Florida International University
FIU School of Computing and Information Sciences

Autonomic Resource Management for Virtualized Database Hosting Systems

Technical Report 2009-07-01

July, 2009

School of Computing and Information Sciences (SCIS)

Florida International University (FIU)

Miami, Florida, USA 33199-0001

Lixi Wang*, Jing Xu†, Ming Zhao*, Yicheng Tu‡, Jose Fortes†

* Florida International University, {lwang007, mzhao}@fiu.edu

† University of Florida, {jxu, fortes}@ufl.edu

‡ University of South Florida, ytu@cse.usf.edu

Autonomic Resource Management for Virtualized Database Hosting Systems

Lixi Wang[*], Jing Xu[†], Ming Zhao[*], Yicheng Tu[‡], Jose 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 allocation of physical resources on demand to a database's VM according to QoS (Quality of Service) requirements. This is a challenging problem because of the highly dynamic and complex nature of database systems and their workloads. An autonomic resource management approach is proposed to address this problem based on adaptive fuzzy modeling and prediction techniques. The approach can effectively capture the relationship between a dynamically changing database workload, which is both CPU and I/O intensive, and its VM's consumption of resources, including both CPU cycles and disk bandwidth. It can be used to predict the resource needs of a database VM online and to guide the on-demand resource allocation according to the workload demand and desired QoS. A prototype of the proposed resource management system is evaluated using typical database workloads based on TPC-H and RUBiS. The results demonstrate that the proposed approach can efficiently allocate resources for a database VM that is serving CPU and I/O intensive queries while meeting the QoS targets.**

I. INTRODUCTION

A system-level virtual machine (VM) (e.g., VMware [1], Xen [2]) can be a powerful platform for deploying and hosting database systems. From the perspective of database users, VMs enable fine-tuned databases to be encapsulated along with their execution environments and conveniently deployed as appliances on different hosting systems. From the perspective of resource owners, VMs allow flexible resource allocation to meet changing database needs and efficient resource utilization by sharing resources between databases and other applications. However, although many important applications, such as Web and application servers, have been widely deployed on VMs, efficient hosting of databases on virtualized resources is still very challenging due to the highly complex and dynamic nature of database systems and their workloads. Typical databases have to serve dynamically changing workloads consisting of a wide variety of queries, whereas the query executions can consume different types and amounts of resources, including both CPU and I/O. These properties make it difficult to host databases on shared resources without compromising performance or wasting resources.

This paper aims to address the above challenges through an autonomic VM resource management system that can automatically control and optimize the allocations of different types of resources to database VMs based on their workload demands and QoS (Quality of Service) objectives. The fundamental goal of this proposed system is two-fold. First, it should be able to automatically learn a database VM's needs for multi-type resources to service a complex query workload so that resources can be efficiently allocated to the VM while satisfying the desired query QoS. Second, it should be able to automatically adapt to the dynamic changes of a database VM's resource usage and timely adjust the VM's resource allocations to maintain both the efficiency of resource usages and the QoS of queries.

To realize the above stated goals, this paper proposes a fuzzy-modeling based online learning and prediction approach to the autonomic resource management of virtualized database hosting systems. In this approach, fuzzy-logic based modeling is adopted to automatically learn the resource usage behaviors of database VMs based on observed query workload characteristics and VM resource consumptions. This modeling method does not require any *a priori* knowledge of the system's internal structure and it can efficiently describe complex and nonlinear system behaviors. Specifically, a database VM's resource model is constructed online and updated dynamically to learn the relationship between a query workload's changing characteristics and the VM's needs of multi-type resources, particularly CPU cycles and I/O bandwidth. This model is then applied also online to predict the database VM's multi-type resource needs for its current workload and to allocate resources efficiently to the VM and meanwhile meet the QoS target for the queries.

This resource management system is implemented for Xen-based VM environments and it is evaluated using a series of experiments based on typical database benchmarks (TPC-H [3], RUBiS [4]). The results demonstrate that the system can efficiently allocate resources for a database VM that is serving CPU and I/O intensive queries while still delivering the same level of performance as when all the resources are dedicated to the VM. The results also show that the system can adapt to dynamic transitions of the database VM's resource usage caused by changing workload intensity and composition, achieving both resource efficiency and query QoS in a timely manner.

In summary, this paper has made the following unique contributions: *1)* It proposes a novel autonomic resource management system for database VMs, which can efficiently allocate different types of resources according to the query workload demand and can timely adapt to changes in their resource usage behaviors; *2)* It develops an implementation for typical Xen-based VM systems, which can manage and optimize the use of both CPU cycles and I/O bandwidth for database VMs serving resource-intensive workloads; 3) The overall approach proposed in this paper is also generally applicable to the virtualized hosting of other kinds of challenging applications that have dynamic and complex resource usage behaviors.

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

II. Background And Motivation

A. System Virtual Machines

The emergence of VMs is driven by the fast maturation and wide availability of virtualization technologies, as well as the rapid growth of computing power on modern computer systems. The VMs considered in this paper are system-level VMs [1][2], which are based on the virtualization of an entire physical host's resources, including CPU, memory, and I/O devices, presenting virtual resources to the guest operating systems and applications. Such VMs are mainly implemented by the layer of software called Virtual Machine Monitor (VMM, a.k.a. hypervisor). Although our proposed techniques can also be applied to some other types of virtualization (e.g., OS-extension based VMs [5][6]), those are not the focus of this paper.

This paper considers the use of dedicated VMs to host different applications and allow them to transparently share the underlying resources. Because the multiplexing of applications to resources is provided at a lower level of the system, it has the following advantages compared to traditional OS-based resource sharing: *1)* VMs provide strong isolation for resource sharing, allowing applications on one VM to be protected from failures and security breaches occurred on another concurrently hosted VM; *2)* Virtualization supports flexible allocation of various types of resources to VMs, and VM migration further enables dynamic balancing of resource usages across physical hosts; *3)* VMs allow application-tailored customization of their execution environments, including OSes and libraries, and enable applications to be seamlessly deployed onto resources with heterogeneous configurations.

