources (about 62.6% CPU and 76.5% disk I/O bandwidth) compared to peak-load-based allocation.

In summary, this paper's main contribution is a novel autonomic system for efficiently allocating resources to virtualized database serving dynamic and complex query workloads. To the best of our knowledge, this paper is the first to study fuzzy modeling for virtualized applications with dynamic, multi-type resource needs. Compared to existing solutions, this approach can effectively capture nonlinear resource usage behavior, timely adapt to dynamic changes in such behavior, and optimize VM resource allocation continuously. In the rest of the paper, Section II introduces the background and motivation, Section III and IV present the fuzzy modeling approach and the resource management

system, Section V discusses the evaluation, Section VI examines related work, and Section VII concludes the paper.

II. BACKGROUND AND MOTIVATION

A. Virtualized Database Hosting

Traditionally, databases are hosted on dedicated physical servers that have sufficient hardware resources to satisfy their expected peak workloads with desired QoS. However, this is often inefficient for the real-world situations in many application domains such as e-business [7] and stream data management [8], where the workloads are intrinsically dynamic in terms of their bursty arrival patterns and ever-changing unit processing costs. Even in domains where static workload exists, the database can dynamically switch from one workload to another at runtime. For example, an online vendor database that serves large number of user queries during the day may switch to internal bookkeeping jobs early in the morning. Consequently, peak-load based resource provision often leads to underutilization of resources for normal state workloads and causes substantial overhead.

Using VMs to host databases can effectively address this limitation, because virtualized resources, including CPU, memory, and I/O, are decoupled from their physical infrastructure and can be flexibly allocated to the databases as needed. This approach allows a database to transparently share the consolidated resources with other applications, with strong isolation between their dedicated VMs. It also allows a database's resource usage to elastically grow and shrink based on the dynamic demand of its workload. Such benefits are important to the efficiency of database hosting in both typical data centers and emerging cloud systems. On one hand, users only need to pay for the resources that their databases actually consume and the QoS that their workloads actually get. On the other hand, resource providers only need to allocate resources as required by the database VMs while saving valuable resources for hosting other applications.

Virtualization also offers a new paradigm for database deployments. Modern databases are sophisticated software systems, where their installation and configuration require substantial domain knowledge and experience as well as considerable efforts from the database administrators. VM-based database hosting allows carefully installed databases to be distributed as simply as copying the data that represent the database VMs. In addition, this approach allows databases to be quickly replicated and distributed for performance and reliability improvements.

B. Autonomic VM Resource Management

The main challenges to the success of VM-based database hosting are how to efficiently allocate resources to VMs and how to do so automatically and continuously. This paper addresses these challenges with an online autonomic resource management system that can automatically allocate resources to VMs based on their dynamic workload demands and QoS requirements. A key task of such an autonomic system is to understand a VM's resource need for its current workload. This task is particularly challenging for database VMs because of their highly complex and dynamic resource usage behaviors. Database queries can be both CPU and I/O intensive and a typical database workload can have a diverse variety of such queries with dynamically changing composition. Hence, the resource usage model of a database VM is typically multi-dimensional and nonlinear, as evidenced by the experiment results presented in Section V.

To understand a database VM's resource need, this paper employs machine learning techniques to capture the relationship between the workload demand and VM resource usage. This approach is advantageous than others in that it creates a VM's resource usage model automatically from data observed from the system without assuming any a priori knowledge about the system's structure (detailed discussion in Section VI). Specifically, this paper proposes to use *fuzzy-logic based modeling* for database VMs, because it is suited to efficiently model systems with complex behaviors [9]. Although our previous work successfully applied fuzzy modeling to control CPU allocation for VMs hosting CPU-intensive applications [10], the management of database VMs raises unique, important research questions: *1)* How to effectively manage a VM with *correlated*, *multi-type* resource need, including not only CPU cycles but also I/O bandwidth? 2) How to timely adapt to the dynamic changes in a VM's resource need in terms of not only *varying intensity* but also *shifting demand across different resource types*? These are the questions addressed by this paper, in which databases serve as an excellent example of applications with dynamic and complex behaviors.

III. FUZZY-LOGIC BASED DATABASE VM MODELING

A. Fuzzy-logic based Modeling

Fuzzy modeling combines fuzzy logic with mathematical equations to describe the discovered patterns of system behavior and to guide the control strategies of the system [11]. 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 determined by fuzzy sets with associated membership functions. Contrast to a crisp set, a fuzzy set allows partial set memberships which can be quantified into numeric values based on a membership function. Commonly used membership functions are *Gaussian*, *Sigmoidal*, *Triangular*, *Trapezoidal* function, etc. The fuzzy rules in a fuzzy model are trained using the input (x) and output (y) observed from the system and together they compose the model representing the system behavior.

While building a fuzzy model, *data clustering* techniques (e.g., [12]) 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. For example, the model for a database VM from the experiment discussed in Section V.B.2) is as follows,

R^1: If $[C_1, C_2]^T$ is in $cluster_1$, then $r_{CPU} = [8.8\ 6.3][C_1, C_2]^T + 3.1$
R^2: If $[C_1, C_2]^T$ is in $cluster_2$, then $r_{CPU} = [-0.5\ 1.5][C_1, C_2]^T + 88$
R^3: If $[C_1, C_2]^T$ is in $cluster_3$, then $r_{CPU} = [12.8\ 0.5][C_1, C_2]^T + 41$

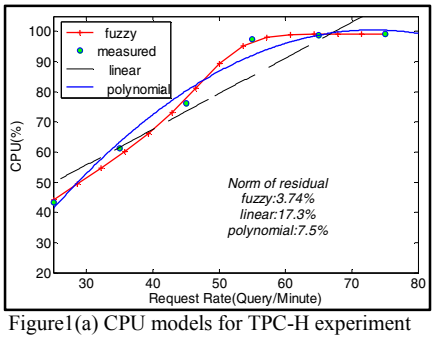

Figure1(a) CPU models for TPC-H experiment

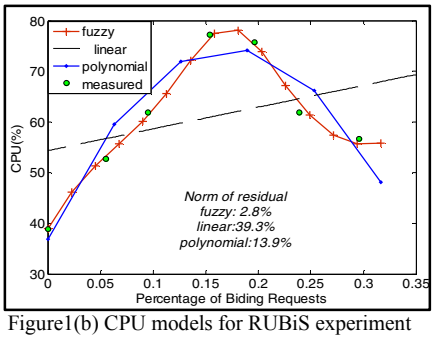

Figure1(b) CPU models for RUBiS experiment

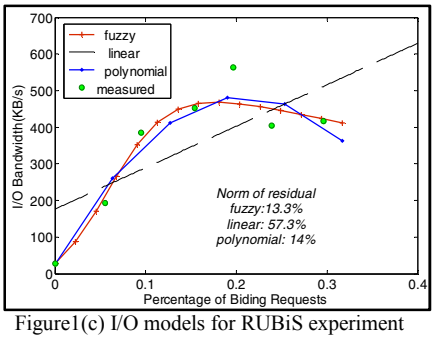

Figure1(c) I/O models for RUBiS experiment

R^4: If $[C_1, C_2]^T$ is in $cluster_4$, then $r_{CPU} = [8.3\ 2.1][C_1, C_2]^T - 68$

The input of the model is the query workload described by a vector of request rates of two types of queries, $[C_1\ C_2]^T$, while the output of the model is the CPU resource usage, r_{cpu}. Given a total of 225 input-output data pairs measured in the experiment, clustering technique is used to produce only 4 clusters which can effectively represent the entire dataset. Each cluster is then treated as a fuzzy set and associated with a fuzzy rule as part of the database VM's 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. Following the above example, given a specific workload input $[C_1, C_2]^T$, the fuzzy model learned from the TPC-H based experiment can be used to predict the CPU demand r_{cpu} following the above steps.

Note that this fuzzy-modeling approach is fundamentally different from traditional rule-based system management approach [13][14]. The latter is based on the use of a set of event-condition-action rules that 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.

B. *Modeling the Non-linearity in Database VMs*

The major difficulty of online resource management for a virtualized database system lies in how to model its intrinsically dynamic and complex behavior in an accurate and efficient way. Commonly used linear modeling methods are no longer sufficient for modeling such a system whose workload consists of different queries with diverse usage of multiple types of resources. Either the bursty arrivals of queries or the transitions between different types of queries in the workload may lead to more complex behavior of the virtualized database. Here we use several concrete examples to demonstrate the nonlinearity in a database VM's resource usage behavior and the advantage of fuzzy modeling.

In the first example, a synthetic database workload based on a sequence of TPC-H [5] queries is executed on a database VM. We gradually increase the workload intensity by adjusting the query request rate until the virtualized database becomes saturated. Figure 1(a) plots the observed average CPU usage of the database VM as the request rate is increased from 35 to 75 request/minute. The nonlinearity in such an OLAP database is evident as the request rate exceeds around 55 query/minute and the system becomes saturated.

The second example considers a typical multi-tier OLTP benchmark, RUBiS [6]. We fix the database tier's query workload intensity by running 1000 concurrent client sessions in RUBiS. But we vary the composition of the query workload by increasing the ratio of bidding and browsing requests to the web tier which correspond to read and write queries, respectively, to the database tier. Nonlinearity is apparent in the CPU and disk bandwidth usages (Figure 1(b) and (c)) of such an OLTP database's behavior, even though the system is not under saturation.

We then study the accuracy of applying fuzzy modeling to the database VMs in the above two examples and compare it to another two commonly used modeling methods, the simple linear regression and the more complex second-order polynomial fitting. The models created by these different methods along with the measured data are shown in Figure 1. The figure also shows their *norm of the residuals*, a common metric for evaluating the goodness of a model, which is defined as the square root of the sum of the squares of the differences between the predicted values and the actual values. The results show that linear model (*linear*) poorly fits the data points; the polynomial model (*polynomial*) can only reflect the trend of the resource need but cannot predict accurately the amount of necessary resources; only the fuzzy model is accurate regarding the entire data set which represent the complete resource usage behaviors of the database VMs. As we will further demonstrate in Section V, our proposed fuzzy-modeling-based approach outperforms others in terms of its accuracy and efficiency.

Note that such modeling-based resource management is different from a typical feedback-control-based approach in which the application's actual performance is used to directly adjust the resource allocation in order to achieve the QoS

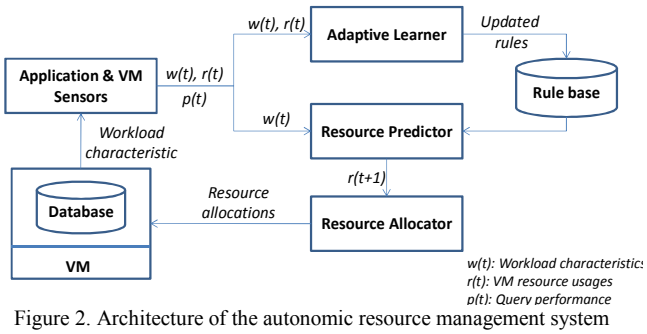

Figure 2. Architecture of the autonomic resource management system

target. In our modeling-based approach, a model is first built to capture the relationship between the application workload and its resource needs for the QoS target, and then used to predict the necessary resource allocation for the current workload demand. Although fuzzy-logic-based feedback controllers also exist [15], the key difference between our approach and those still lies in the fact that fuzzy logic is used to build a model for the managed system instead of to directly decide how to control the system.

IV. AUTONOMIC DATABASE VM MANAGEMENT

Figure 2 illustrates the architecture of our proposed resource management for virtualized databases based on the aforementioned fuzzy modeling approach. This system consists of four key modules. As a workload executes on the database VM, the *Application and VM Sensors* monitor the workload $w(t)$, its performance $p(t)$, and the VM's resource usage $r(t)$. With this model and the current workload $w(t)$, the *Resource Predictor* estimates the resource need for time $t+1$ and the *Resource Allocator* adjusts the allocation accordingly. Together, these modules form a closed-loop for the database VM's resource control and optimization. They are described in detail in the rest of this section.

A. Application and VM Sensors

1) Workload Characterization

Workload characterization is the process of describing the characteristics of a workload that is relevant to its resource usage behaviors when executed on a database VM. Such characteristics provide important inputs to the effective modeling and prediction of a database VM's resource needs. A commonly used workload characteristic is the request rate which describes the workload's overall intensity and is often strongly correlated with its resource demand. However, the characterization of a database workload is more challenging, because it can consist of different queries with diverse use of multiple types of resources.

To address this challenge, we propose to characterize a database workload by first classifying its queries into a small number of groups based on their resource usage patterns and then describing the whole workload as a vector of arrival rates of these groups. Specifically, for Online Transaction Processing (OLTP) workloads which consist of small transaction queries, these queries can be simply classified into two groups based on whether there are intensive write operations to the database. These two groups typically have distinct demands for CPU and I/O resources. However, for Online Analytical Processing (OLAP) workloads, their query statements are more complex and difficult to differentiate based on only their query patterns. We propose to classify these queries by leveraging the query cost estimation provided by the database system.

A database's query optimizer has intimate knowledge of a query's execution plans and their incurred costs of different types of resources. Such knowledge can be extracted from the database and used to classify queries for workload characterization. Typically, the query cost is defined as a function of the amount of resource usages estimated by the database, which can be extracted as a vector of different resource costs. Given such a vector for each query in the workload, data clustering [12] is used to group queries based on their costs and derive a concise representation of the entire workload. Note that the database's cost estimation cannot be directly used to infer its VM's resource needs because, first, its accuracy is often limited [16], and second, it does not capture the entire VM's resource needs.