B. 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][9], where the workloads are intrinsically dynamic in terms of their bursty arrival patterns and ever-changing unit processing costs. Even under domains where traditional 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. Therefore, the limitations of the traditional database hosting approach are two-fold. First, peak-load based resource provision leads to overprovision and thus underutilization of resources for normal state workloads. This can cause considerable infrastructural and operational overhead. Second, as a steady-state workload demand exceeds its previously expected peak value, the database's performance may drop dramatically due to overload, unless it can be moved to a more powerful server through a lengthy relocation process.

Using VMs to host databases can effectively address the above limitations, 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. Virtualization can consolidate many dedicatedly provisioned physical servers into a small number of shared ones, where each of them can be carved into multiple virtual resource containers to provision resources to applications. This approach allows a database system to share the consolidated resources with other databases and applications, with strong isolation by hosting them on dedicated VMs. It also allows a database VM's resource allocation to elastically grow and shrink based on the workload's demand. In addition, database VMs can be dynamically migrated across physical machines for resource optimization.

Virtualization also offers a new paradigm for database deployments. Modern databases have become rather sophisticated software systems, where their installation, configuration, and tuning often require substantial domain knowledge and experience as well as considerable efforts from the database administrators (DBA). This presents a hurdle to the wide deployment and effective use of databases. VM-based database hosting allows carefully installed and finely tuned databases to be distributed as simply as copying the data that represent the database VMs. In this way, a DBA only needs to install, configure, and tune a database once in the environment provided by a VM. With that, the deployment of the database on a new host only entails transferring the VM data to the host, creating a new VM instance from the data, and starting the new database that is already deployed in the VM. In addition, this approach allows databases to be quickly replicated and distributed for performance and reliability improvements.

C. Autonomic VM Resource Management

VM-based application hosting allows dynamic resource allocations based on the demands from applications, thereby improving the overall resource utilization. However, a key challenge to the success of this approach is how to allocate resources to a VM to achieve both the application desired QoS and the system desired resource efficiency, and how to do so for all the VMs automatically and continuously. To address this challenge, autonomic computing techniques can be employed to realize self-managing of VM resource configurations according to the high-level application performance and resource utilization objectives [10]. A Monitor-Analyze-Plan-Execute (MAPE) control loop [11] can be deployed to monitor the VM's workload demand, analyze its resource needs, plan its resource configuration, and then execute it accordingly. This paper follows this approach to build an autonomic system for the resource management of virtualized database hosting systems.

An important task of an autonomic resource management system is to analyze a VM's resource usage behaviors and decide its proper resource configuration based on its hosted application's workload demand and QoS requirements. This task is particularly challenging for database systems because of their highly complex and dynamic multi-type resource usages. 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. This makes it difficult to determine the allocations of multi-type resources to a database VM without over-provisioning and yet satisfying its desired query performance.

In order to understand a VM's resource needs for its hosted application, several different types of approaches have been proposed and they are examined in details in Section V. In particular, machine learning techniques can be employed to learn the relationship between the workload demand and the VM's resource usages, which can be then used to guide the resource allocation to the VM. Machine-learning based approaches are advantageous than others in that a VM's resource usage model is automatically created from data observed from the system, without assuming any *a priori* knowledge about the system's structure.

Compared to other typical machine learning techniques, fuzzy-logic based modeling is particularly suited to efficiently model systems with complex behaviors [17]. Our previous work has successfully applied this approach to the CPU resource management of VMs hosting CPU-intensive Web servers [18]. However, the management of virtualized database systems raises new, important challenges: First, how to effectively manage a VM with multi-type resource demands, including not only CPU cycles but also I/O bandwidth, which is known to be difficult to model and control; Second, how to timely adapt to the dynamic changes in a VM's resource needs in terms of not only changing intensity but also shifting demand across different resource types. These new challenges are addressed by the system proposed in this paper in which databases serve as an excellent example of applications with dynamic, multi-type resource usage behaviors.

D. Fuzzy-logic based System Modeling

Fuzzy logic [17] is suited for dealing with uncertain problems in real world, which transforms imprecise linguistic statement into quantified logical input-output relations by using mathematical functions. A set in the fuzzy world, describes vague concepts such as hot weather, faster runner, etc., called *fuzzy set*, which no longer has a crisp, clearly defined boundary. Instead, elements can be contained to a fuzzy set with a partial degree of membership which is determined by a *membership function* that maps the input space to a membership value between 0 and 1. A membership function can be of any shape as long as its range is within [0, 1] and the commonly used ones include triangular, Gaussian, and sigmoid functions.

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. 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. To reduce the size of the dataset and the corresponding number of fuzzy rules, data clustering techniques are often employed to derive a concise representation of the system's behavior and to determine a minimum number of fuzzy rules needed in the model.

The process of formulating 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.

III. Approach

This paper proposes an autonomic resource management system to automatically control and optimize the multi-type resource utilizations for database VMs serving dynamic and complex workloads. The objective of this system is two-fold: First, without any *a priori* knowledge of the database system itself, the proposed management system should be able to determine the relationship between the database workload demands and the VM's needs of multi-type resources for meeting the desired QoS; Second, the system should be able to timely update its model and adjust the resource allocation when the workload is dynamically changing its intensity and shifting its demands across different resource types.

Figure 1 illustrates the high-level architecture of our proposed autonomic resource management system which consists of four key modules, Application and VM Sensors, Adaptive Learner, Resource Predictor, and Resource Allocator. As a workload executes on the database VM, the Application and VM Sensors monitor the characteristics of the workload *w(t)*, its performance *p(t)*, and the VM's resource usage *r(t)*. This information is fed to the Adaptive Learner to model the VM's resource needs and continuously updates the model. With this model and the monitored current workload *w(t)*, the Resource Predictor produces an estimation of the resource needs for time *t+1*. Based on this prediction, the Resource Allocator then adjusts the VM's resource configurations accordingly. Together, these modules form a continuous closed loop for the database VM's resource control and optimization.

A. Application and VM Sensors

As illustrated in Figure 2, the Sensors collect real-time information from both an application (current workload characteristics and QoS) and its VM (current resource usage), which are necessary to build the VM resource usage model and to predict its current resource needs.

1) Workload Characterization

Workload characterization is the process of describing the characteristics of a workload that are relevant to its

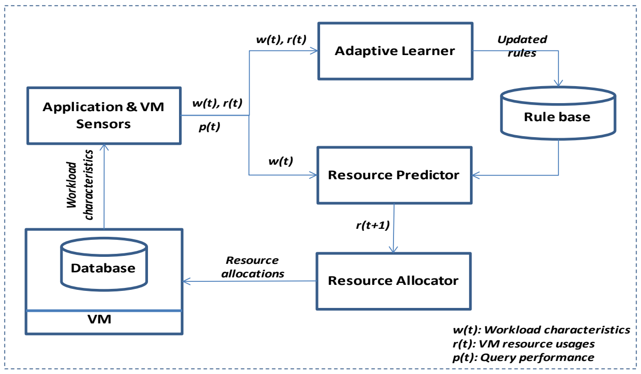

Figure 1. Architecture of the autonomic resource management system for VM-based databases

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 intensity and is strongly correlated with its resource demand. For example, a typical workload for a Web server can be characterized by its HTTP request rate, where the more HTTP requests received per period of time, the more CPU cycles the VM will consume during the period in order to process those requests [18]. However, the request composition of the workload may also have a significant impact on the resource demands and need to be considered for the modeling. A Web server's workload can contain a mix of static-content requests and dynamic-content requests, which have different needs on the use of CPU and I/O resources. Therefore, the ratio between these two types of requests can be taken as another characteristic to describe such a workload and contributes to the modeling of the Web server VM's resource usage.

The characterization of a database workload is even more challenging because of the complexity and diversity of individual queries and the dynamism of the workload. A real-world database's workload typically consists of large numbers of different kinds of queries, each with widely varying resource requirements, where the workload's query composition also changes dynamically over time [19]. This poses a great challenge to consider all the different resource-intensive queries for characterizing the behaviors of a database workload. To address this challenge, this paper employs the following two different solutions for effective database workload characterization.

The first solution is to group the queries in a workload into a small number of categories based on their behaviors in consuming resources, and then describe the workload as a vector of arrival rates of these different categories. For workloads that have a fixed set of queries (even though the size of this set may be very large), offline modeling can be performed to observe their resource usage behaviors and data clustering techniques can be used to group the queries into representative clusters. For example, Martin and Wasserman *et al.* [20] have considered the use of unsupervised data mining techniques to classify the TPC-H queries into four clusters which have distinct resource consumption patterns. However, for workloads with queries that cannot be learned or modeled *a priori*, they have to be characterized online. To this end, we will consider in our future work the use of query cost estimation from DBMS (Database Management System) to drive the online characterization of such a workload.

The second solution is to describe a database workload based on the knowledge of the application that generates the database queries. Particularly, in a typical multi-tiered Web system, queries received at the database tier are triggered by the user requests at the Web tier. The types of such requests can have distinct impact on the resource usages of their triggered queries. For example, in a multi-tier online auction site [4], the user interactions with the website mainly include several different types such as browsing, bidding, and selling, which generate different types of queries with distinct CPU and I/O usage behaviors (Section IV.C). Consequently, the composition of these different types of HTTP requests received at the Web tier, which is easily observable, can be then taken to infer the characteristics of the corresponding query workload at the database tier.

2) *Resource Usage Monitoring*

The VM Sensor is responsible of collecting the VM's real-time resource usage information, which is the other key component of the inputs to VM resource usage modeling. Although workload characteristics are collected from the application running inside of the VM, resource utilizations need to be monitored from outside of the VM. This is important because the application's virtual resource usages do not truthfully represent its VM's physical resource usages. Due to the need of virtualization, an application's use of one type of resources may incur the use of a different amount of this type as well as the use of other types of resources for the VM. For example, an application's I/Os to its VM's virtual disk can trigger a different amount of I/Os to the physical disk, depending on how the virtual disk is stored physically; The processing of these virtual I/Os also costs CPU cycles due to the need of I/O virtualization. Therefore, resource usage monitoring needs to be done at the VM level, instead of at the application level, in order to capture all the necessary resources for servicing its workload on a VM.

The VM Sensor monitors multiple types of resources including CPU, memory, and disk and network I/Os. This is key to the modeling of a database application because it has complex resource usage behaviors and can have intensive demands for multi-type resources. Database management and query optimization can be highly CPU intensive, whereas loading and storing records to and from storage can be highly disk I/O intensive. In addition, if the database storage is delegated to a remote server across the network, as in many commercial database setups, the database system can also be highly network I/O intensive. Therefore, the resource management system considers CPU cycles and I/O bandwidths as the resource costs for database queries and monitors both of their usages for the VMs.

Typical VM technologies all provide a comprehensive interface for the monitoring of VM resource usages. In the prototype built upon the Xen VM environment, the VM Sensor is implemented as a user-level daemon running on Xen's management and I/O VMs, which are privileged VMs dedicated for VM management and I/O processing for the entire physical host. Specifically, the resource monitoring daemon uses Xen's *xentop* utility to collect a VM's CPU utilization and the Linux *iostat* utility to collect the statistics of a VM's I/O bandwidth usages.

3) Performance Measurement

In addition to the information about application workload and VM resource usage, the management system also needs to monitor the application's current performance, which is important to decide whether the current resource usage can represent the VM's resource needs for the desired QoS, and whether the current resource allocation is sufficient to meet the desired QoS. The information of application performance is also collected by the Application Sensor. For a typical application, the commonly used performance metrics include throughput (number of completed requests per unit time) and average request response time (average service time of requests per unit time). Generally, workloads with large volumes of small requests are more interested in throughput, whereas those with small number of large requests are more concerned with the average response time.

These two metrics are also often used to measure a database workload's performance, both of which can be collected by the Application Sensor. Note that we consider a workload as a *continuous*, *dynamic* process. Therefore, the throughput and average response time reported by the Sensor are fine-grained, real-time measurements taken periodically from the queries, rather than the overall values measured only once for the entire workload. To implement the performance measurement, the Sensor can collect such information from typical DBMSs that are capable of monitoring and reporting query statistics in their logs. A more general way is to implement the Sensor as a proxy that interposes between the database client and server, so it can forward the queries to the database and meanwhile measure their performance. This is the approach taken by our prototype, where the Sensor runs as a database proxy on the same VM that the database is hosted on.