The above workload characterization methods are applied online periodically, in order to reflect the workload's current characteristics and used as input to the Adaptive Learner described below for modeling the database VM's current behavior. Note that in this paper, the workload of current time step t is used as the prediction of the workload of the next time step $t+1$ based on the assumption that no sudden change happened within one period of time. Therefore, $w(t)$ is also used as the input for the Resource Predictor discussed below to estimate the resource demand $r(t+1)$. In our future work, we will consider more advanced workload prediction using forecasting methods.

2) Resource Monitoring and Performance Measurement

The VM Sensor monitors a VM's resource consumption, which is the other key piece of information for modeling the VM's resource usage behavior. The monitoring has to be done outside of the VM, because the application's resource usage inside of the VM does not truthfully represent its entire VM's resource usage which entails overhead from both the guest operating system and the use of virtualization. The VM Sensor in our system monitors multiple types of resources including CPU, memory, and disk and network I/Os, as a database VM can make intensive use of multi-type resources.

In addition to the information about application workload and VM resource usage, the proposed system also needs to monitor the application's current performance, in order to determine whether the current resource allocation can meet the desired QoS. This measurement is also done by the Application Sensor, using the typical performance metrics such as throughput and response time. Note that we consider a workload as a *continuous*, *dynamic* process. Therefore, the performance reported by the Application Sensor is fine-grained, periodically taken measurements (e.g., every 10s), rather than the overall value measured only once for the entire workload. The Application Sensor can be generally implemented as a query proxy that is inserted between the database client and server, so it can forward queries to the database and meanwhile measure their performance.

B. Adaptive Learner and Resource Predictor

The *Adaptive Learner* creates and updates the model that represents the relationship between a database workload and its VM's resource need. It employs the fuzzy modeling approach to automatically discover this relationship, where fuzzy rules are constructed based on the input and output data pairs, $\langle w(t), r(t)\rangle$, collected by the Application and VM Sensors. Both the workload input $w(t)$ and the resource usage output $r(t)$ can be vectors with multiple dimensions. For $w(t)$, each dimension represents a certain characteristic of the workload and for $r(t)$ each dimension maps to one type of resources. In order to learn a model that represents the resource needs of the VM for a specific QoS target, only qualified input-output data pairs $\langle w(t), r(t)\rangle$ whose workload performance $p(t)$ meet the QoS target are fed to the Adaptive Learner. In this way, the resulting model trained based on the filtered data can capture the VM's resource needs in order to meet the given QoS target. When the QoS target changes, the model will be different as the qualified training data change.

While creating a fuzzy rule base from the qualified input-output data, it is inefficient to generate one rule for every specific data pair. In order to build a concise fuzzy rule base with a small number of rules that can still effectively represent the VM's behavior, a clustering method is used to group similar data points into clusters. In particular, the Adaptive Learner adopts an efficient one-pass clustering algorithm, subtractive clustering [12]. Each resulting cluster exemplifies a representative characteristic of the system behavior and can be used to create a fuzzy rule accordingly.

The Adaptive Leaner generates Sugeno-type fuzzy rules [11] from the clustered data for modeling the database VM. This type of fuzzy rules uses a crisp, linear or constant function as the membership function, which is suitable for mathematical analysis. Suppose for input the workload $w(t)$ is described by N different characteristics, $[C_1, C_2, \ldots, C_N]$ and for output, two types of resources, CPU and I/O, $[R_{CPU}, R_{IO}]$, are consumed. If K clusters are formed from all the data pairs, then K rules are produced for this fuzzy model. The rule base is constructed as follows:

R_i: IF input $[C_1, C_2, \ldots, C_N]$ is in cluster i,
THEN output $[R_{CPU}, R_{IO}]^{\mathrm{T}} = A_i[C_1, C_2, \ldots, C_N]^{\mathrm{T}} + b_i$

Each fuzzy rule is generated in a way that the corresponding cluster specifies a fuzzy set in the antecedent associated with a Gaussian membership function, $\mu(w) = \mathrm{e}^{-\frac{(w-c)^2}{2\sigma^2}}$, where the Gaussian center c is set as the center of the cluster, and the parameter σ is equal to the radius of the cluster. We choose Gaussian membership function for specifying fuzzy sets in order to provide a smooth output surface. In the consequent of a fuzzy rule, the output $r(t)$ is a linear function of $w(t)$, where the matrix A_i and vector b_i are fitting parameters estimated using the least-squares method.

The above modeling is performed periodically as queries are executed on the database VM, and it is capable of dynamically adapting to transitions in the VM's resource usage behaviors. Such a transition can be triggered by not only the change of the workload's intensity but also the change of its composition of queries with different resource needs. To adapt to such dynamic changes, the Adaptive Learner updates the VM's resource usage model at the end of every control period based on the latest data collected by the Sensors. So when a transition occurs, new data points that reflect the workload's current characteristics and the VM's current resource usages are used for modeling. As those data points become part of the online training data, the clustering result will be updated with a possibly different number clusters with different centers, so that a new set of fuzzy rules can then be created to represent the VM's current behavior. In this way, both the system structure and parameters are learned and adapted in real time from online data streams. The system model is gradually evolved instead of using 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 (typically under a second) for fine-grained resource control interval.

With the fuzzy model created from the Adaptive Leaner, the *Resource Predictor* performs fuzzy inference to generate an estimate of the resource need r given the workload input w. Based on the aforementioned clustering-based Sugeno-type fuzzy model, a Gaussian membership function is used in the antecedent of each rule to fuzzify the input w to its membership of the cluster in every rule. The membership value computed is then used as the weight for implication. In defuzzification, the consequent output of each rule is generated by the linear equation specified by associated parameters. The final output derived by aggregating all the weighted fuzzy outputs becomes the amount of resources estimated by the Predictor. This estimation is then sent to the Resource Allocator to guide the VM's resource allocation.

C. Resource Allocator

In a virtualized system, a VM serves as a resource container to the hosted database, where different types of resources can be dynamically allocated to this container for serving its workload. This is in contrast to traditional, non-virtualized database hosting, where a database's resource availability is statically defined by its physical machine's configuration. The Resource Allocator periodically (e.g., every 10 seconds) adjusts the multi-type resource allocation to database VMs based on the Resource Predictor's estimate. The Resource Allocator also needs to deal with situations where the resource prediction is inaccurate and causes the database's performance to diverge from the QoS target. This happens when the database's workload is first started or when its resource usage behavior changes so the Adaptive Learner cannot properly model the VM's current behavior.

In our approach, a backup resource allocation policy is employed to quickly recover from performance loss resulted from QoS violations when the VM's resource need is underestimated due to inaccurate workload modeling. This backup policy is invoked based on the recent information on the application's performance measurement $p(t)$, for instance, after the QoS target is missed for several (e.g., two) consecutive periods of time. This backup policy increases the current resource allocation by a fixed percentage (e.g., 100%) in order to satisfy the VM's unknown resource need

which is beyond its previous resource allocation level. (The choice of how soon to invoke the backup policy when a QoS violation happens is studied in Section V.D.) This fixed increment of resource allocation is accumulated until the QoS comes back to the target value, and afterwards the resource allocation is sustained at that level until the target is met for several (e.g., two) consecutive periods of time. Because the VM resource usage can be controlled at a fine granularity (in the matter of seconds), this mechanism allows the performance loss to be quickly recovered. Meanwhile, it also allows qualified data points to become quickly available so that the model can be timely updated to correctly reflect the VM's current resource needs.