B. Adaptive Learner

The Adaptive Learner module is responsible for creating and updating the model that represents the relationship between a database workload and its VM's resource needs. Although DBMS also provides estimations of the resource costs for queries based on its internal query evaluation strategy, this mechanism is not used in our approach to predict the database VM's resource needs because of two important reasons. First, the accuracy of DBMS query cost estimation is known to be limited [21]. Although it works well for relative cost comparison for the purpose of selecting the optimal execution plan for a query, it is not sufficient to provide an accurate estimation of the query's actual resource needs and to guide the resource allocation. Second, the resource estimation given by DBMS, which runs inside of a VM and is completely unaware of the virtualization, cannot truthfully capture the resource needs of the VM, which can be significantly influenced by the virtualization process and the resource competition from co-hosted VMs.

Therefore, in our proposed resource management system, the Adaptive Learner employs a fuzzy modeling based approach to automatically discover the relationship between a database workload and its VM's resource needs. Fuzzy modeling is the process of constructing fuzzy rules based on the input and output data pairs, $<w(t), r(t)>$, which are periodically 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 certain characteristic of the workload and for $r(t)$ each dimension maps to one type of consumed resources. Note that the Learner needs to filter out the unqualified data points collected when the workload performance $p(t)$ cannot satisfy its QoS objective, because such data do not represent the actual resource needs of the VM and cannot be used to train the VM's resource model. Caution also needs to be taken to ensure that sufficient qualified data points are available in time so that the desired model can be quickly created as discussed later.

The Adaptive Leaner builds a fuzzy rule base from the qualified input-output data to model a database VM's resource usage behaviors. However, it is not efficient to generate one rule for every specific data pair, which may also lead to over-fitting due to error or noise in the data. In order to build a concise rule base with a small number of fuzzy rules that can effectively represent the VM behaviors, clustering method is used to group similar data points into clusters. Specifically, the Learner adopts an efficient one-pass clustering algorithm, subtractive clustering [22]. This method starts from assuming each data point as a potential cluster center and selecting the cluster center with the most number of neighbors within a certain radius. After removing the data points belonging to the previous cluster, it continues to determine the next center for the remaining data until every data point belongs to a certain cluster. Once the clustering completes, each resulting cluster exemplifies a representative characteristic of the system behaviors and can be used to create a fuzzy rule accordingly.

Tthe Adaptive Leaner generates Sugeno-type fuzzy rules [23] from the clustered data for VM resource usage modeling. This type of fuzzy rules uses a crisp, linear or constant function as the membership function, which is suitable for mathematical analysis. To elaborate on this modeling process, 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 following:

R^i: IF input $[C_1, C_2, \ldots, C_N]$ is in cluster i,

THEN output $[R_{CPU}, R_{IO}]^T = \boldsymbol{A_i}[C_1, C_2, \ldots, C_N]^T + \boldsymbol{b_i}$, $0<i<K$

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, $\mu(w) = e^{-\frac{(w-c)^2}{2\sigma^2}}$, 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. In the consequence of a fuzzy rule, the output $r(t)$ is a linear function of *w(t)*, where the matrix $\boldsymbol{A_i}$ and vector $\boldsymbol{b_i}$ are fitting parameters estimated using the least-squares method.

The above modeling process is performed continuously online 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 the change of the workload's composition of queries with different types of resource demands. It can also occur due to the change of the database's query optimization strategy. In order to adapt to these dynamic changes, the Adaptive Learner continuously updates the VM's resource usage model based on the data collected by the Sensors in real time. So when a transition happens, new data points that reflect the workload's current characteristics and the VM's current resource usages are fed to the Adaptive Learner for modeling. A new set of clusters are discovered from these data to represent the current characteristics in the database VM's model. Finally, the fuzzy model is updated with a new set of fuzzy rules that represent the VM's current resource usage behaviors for its current workload. 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 as opposed to a fixed structure model, and the learning process is incremental and automatic. Owing to the speed of subtractive clustering and fuzzy modeling, this whole model updating process can be completed quickly within a find-grained resource control interval.

C. *Resource Predictor*

With the fuzzy model created from the Adaptive Leaner, the Resource Predictor module performs fuzzy inference to generate an estimate of the resource demand *r(t)* given its current workload *w(t)* collected from the Application Sensor. In a clustering-based Sugeno-type fuzzy model mentioned above, Gaussian membership function is used in the antecedent of each rule to fuzzify the input *w(t)* 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. A final output is then aggregated from all the weighted fuzzy outputs. The final amounts of resources estimated by the Predictor are considered as the demands of resources for the next resource control interval and sent to the Resource Allocator to guide the VM resource allocation.

D. *Resource Allocator*

In virtualized database hosting systems, a VM serves as a virtualized resource container to the hosted database, where different types of resources can be dynamically allocated to this container to servicing the database's workload. This is in contrast of traditional, non-virtualized database hosting, where a database's resource availability is statically defined by its physical host's configuration. In our proposed resource management system, the Resource Allocator module is responsible for periodically adjusting the allocations of multi-type resources to a database VM according to the Resource Predictor's estimation of its resource needs.

However, the Resource Allocator needs to deal with situations where the resource prediction given by the VM's fuzzy model is inaccurate and causes the database's query performance to diverge from the desired QoS target. This happens when the database workload is first started or when its resource usage behaviors are changing, because the fuzzy rule base does not have the necessary rules to reflect these new behaviors yet. Such inaccuracy in the VM resource estimation is addressed by the resource management system according to the following two different scenarios.

If the VM's resource needs are underestimated, it will lead to performance drop for the workload to perform below its QoS target. To quickly recover from the performance loss due to this underestimation, the Resource Allocator invokes a backup resource allocation policy after the QoS target is missed for several consecutive periods of time. This backup policy increases the current resource allocation by a fixed percentage in order to satisfy the VM's unknown resource needs which are beyond the previous resource allocation level. This fixed increment on VM 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 consecutive periods of time. Because the VM resource usage can be monitored and controlled at a fine granularity, this backup policy allows the resource underestimation to be quickly recovered and the performance loss to be small and transient. Meanwhile, it is also important for fast model adaptation, because it allows qualified data points to become quickly available so that the model can be updated to reflect the current resource needs and produce correct resource estimations in a timely manner.