However, the backup policy is only a supplemental method to our proposed fuzzy-modeling-based resource allocation. Although in the form of a traditional event-condition-action rule, it cannot substitute for the fuzzy model. The event-condition-action rules have to be predefined based on experts' knowledge, while the fuzzy model is automatically learned from the controlled system. Further, the event-condition-action rules are often statically defined, while the fuzzy model can be updated online to adapt to the changes in the system. Hence, the backup policy is only triggered when the model is inaccurate and unable to get qualified data to update itself. With the assumption that a workload's stable phases are much longer than its transition phases, the fuzzy model should be able to correctly predict the resource needs for most of the time therefore the backup policy would only be used infrequently.

V. EVALUATION

A. *Setup*

This section evaluates our approach using representative database workloads hosted on a typical VM environment. The testbed is a quad-core Intel Q6600 2.4GHz physical machine with 4GB RAM and 142GB SATA disk. 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. The evaluated databases are hosted on DomUs, while our resource management system is hosted on Dom0. In all the experiments, the management system monitors and controls the database VM's usage of *both* CPU cycles and disk I/O bandwidth every *10* seconds. In the VM Sensor, resource monitoring is done using xentop and iostat, where the I/O bandwidth usage is considered as the sum of reads and writes per period of time. In the Application Sensor, a database proxy deployed on Dom0 is used to measure the performance of the database VM. The Resource Allocator uses Xen's sEDF CPU scheduler to assign CPU allocations and Linux's dm-ioband I/O controller to set the cap for disk I/O bandwidth [17]. The sEDF scheduler uses 100ms period in the work-conserving mode. Another DomU VM running a CPU-intensive program is pinned on the same core assigned to the database VM to consume the surplus CPU cycles. Other VMs involved in our experiments are served as clients running outside of our testbed.

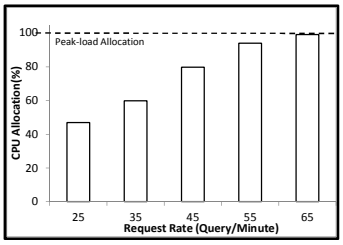

Figure 3(a). CPU allocation for the CPU-intensive TPC-H workload

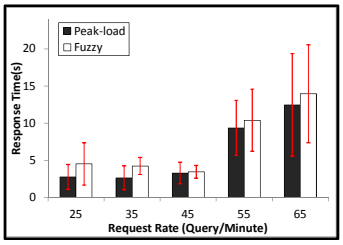

Figure 3(b). Response time for the CPU-intensive TPC-H workload

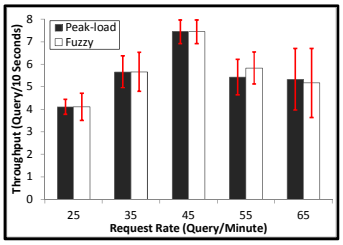

Figure 3(c). Throughput for the CPU-intensive TPC-H workload

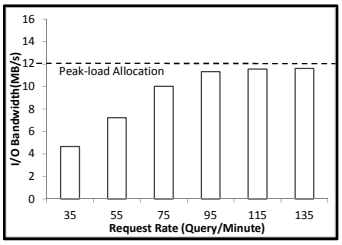

Figure 4(a). I/O bandwidth allocation for the I/O-intensive TPC-H workload

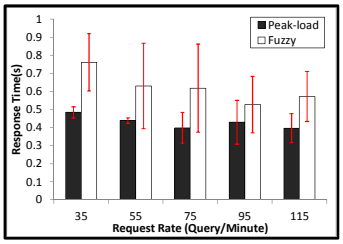

Figure 4(b). Response time for the I/O-intensive TPC-H workload

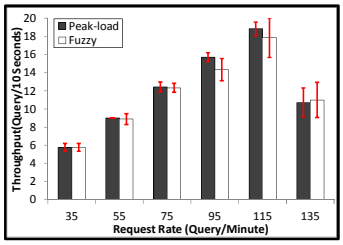

Figure 4(c). Throughput for the I/O-intensive TPC-H workload

Two typical database benchmarks, TPC-H [5] and RUBiS [6], are used in our experiments. The performance metrics considered in TPC-H include both average query throughput and average query response time measured every 10s. But in RUBiS only response time is considered because it is strongly correlated with throughput for this benchmark. Two different resource allocation schemes are compared: 1) The *peak-load-based resource allocation*, where the database VM is statically allocated sufficient resources based on its peak-load demand; 2) The *fuzzy-modeling-based resource allocation*, where the VM's resources are dynamically allocated based on our proposed approach. By comparing the VM's resource usage and the benchmark's performance between these two cases, we evaluate whether our proposed approach can achieve the same level of QoS while saving resources compared to peak-load based static resource allocation.

B. *TPC-H Experiments*

TPC-H provides 22 representative queries of business decision support systems, which involve the processing of large volumes of data with a high degree of complexity. Based on these queries, we construct synthetic workloads with varying demands of different types of resources. With peak-load based allocation, 100% CPU and 12MB/s or 10 MB/s I/O are allocated to the database VM statically. With fuzzy-modeling-based allocation, there are two phases involved. In the training phase, the fuzzy model is learned

without resource restrictions, while in the testing phase the model is applied to predict the resource demand and control the resource allocation. The evaluation of more realistic workloads with online training is discussed in Section V.C. The database used here is PostgresSQL 8.1.3 with 2 GB of data, hosted on a VM with one CPU and 1GB RAM.

1) Workload characterization for TPC-H

To characterize a TPC-H workload, its standard queries are first clustered based on the cost estimation extracted from PostgreSQL. A query cost vector <*CPU_cost*, *IO_cost*> is extracted from PostgresSQL's execution plan to represent the database estimated CPU and I/O costs for each query. Subtractive clustering is then used to group all the 22 queries based on their cost vectors, where a small radius of 0.1 is used in the clustering to derive tight clusters.

The result identifies four clusters. *Cluster I* containing single query Q1 and *Cluster II* containing single query Q18 represent highly and moderately CPU-intensive query, respectively. *Cluster III* including Q4, Q6, Q15 and Q12 represents highly I/O-intensive queries. *Cluster IV* including most of the remaining queries represents simple queries which are neither CPU nor I/O intensive. This result is experimentally verified by the actual resource usages when running the queries separately on the database VM. The only exception is Q22 which is identified as another single-query cluster and estimated by the database's cost model as both CPU and I/O intensive. However, its actual usage of CPU and I/O is very low, similarly to the queries in Cluster III, which confirms our discussion in Section IV.A that the database's query cost estimation cannot be used directly to infer the VM's resource need.

2) CPU-intensive Workload

The first experiment evaluates our approach for a CPU-intensive workload consisting of the two queries, Q1 and Q18, from Cluster I and II. While keeping the ratio of these two clusters constant (3:2), the workload's total request rate is varied between 25 to 65 requests per minute. A set of evenly distributed request rate values (225 data points) within this range are used to train the model which produces a 3-rule base. The workload is then run with a different set of request rate values (150 data points) to test the model, for each value, the workload is kept running for 300s. In the fuzzy-modeling-based approach, the resource allocation is done periodically every 10 seconds.

The CPU allocation and workload performance from using the fuzzy-modeling-based resource allocation and the peak-load-based resource allocation are compared in Figure 3. Note that both the workload performance and resource allocation shown from fuzzy-modeling-based approach are average values calculated from the measurements for each specific request rate. The performance obtained in the fuzzy-modeling-based allocation is always at the same level as the peak-load-based allocation even when the system becomes saturated after the request rate exceeds 55 query/minute. This demonstrates that our proposed fuzzy model is able to capture complex system behaviors over large region of the operating space. The throughput is within 96.4% to 100% of the peak-load-based allocation, while the average response

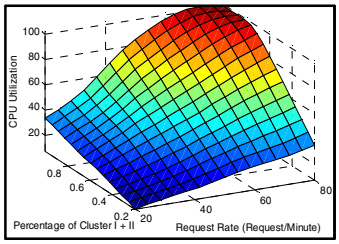

Figure 5(a). CPU model for the CPU/IO-intensive TPC-H workload

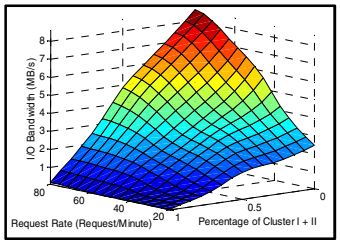

Figure 5(b). I/O model for the CPU/IO-intensive TPC-H workload

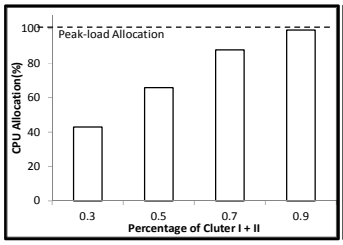

Figure 6(a) CPU allocation for the CPU/IO-intensive TPC-H workload

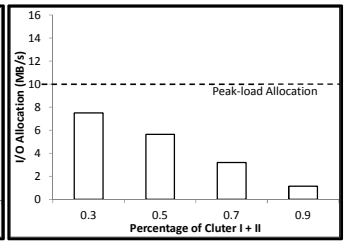

Figure 6(b) I/O allocation for the CPU/IO-intensive TPC-H workload

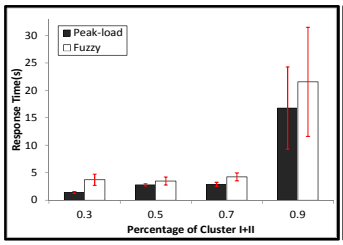

Figure 6(c) Response time for the CPU/IO-intensive TPC-H workload

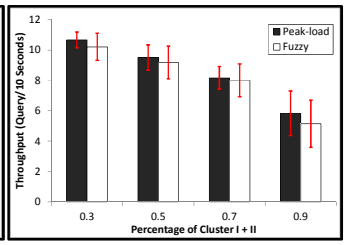

Figure 6(d) Throughput for the CPU/IO-intensive TPC-H workload

times only increase by at most two seconds. (The throughput is expressed in terms of number of completed queries every 10s, because these queries are complex and time-consuming.) At the same time, substantial amount of CPU allocation is saved when the workload is below the peak load. Note that, because of the difference in CPU intensity between Cluster I and II queries, the VM's CPU need changes as the ratio of these two clusters varies. Our approach can also properly model this behavior and accurately predict the VM's CPU need by taking this ratio as another input to the modeling. These results are omitted due to the limited space.

3) I/O-intensive Workload

In the second experiment, we consider an I/O-intensive workload using queries, Q6, Q15, Q12 and Q4, from Cluster III, which access a 200MB database table. We intentionally modified the original queries to only touch on a small region of the table so that we can vary the total request rate in a larger range. Further, the contiguous queries in the workload are set to access different regions so that the workload is always I/O intensive. Note that the purpose of this setup is only to make the experiment more interesting and it is only used in this experiment. The workload is created with a sequence of queries randomly picked from Cluster III. The total request rate of the workload varies from 20 to 140 requests per minute, where the training set (250 points) and the test set (200 points) are created similarly to the previous experiment. The resulting fuzzy model contains 4 rules.