If a VM's resource needs are overestimated, it will lead to either underutilization of the allocated resources or overachieving of the query QoS. In the former case, the VM's actual resource needs are already reflected by its current resource usages, which will be collected by the Application Sensor and taken by the Adaptive Learner to update the model and correct the inaccuracy automatically. In the latter case, the model does not have the knowledge of the VM's current resource needs, so another backup policy also needs to be invoked to reduce the resource allocation by a fixed amount. Its usage is similar to the use of the backup policy described in the above resource underestimation scenario, except that the resource allocation is decreased instead of being increased. This ensures that the resource allocation can be quickly brought down to the necessary level so that efficient VM resource usages can be achieved while meeting the desired query QoS target.

Finally, the Resource Allocator also needs to handle the situation where the available resources on a physical host are not sufficient to satisfy the needs from all the concurrently hosted VMs. If applicable, some VMs can be migrated to other hosts in order to reduce the resource contention. Otherwise, the Resource Allocator needs to perform a cross-VM optimization based on their SLAs (Service-level Agreements) and to maximize the total profit that can be obtained from the VMs. This is not the focus of this paper, and our previous work of profit-driven cross-VM resource optimization can be leveraged here, which maximizes the

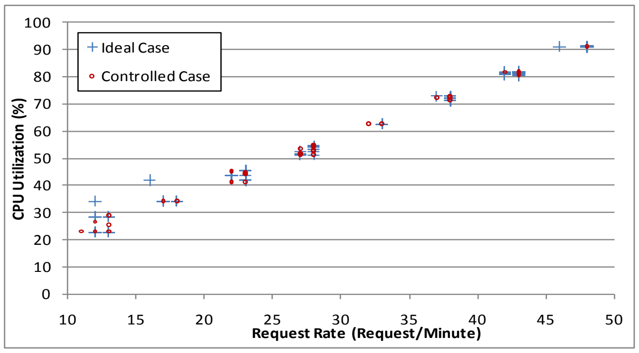

Figure 2. CPU usage for the CPU intensive workload

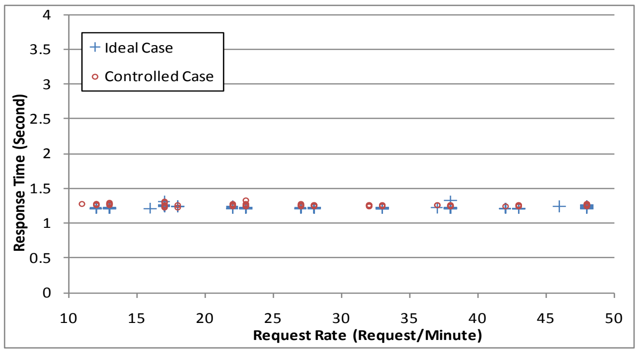

Figure 3. Average query response time for the CPU intensive workload

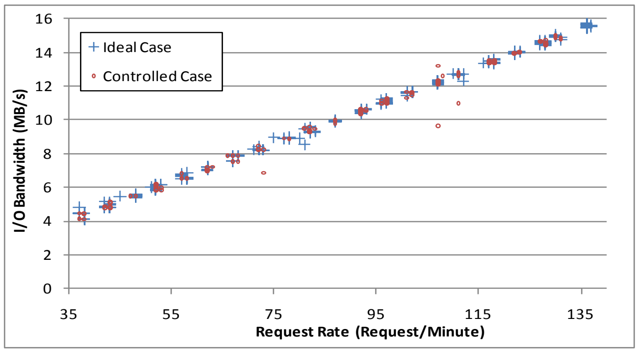

Figure 4. I/O bandwidth usage for the I/O intensive workload

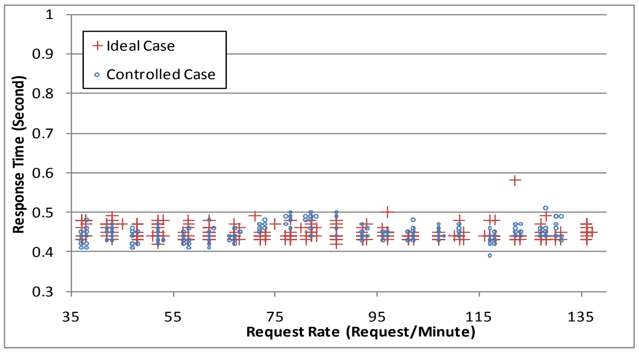

Figure 5. Average query response time for the I/O intensive workload

total profit by considering both the revenues from hosting applications and the penalty from unsatisfied QoS [18].

Typical VM technologies all support dynamic and fine-grained adjustment to a VM's resource configuration. In a Xen-based VM environment, the Resource Allocator is implemented as a user-level daemon running on the privileged management VM. Specifically, the allocation of CPU cycles is done through the sEDF CPU scheduler [24] implemented by Xen. The CPU share for a VM is presented by both slice *s* and period *p* in a way that *s* units of time in each period of length *p* is received by that VM. Xen does not directly support the allocation of disk I/O bandwidth to VMs, but it is implemented in our resource management system using *dm-ioband* [25], a Linux block I/O bandwidth controller, by throttling the VM's use of I/O bandwidth to the physical disk where its disk image file is stored.

IV. EVALUATION

A. Setup

This section evaluates our proposed autonomic resource management system using database benchmarks hosted on a typical VM environment. The testbed is deployed on a quad-core Intel Q6600 2.4GHz physical machine, which has 4GB of memory and 142GB of SATA disk storage. Xen 3.3.1 is installed to provide the VMs, where the OS for both dom0 and domU VMs is Ubuntu Linux 8.10 with paravirtualized 2.6.18.8. A dedicated domU VM is used to host the database server, and our resource management system is hosted on a separated VM. The VM's use of CPU and disk I/Os are collected using *xentop* and *iostat*, where the I/O bandwidth usage is the sum of reads and writes per unit time. Resource allocations to VMs use the sEDF CPU scheduler to assign CPU shares and using *dm-ioband* I/O controller to set the cap for I/O bandwidth. The sEDF scheduler uses 100ms period in the work-conserving mode. The VM resource monitoring and control period is set to 20 seconds in Section IV.B and 10 seconds in Section IV.C. The overhead from the management system is very small, which uses about 20MB memory and 1% CPU when measured every second.