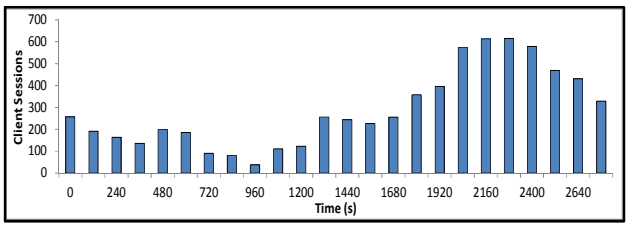

Figure 7(a). Trace for RUBiS with changing intensity

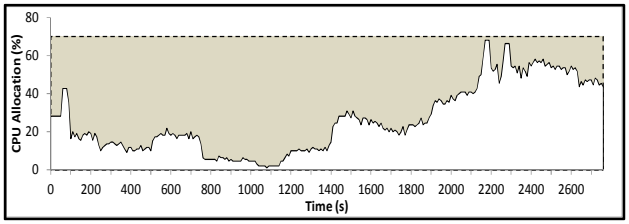

Figure 7(b). CPU allocation for changing intensity workload

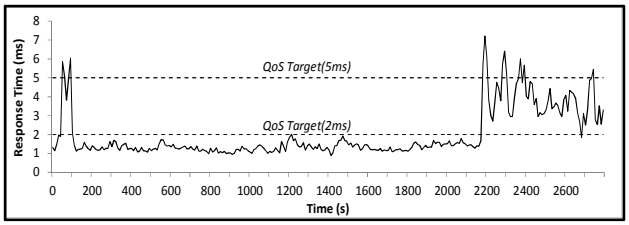

Figure 7(c). Performance for changing intensity workload

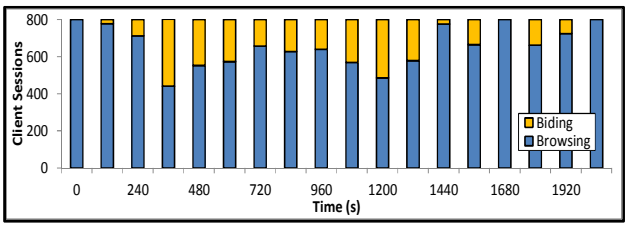

Figure 8(a). Trace for RUBiS with changing composition

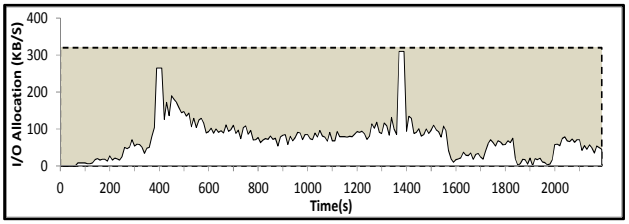

Figure 8(b). I/O allocation for changing composition workload

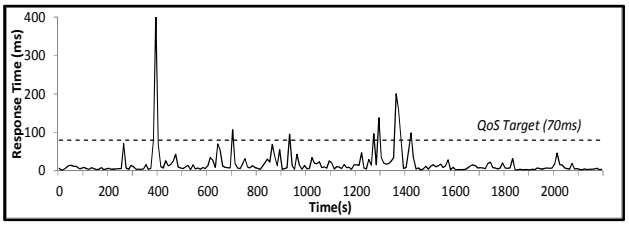

Figure 8(c). Performance for changing composition workload

Figure 4 compares the I/O bandwidth allocation, query response time, and query throughput between using fuzzy-modeling-based and peak-load-based resource allocation. (The response times for the request rate of 135, not shown in the figure due to the large magnitude, are 50.6s and 52.9s for peak-load-based and fuzzy-modeling-based allocations respectively. The CPU allocations are also omitted because this experiment is not CPU-intensive.) The results also demonstrate that our approach can accurately model the database VM's I/O bandwidth need for such an I/O intensive workload. The throughput is within 89.5% to 100% of the as the peak-load-based allocation, but up to 30% increase in response time is observed. We believe that this overhead is due to the non-work-conserving nature of the dm-ioband I/O bandwidth controller, which increases the queuing delay of the queries, affecting only the query response time but not the throughput. We will investigate how to improve dm-ioband for query response time in our future work. Nonetheless, substantial amount of I/O bandwidth is still saved using the fuzzy-modeling-based approach when the workload is below the peak load.

4) CPU/IO-intensive Workload

In the third experiment, we consider a workload that is both CPU and I/O intensive, by mixing queries from Cluster I (Q1), Cluster II (Q18), and Cluster III (Q6 and Q15). For simplicity, the ratio of the queries from Cluster I and II is fixed to 1:1 in the workload, but the total ratio of Cluster I+II over the entire workload composition is varied from 0.3 to 0.9. In addition, the total request rate of the workload also varies from 20 to 80 requests per minute. Different sets of data points are evenly taken from these data ranges for training (450 data points) and testing (150 data points).

This experiment is designed to evaluate our approach's ability to model a both CPU- and I/O-intensive workload with both changing intensity and changing composition. The model's input, the workload is characterized by both the total request rate and the ratio of Cluster I+II and Cluster III queries. The resulting model is illustrated by two 3-D sub-models each consisting of 12 fuzzy rules. The results show that our approach can properly capture such complex behaviors of the database VM. Figure 6 shows the resource allocation and workload performance when the request rate is fixed at 75 requests per minute but the Cluster I+II/Cluster III ratio varies. Compared to using peak-load-based resource allocation, the performance degradation from using fuzzy-modeling-based allocation is less than 5s in average response time and less than 10% in throughput, while saving both CPU and I/O bandwidth allocations. (The results from other request rates are similar and omitted here.) These results show that the VM's fuzzy model can accurately predict both its CPU and I/O need and the resource management system can effectively control them simultaneously, delivering good QoS to such a both CPU- and I/O-intensive workload.

C. RUBiS Experiments

RUBiS models an online auction site that supports the core functionalities such as browsing, selling, and bidding [6]. A typical two-tier setup is used to set up RUBiS, where the Web tier and database tier are deployed on separated VMs. The Web-tier VM hosts Apache Tomcat 4.1.40 with RUBiS and its clients while the database-tier VM hosts MySQL 5.0 with 1.1 GB of data. Both VMs are configured with one CPU and 1GB RAM. Since these experiments are performed *completely online*, only the first 10 data points collected are used to initialize the model. Afterwards the

model is used to allocate resources right away and in the meantime it is updated with new observed data every 10s. By interposing a MySQL proxy before the database tier, our system characterizes its query workload online in terms of intensity and composition. The composition can be captured by the ratio of two types of queries, the SELECT queries, which are read-only, and the INSERT and UPDATE queries, which are writes to the database.

1) Simulation of Real-world Workload

Compared to the synthetic workloads used in the above TPC-H experiments, here we constructed two more realistic workloads, one with changing intensity and the other with changing composition, based on real traces from the 1998 World Cup site [18]. This method is similar to those used by the related work for creating realistic workloads [19][20] .

The first workload with changing intensity is a browsing-only mix (Fig 7(a)) derived from a typical one-day hourly trace from the World Cup site. We first vertically scale the range of request rate to what our RUBiS setup can handle, i.e., mapping [50000, 100000] request/hour in the World Cup trace to [0, 1000] request/second in the RUBiS workload. Second, we horizontally scale the duration of workload from 24 hours to 2880 seconds, to speed up the replay of the trace. Since the workload intensity in RUBiS is controlled by the number of concurrent client sessions to the web tier, another mapping is created from the desired request rates to the number of client sessions.

The second workload is constructed in a similar way but we place emphasis on the variation in workload composition while keeping its intensity constant (the number of concurrent client sessions to the web tier is fixed at 800). Another one-day hourly trace with a stable request rate is chosen to derive this workload. We identify the read and write requests in the World Cup trace based on the "Get" and "Post" method, respectively, used in each request. The ratio of the read and write requests in this trace is then mapped to the ratio of the browsing and bidding requests in the RUBiS workload (Figure 8(a)), which corresponds to the SELECT to INSERT/UPDATE ratio to its database workload.

The desired QoS target for these workloads is defined according to the performance of the database VM under the peak-load-based resource allocation which statically assigns 70% CPU and 320KB/s disk I/O bandwidth. For the changing intensity workload, the QoS target is 2ms when the web tier is not saturated and 5ms otherwise. For the changing composition workload, the QoS target is set to 70ms.

2) Results

Figures 7(b) and (c) show the CPU allocation and query performance of the database VM for the changing intensity workload (the I/O allocation result is omitted because this workload is not I/O intensive). As soon as the model is initialized through the first ten data points, it is able to accurately predict the VM's resource need throughout most of the experiment even when the burst occurs at time 480s, 1450s, and 1930s without using the backup resource allocation policy. At time 2100s, the system is under its peak load, current model underestimates the CPU need and the backup resource allocation policy is triggered to ensure the availability of qualified data for model adaption. After two control periods (20s), the model is adapted to the new system behavior and able to correctly predict the new resource need while the backup policy is stopped. The shaded area in Figure 7(b) illustrates the amount of resource saved (62.6%) in fuzzy-modeling-based resource allocation. Figure 7(c) shows that the average query response time can meet the desired QoS target most of the time (Only 11 QoS violation periods occurred throughout the entire experiment).

Figures 8(b) and (c) show the I/O allocation and query performance of the database VM when running the changing composition workload (the CPU allocation results is omitted due to limited space). It is evident that the fuzzy-modeling-based resource allocation can quickly react to the changes in workload composition and deliver the desired QoS most of the time. The spikes occurred at 360s and 1400s are caused by rapid shifts in the ratio of the workload's biding and browsing requests. The backup policy was invoked only at these two times to quickly adapt the model and meet the QoS target again. We believe that by improving the dm-ioband I/O bandwidth controller with work-conserving scheduling can further reduce the spikes in response time during such abrupt transitions.

D. Modeling Sensitivity and Overhead

A key parameter used in the backup policy is the threshold for deciding when to invoke and stop the backup policy. In the above RUBiS experiments, this threshold is set to two, which means that the backup policy is triggered when the QoS target is missed for two consecutive control periods and then canceled after the QoS target is met again for two consecutive periods. When the backup policy is effective, it quickly increases the VM's resource allocation by doubling it every time the required QoS is violated. When it is stopped, the predicted resource need from the updated model is again used to decide the VM's resource allocation.

In the last experiment, we study the sensitivity to this threshold of our proposed approach study using a workload with changing composition created by switching between four mixes, each producing a constant percentage of write queries, 0%, 4%, 8%, and 20% respectively, to the database tier. Each mix lasts 300 seconds and then transits immediately to the next mix. The number of concurrent client sessions is kept at 200. We run this workload on RUBiS and use the fuzzy-modeling-based resource allocation with different threshold values. The result shows that the same level of average throughput (21 query/s) can be achieved when the threshold value varies from 1, 2, to 4, but the total number of uses of the backup policy drops from 12, 2, to 1, respectively (the figures are omitted due to limited space). It confirms that if the threshold is set lower, the backup policy is invoked more often while it is set higher, longer QoS violations are experienced during the transitions. The result verifies that the threshold value of two is a good choice but in general this tradeoff can be determined by considering both the QoS requirement and resource cost.

We also measured the overhead of our approach for modeling and controlling the database VM's resource usage in the RUBiS experiment. The resource consumed by the

management system is small, which is less than 20MB of memory and 1% of CPU when measured every second. The time required for modeling is also small, although it slightly increases as the size of the training data grows. With 1000 data points, it takes about 0.4s. In practice, when a sliding window is used to ensure the freshness of training data, this overhead will remain negligible. The time required for fuzzy inference is even smaller and independent of the dataset size.

VI. RELATED WORK

Various solutions have been studied in the literature to address the problem of automatically deciding a VM's resource needs based on its hosted application's demand and QoS requirement. Although there is also a significant body of related work on performance modeling of non-virtualized systems, due to the limited space, the discussions here focus only on virtualized resource management.

The first category of solutions employs queuing theory to construct analytical performance models for virtualized applications. For example, Doyle *et al*. derive analytical models from basic queuing theory to predict response times of Internet services under different load and resource allocation [21]; Bennani *et al*. consider using multiclass queuing networks to predict the response time and throughput for online and batch workloads on VM based application environments [22]. However, solutions of this type are restricted by their often simplified assumptions on a virtualized system's internal structure, and are difficult to capture the system's complex resource usage behavior.

The second category of solutions applies control theory to adjust VM resource allocation and achieve the desired application performance. Such solutions often assume a linear performance model for the virtualized application, although it can be updated adaptively as the system moves from one operating point to another. For example, first-order autoregressive models are used to manage CPU allocation for Web servers [23][24]. Linear multi-input-multi-output (MIMO) models are used to manage the multi-type resources for multi-tier applications [25][26]. Similar MIMO model is also used to allocate CPU resource for compensating the interference between concurrent VMs [27]. This work also points out that a nonlinear model can model such interference much more accurately. In addition, our ongoing work, a fuzzy-model-based predictive controller, which embeds fuzzy-modeling into a predictive control system, shows promising results for both effectively capturing complex system behaviors and quickly adapting to changes in the system [28].

The third category of solutions considers machine learning techniques to automatically learn the complex model for a virtualized system based on data observed from the system. For example, simple regression method is used to predict the performance impact of VM memory allocation [29]; Regression method is also employed to map a resource usage profile obtained on a physical system to that on a virtualized system [30]; Reinforcement learning technique is used to automatically tune VM resource configuration and to achieve good performance [31]; Signal processing technique is employed to predict repeating resource usage patterns for applications and hosts in a cloud [19] ; Finally, Kund *et al.* use artificial neural networks to build performance models for a variety of applications which also prove to be highly nonlinear and complex in many cases [32] .

Our proposed fuzzy-modeling-based solution falls into the third category. It is advantageous in that it does not require any a priori knowledge of the VM's system model, and it can efficiently model a nonlinear system with dynamically changing resource usage behaviors. Compared to the aforementioned adaptive linear model, such a continuous nonlinear model can accurately capture the system's entire behavior and allow optimized resource allocation over the entire operating space.

In the related research of virtualized database hosting systems, Farooq *et al.* experimentally evaluated VM-based databases and showed that the overhead is very small compared to natively hosted databases [33], which also confirms the feasibility of such approaches. Soror *et al.* address the problem of automatic resource configuration for database VMs by calibrating database's internal query cost model [34]. However, this work treats a workload as a static entity with a fixed set of queries, so the performance considered is the overall runtime and the VM configuration is done statically for the entire workload. The offline calibration process considers VM's use of CPU, memory, and I/Os as independent from each other, which may not hold due to the complexity of resource virtualization. When the database's cost model is inaccurate, this work employs online refinement by assuming a linear resource usage model. Therefore, it is unclear how this approach would apply to and how well it would perform for a workload with complex resource usage and dynamically changing behavior, which can be well handled by our proposed approach.

There is also related autonomous database work [35][36][37], which focus on a database's internal tuning and query optimization, and is orthogonal and complementary to the problem addressed by this paper, which focuses on the resource allocation to an entire database VM. Previous work on workload characterization [38][39] also considers it as the key to understanding the resource intensity of a database workload. In these studies, a workload is often described with time-invariant structure and parameters, which is far from the real-world situations. This paper improves the static workload characterization method adopted by the database community by allowing online and adaptive characterization.

VII. CONCLUSIONS AND FUTURE WORK

Virtualization can greatly facilitate the deployment of database systems and substantially improve their resource utilization. To fulfill this potential, resource management is the key, which should be able to automatically allocate resources to database VMs based on their QoS targets. This paper presents an autonomic resource management system that can achieve this goal through a fuzzy modeling based approach, which models a database VM's resource usage behaviors based on observed data and predicts its resource needs for its current workload demand. This process is done periodically (in terms of seconds) online to guide dynamic resource allocation and adapt to changes in the system.