Two typical database benchmarks are considered in our experiments, including TPC-H [3] and RUBiS [4]. Their performance is measured in two different cases: the *controlled case*, where the database VM's resource usages are controlled based on our proposed approach; and the *ideal case*, where there is no restriction on the database VM's resource usages. By comparing VM resource usages and workload performance between the controlled case and ideal case, these experiments evaluate whether our proposed system can correctly estimate the database VM's resource needs, achieve the same level of performance as in the ideal case, and save substantial resources compared to peak-load based static resource allocation. Every experiment is repeated multiple times and the results from the average case are reported here. Each run of the experiments is started with cold memory cache by restarting the database.

B. TPC-H Experiments

TPC-H provides 22 representative queries in business decision support system. The executions of these queries involve the processing of large volumes of data with a high degree of complexity, which can result in diverse behaviors

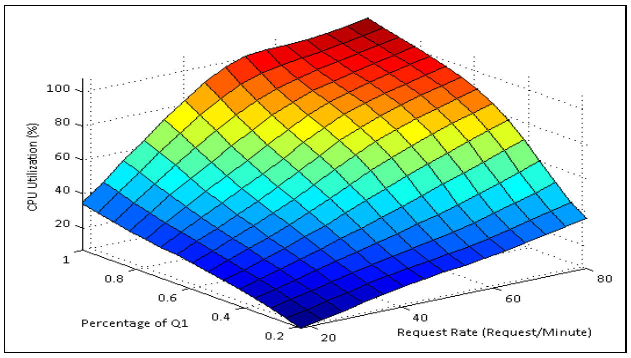

Figure 6. CPU model for the mixed CPU/IO intensive workload

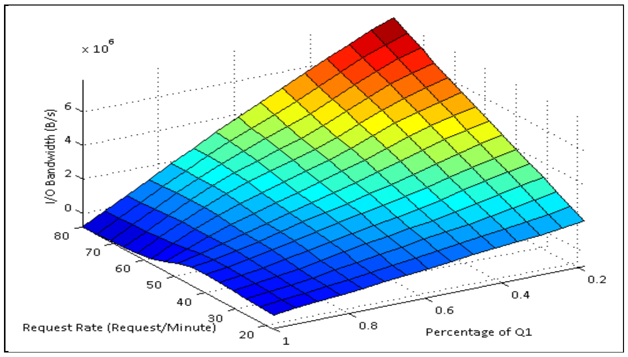

Figure 7. I/O model for the mixed CPU/IO intensive workload

of multi-type resource usages. Based on these queries, we have constructed synthetic database workloads with varying demands of different types of resources. These workloads are kept relatively simple in order to clearly demonstrate the impact of workload intensity and composition on the VM's resource usages. In these experiments, training and testing of the VM resource model are done separately using different subset of the available data points, in order to evaluate the accuracy of our fuzzy-modeling based approach for learning and predicting the resource needs of a database VM. The modeling of more realistic and complex workloads with online training is considered in Section IV.C. The database used here is an instance of PostgresSQL 8.1.3 with 1.1 GB of data, hosted on a VM with one CPU and 512MB memory.

1) CPU-intensive Workload

The first experiment considers the Pricing Summary Report Query (*Q1*) from TPC-H, which reports the amount of business that was billed, shipped, and returned. It is a CPU intensive query that involves a series of SUM and AVG operations based on GROUP BY and ORDER BY clauses. The workload is generated by continuously issuing copies of Q1 to the database with an increasing request rate from 10 to 50 request/minute. Each rate is sustained for a period of 300 seconds before incremented to a higher rate. Different sets of the data are used for training and testing (100 points each).

Because the resulting workload is not I/O intensive, here we only show the results from the VM's CPU modeling and allocation. Figure 2 and 3 compare the CPU usage and the average query response time between the controlled case and the ideal case as the request rate in the test dataset increases. The results show that the learned model can be used to accurately predict the database VM's CPU needs for delivering the same performance as in the ideal case for such a CPU intensive workload.

2) I/O-intensive Workload

In the second experiment, the workload is generated in a way similar to the previous one, except that it uses the Forecasting Revenue Change Query (*Q6*) from TPC-H, which quantifies the amount of revenue increase resulted from eliminating certain companywide discounts in a given percentage range in a given year. This query examines a large table in the database and is highly I/O intensive. The size of this table is reduced in this experiment to make Q6 less I/O intensive, so that we can vary its request rate in a larger range (35~135 request/minute). Further, in order to show the workload's worst case I/O demand, this downsized table is duplicated many times so that contiguous Q6 requests access different copies of the table and the memory caching cannot help to speed up the query processing. The training dataset has 250 points and the test set has 200 points.

Because the resulting workload is not CPU intensive, here we only show the results from the VM's I/O modeling and allocation. Figure 4 and 5 compare the I/O usage and the average query response time between the controlled case and ideal case for the test dataset. The results show that our approach can also accurately model the I/O bandwidth needs of such an I/O intensive workload and meanwhile achieve the best performance as in the ideal case.

We recognize that the prediction given by our approach for the database VM's I/O needs is the upper bound, because memory caching can leverage data locality to reduce the VM's actual I/O usage. Nonetheless, compared to peak-load based static resource allocation, it can still save substantial I/O bandwidth which can be used to satisfy the needs of other VMs. Further improvement upon this upper bound estimation requires knowledge of database system's memory cache usage, which will be considered in our future work.

3) Mixed CPU/IO-intensive Workload

In the third experiment, the previously considered two queries, Q1 and Q6, are mixed together to construct a workload that is both CPU and I/O intensive. The total request rate of this mixed workload is increased from 20 to 80 request/minute, where at each rate the composition of the mixture workload is also changed by varying the ratio between Q1 and Q6 from 0.25 to 4. This experiment is designed to evaluate our proposed system's effectiveness of modeling a database VM serving workloads with changing query request rate and query composition, both of which have a significant impact on the VM's resource usages. Correspondingly, the workload is characterized by both the overall request rate and the ratio between Q1 and Q6. We use 450 data points for training and 350 for testing.