Experiments based on typical database benchmarks show that our system can accurately estimate a database VM's resource needs for dynamic and complex query workloads, meet the desired query QoS, and save substantial resources compared to peak-load based static allocation.

However our modeling-based approach relies on the predefined backup policy to deal with situations where the VM's resource demand is misestimated due to dynamic changes in the VM's resource usage behaviors. To eliminate the need for such a supplementary strategy, we are studying a new approach which combines fuzzy modeling with predictive resource control [28]. This approach allows a VM's resource allocation to be directly adjusted based on the application's performance feedback and the QoS target. Our preliminary results show that it can automatically track the QoS target and quickly adapt to changes in the system. In addition, it also employs multi-input-multi-output fuzzy modeling which can simultaneously model the resource usages of multiple VMs and at the same time capture the interference between them.

Although this paper focuses on virtualized databases, we believe that our proposed fuzzy-modeling-based approach is generally applicable to the virtual resource management for other types of applications. The only application-specific part of this approach is on its use of an application's internal knowledge to help the workload characterization. Compared to the traditional approach which treats a VM as a black-box, such a gray-box resource management approach can be more accurate for modeling virtualized applications that have dynamic and complex resource usage behaviors.

ACKNOWLEDGMENT

This research is sponsored by National Science Foundation under grant CCF-0938045, OCI-0910812, IIP-0932023, CNS-0855123, and IIP-0758596, Department of Homeland Security under grant 2010-ST-062-000039, and a USF Seed Grant from Florida Energy System Consortium. 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.

REFERENCES

[1] VMware, URL: http://www.vmware.com.
[2] P. Barham, B. Dragovic, K.Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, "Xen and the Art of Virtualization", SOSP, 2003
[3] Amazon Elastic Compute Cloud, URL: http:// aws.amazon.com/ec2/.
[4] Windows Azure, URL: http://www.microsoft.com/windowsazure/.
[5] TPC-H Benchmark Specification, URL: http://www. tcp. org.
[6] C. Amza, A. Chanda, A. Cox, S. Elnikety, R. Gil, K. Rajamani and W. Zwaenepoel, "Specification and Implementation of Dynamic Web Site Benchmarks", WWC-5, 2002.
[7] A. Chen, P. Goes, A. Gupta and J. Marsden, "Heuristics for Selecting Robust Database Structures with Dynamic Query Patterns", EJOR, 2006.
[8] M. Wang, T. Madhyastha, N. Chan, S. Papadimitriou and C. Faloutsos, "Data Mining Meets Performance Evaluation: Fast Algorithms for Modeling Bursty Traffic", ICDE, 2002.
[9] L. Zadeh, "Fuzzy Sets", Information and Control, 1965.
[10] J. Xu, M. Zhao and J. Fortes, "Autonomic Resource Management in Virtualized Data Centers Using Fuzzy-logic-based Control", Cluster Computing, 2008.
[11] T. Takagi and M. Sugeno, "Fuzzy Identification of Systems and its Application to Modeling and Control" TSMC, 1985.
[12] S. Chiu, "Fuzzy Model Identification Based on Cluster Estimation", Journal of Intelligent and Fuzzy Systems, Vol. 2, No. 3, 1994.
[13] HP-UX Workload Manager, http://docs.hp.com/en/5990-8153/ch05s12.html.
[14] J. Rolia, L. Cherkasova and C. McCarthy, "Configuring Workload Manager Control Parameters for Resource Pools", NOMS, 2006.
[15] Y. Diao, J. Hellerstein and S. Parekh, "Optimizing Quality of Service Using Fuzzy Control", DSOM, 2002.
[16] S. Chaudhuri, "Relational Query Optimization – Data Management Meets Statistical Estimation", Communications of ACM, 2009.
[17] dm-ioband, URL: http://sourceforge.net/apps/trac/ioband.
[18] M. Arlitt and T. Jin, "Workload Characterization of the 1998 World Cup Web Site," in HP Technical Report, 1999.
[19] Z. Gong and X. Gu, "PAC: Pattern-driven Application Consolidation for Efficient Cloud Computing", MASCOTS, 2010.
[20] G. Jung, M. Hiltunen, K. Joshi, R. Schlichting and C. Pu, "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures", ICDCS, 2010.
[21] R. Doyle, J. Chase, O. Asad, W. Jin and A. Vahdat, "Model-Based Resource Provisioning in a Web Service Utility", USENIX, 2003.
[22] M. Bennani and D. Menasce, "Resource Allocation for Autonomic Data Centers using Analytic Performance Models", ICAC, 2005.
[23] X. Liu, X. Zhu, S. Singhal and M. Arlitt, "Adaptive Entitlement Control of Resource Containers on Shared Servers", IM, 2005.
[24] Z. Wang, X. Zhu and S. Singhal, "Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions", DSOM, 2005.
[25] P. Padala, K. Hou, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal and A. Merchant, "Automated Control of Multiple Virtualized Resources", SIGOPS/EuroSys, 2009.
[26] X. Liu, X. Zhu, P. Padala, Z. Wang and S. Singhal, "Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform", CDC, 2007.
[27] R.Nathuji and A. Kansal, "Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds", Eurosys, 2010.
[28] L.Wang, J. Xu, M. Zhao and J. Fortes, "Adaptive Virtual Resource Management with Fuzzy Model Predictive Control" FeBID, 2011.
[29] J. Wildstrom, P. Stone and E. Witchel, "CARVE: A Cognitive Agent for Resource Value Estimation", ICAC, 2008.
[30] T. Wood, L. Cherkasova, K. Ozonat and P. Shenoy, "Profiling and Modeling Resource Usage of Virtualized Applications", Middleware, 2008.
[31] J. Rao, X. Bu, C. Xu, L. Wang and G. Yin, "VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration", ICAC, 2009.
[32] S. Kundu, R. Rangaswami, K. Dutta and M. Zhao, "Application Performance Modeling in a Virtualized Environment," HPCA, 2010.
[33] U. Minhas, J. Yadav, A. Aboulnaga and K. Salem, "Database Systems on Virtual Machines: How Much do You Lose?", SMDB, 2008.
[34] A. Soror, U. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis and S. Kamath, "Automatic Virtual Machine Configuration for Database Workloads", SIGMOD, 2008
[35] G. Weikum, A. Moenkeberg, C. Hasse and P. Zabback, "Self-tuning Database Technology and Information Services: From Wishful Thinking to Viable Engineering", VLDB, 2002.
[36] S. Chaudhuri and G. Weikum, "Foundations of Automated Database Tuning", ICDE, 2006.
[37] B. Schroeder, M. Harchol-Balter, A. Iyengar and E. Nahum, "Achieving Class-based QoS for Transactional Workloads", ICDE, 2006.
[38] P. Martin, S. Elnaffar and T. Wasserman, "Workload Models for Autonomic Database Management Systems", ICAS, 2006.
[39] T. Wasserman, P. Martin and D. Skillicorn, "Developing a Characterization of Business Intelligence Workloads for Sizing New Database Systems", DOLAP, 2004.