Figure 6 and 7 illustrate the database VM's resource models learned from the training dataset, which show two three-dimensional nonlinear relationships between the query workload's request rate and Q1/Q6 ratio and the VM's CPU and I/O usages. The results demonstrate that our system can properly capture such complex behaviors in VM resource usages. Figure 8 and 9 compare the CPU and I/O usages between the controlled case and ideal case when the request

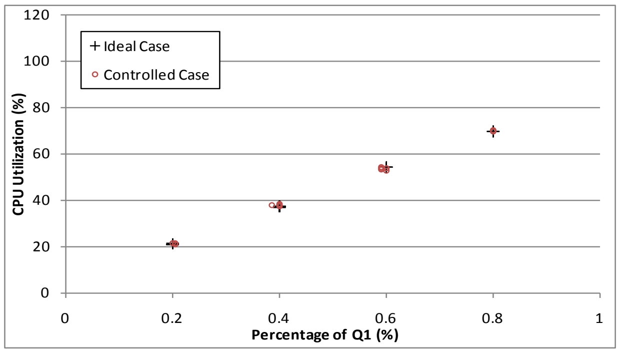

Figure 8. CPU usage for the mixed CPU/IO intensive workload when the request rate is 45 request/minute

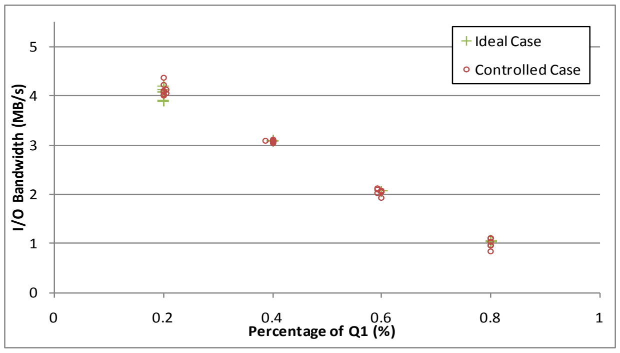

Figure 9. I/O bandwidth usage for the mixed CPU/IO intensive workload when the request rate is 45 request/minute

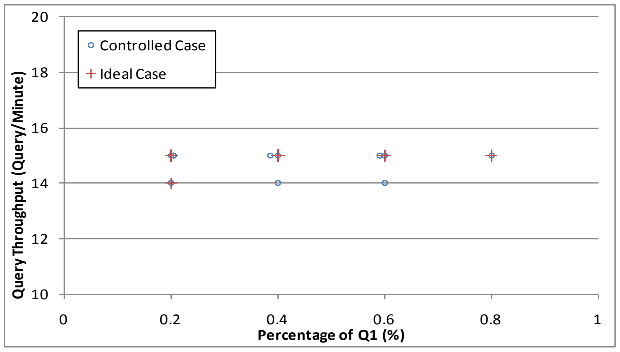

Figure 10. Average query response time for the mixed CPU/IO intensive workload when the request rate is 45 request/minute

rate is 45 request/minute and the percentage of Q1 varies from 0.2 to 0.8 in the test dataset; Figure 10 compares the query throughput between these two cases in this setup. (The results from other request rates are similar and omitted here for conciseness.) The results show that the VM's resource model can accurately estimate both its CPU and I/O demand simultaneously for delivering the ideal performance to such a workload with mixed CPU and I/O intensive queries.

C. *RUBiS*

RUBiS models an online auction site that supports the core functionalities such as browsing, selling, and bidding [4]. It is used to evaluate our proposed approach's accuracy and adaptability for modeling the resource needs of a database VM servicing a highly complex workload with dynamically changing characteristics. A typical two-tier setup is employed in these experiments, where the Web server and database server are deployed on separated VMs. The Web VM hosts Apache Tomcat 4.1.40 with the RUBiS website and its clients, and it is configured with one CPU and 512 MB of memory. The database VM hosts MySQL 5.0 with 1.1 GB of data and it is configured with one CPU and 1 GB of memory. Each client session opens a persistent HTTP connection to the Web server, where the requests represent a variety of interactions between a user and the auction site, which mainly include browsing items, bidding, and buying and selling items. These HTTP requests will then trigger corresponding queries to the database server.

Three types of RUBiS workloads are considered, which have different mixes of user interactions: The *browsing mix* simulates users' browsing requests on the auction site which include only read-only interactions; the *bidding mix* also simulates users' bidding and selling requests, which account for 15% of the entire workload's requests, so it has both read and write interactions with the auction site; the *high-bidding mix* simulates even more bidding and selling requests (30% of the entire workload's requests) and is thus more write-intensive than the other two mixes. The think time in all RUBiS mixes are generated from a negative exponential distribution with a mean value of seven seconds.

The number of clients and the composition of the different types of user requests have strong impact on the corresponding database workload's query request rate and usage of different types of resources. Therefore, they are both considered as input to the modeling of the database VM's resource needs and they can be measured from the Web tier which interfaces with users on the auction site. Unlike the TPC-H experiments, in this experiment, the training of VM resource model is done completely *online*, i.e., the model is continuously updated as new data come in while it is applied to predict VM resource needs.

The first RUBiS experiment studies our proposed approach's accuracy and adaptability for a *workload with dynamically changing intensity*. The 15% bidding workload mix is used with its number of clients varying from 100 to 300, and then to 500, where at each value it is sustained for a period of 300 seconds and then incremented to a higher rate without up ramp and down ramp. Figure 11 and 12 compare the database VM's CPU and I/O usages between the controlled case and ideal case; Figure 13 compares the query throughput between these two cases. The results show that both resource usages and query throughput in the controlled case are consistent with the ideal case. It proves that our approach can properly allocate resources to the VM for such a complex workload to deliver the ideal performance. It does so without dedicating the entire physical host to the VM and thus saves substantial CPU cycles and disk I/O bandwidth.

This second RUBiS experiment evaluates our approach for a *workload with dynamically changing query mix*. It is done by dynamically switching the workload from the browsing mix to the bidding mix, and then to the high-bidding mix, where each one lasts 600 seconds and transits to next mix without up ramp and down ramp. The number of concurrent client sessions is kept at 100 throughout the

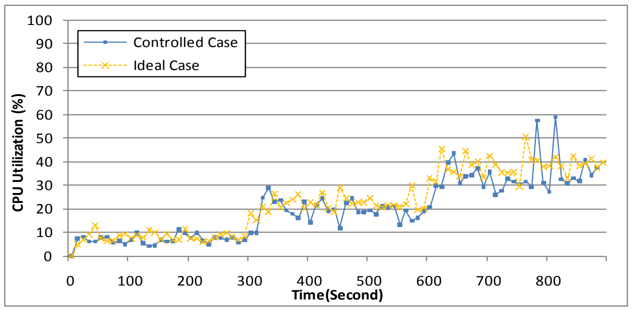

Figure 11. CPU usage for the RUBiS workload with changing intensity

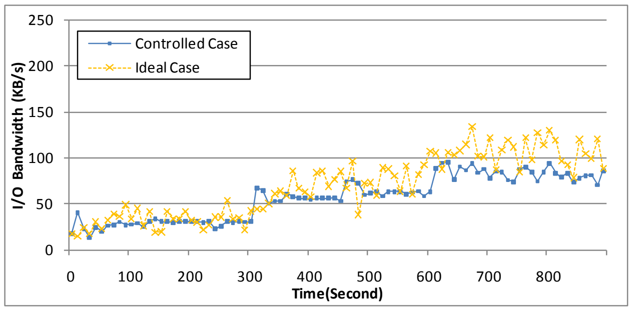

Figure 12. I/O usage for the RUBiS workload with changing intensity

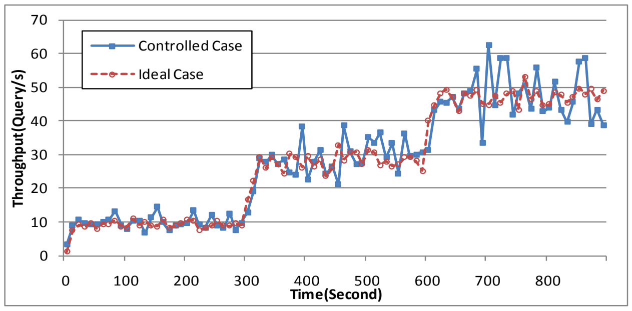

Figure 13. Throughput for the RUBiS workload with changing intensity

experiment. Figure 14 compares the database VM's I/O bandwidth usage between the controlled case and ideal case (the comparison of CPU usage is omitted here because this workload is not CPU intensive); Figure 15 compares the query throughput between these two cases. These results show that, during most of the experiment, the controlled case can closely follow the ideal case in terms of both I/O usage and query performance for such a complex workload.

However, when the workload switches from the first mix to the second mix at the 610 second, the controlled case's performance becomes temporarily worse than the ideal case, as the model learned from the previous mix's behaviors is no longer suitable for estimating the new mix's I/O bandwidth needs. As a result, the backup policy is automatically invoked when the QoS target is missed for three consecutive periods and an additional fixed amount of I/O bandwidth (100%) is allocated upon the model's estimated I/O needs, in order to quickly bring the QoS back to the target level and meanwhile allow new qualified data to become available for updating the model. This explains the temporary over-allocation in the controlled case compared to the ideal case during the transition phase (from 640 to 720 second). The results also show that the QoS in the controlled case indeed quickly converges to the ideal case after the transition. The backup policy is stopped when the QoS target is met for three consecutive periods, and the model's prediction is again used to allocate I/O bandwidth as normal as it can now correctly estimate the VM's I/O needs for the new workload mix. (Although not shown in the results, similar behaviors are also observed in the previous RUBiS experiment when the workload transits from 100 concurrent clients to 300.)

Note that this experiment setup reveals the worst-case performance of our system because the transitions between different workload mixes happen suddenly and abruptly. Due to the nature of RUBiS, a surge in request rate also occurs whenever a workload mix starts, which makes it even more challenging to handle the transtion. However, for real-world workloads, such transitions are typically incremental and slow, which would allow our system more time to adapt to the change and reduce the performance loss during the transitions. Nonetheless, even for such a challenging setup, our system can still properly deal with the transitions and have short adverse impact on the query performance, which further proves its good adaptability.

V. 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. They generally employ modeling or control techniques to determine VM resource configurations and adopt autonomic techniques [10][11] to automatically optimize VM resource allocations. Although there is also a significant body of related work on the performance modeling of non-virtualized systems, due to the limited space, the discussions here focus only on virtualized resource management. Nonetheless, the solutions proposed for non-virtualized autonomic resource management can also fall into the several categories discussed below.

The first category of solutions employs *queuing theory* to construct analytical performance models for applications executed on VMs. For example, Doyle *et al.* derived analytical models from basic queuing theory to predict response times of Internet services under different load and resource allocation [26]; Bennani *et al.* considered using multiclass queuing networks to predict the response time and throughput for online and batch workloads on VM based application environments [27]. However, the effectiveness of this type of solutions is restricted by their assumptions about a virtualized system's internal model, which are often insufficient for capturing its resource usage complexities.

The second category of solutions applies control theory to automatically adjust VM resource allocation in order to achieve the desired application performance. Such solutions often assume a linear relationship between QoS parameters and control parameters and involve a system identification phase to train the model parameters. In addition, the control parameters typically must be specified or configured offline on a per-workload basis. For example, autoregressive models have been used to map CPU allocation to average response times of Web servers [28][29]; Autoregressive moving

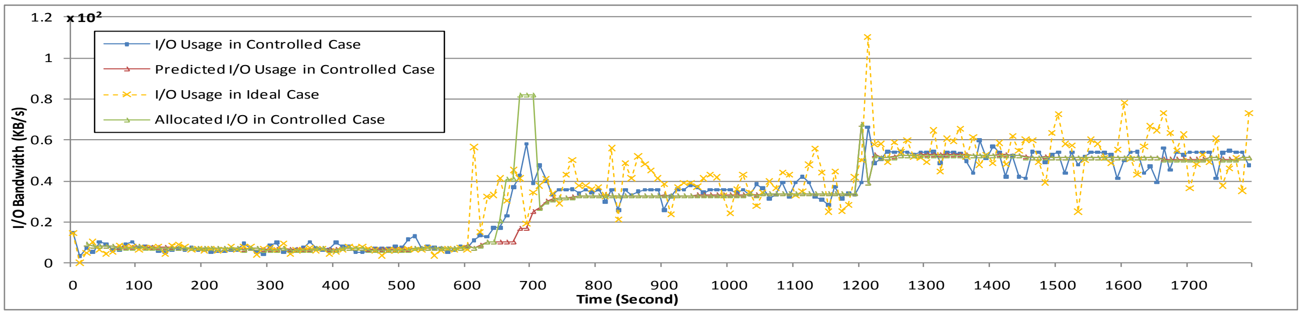

Figure 14. I/O bandwidth usages for the RUBiS workload with changing mix

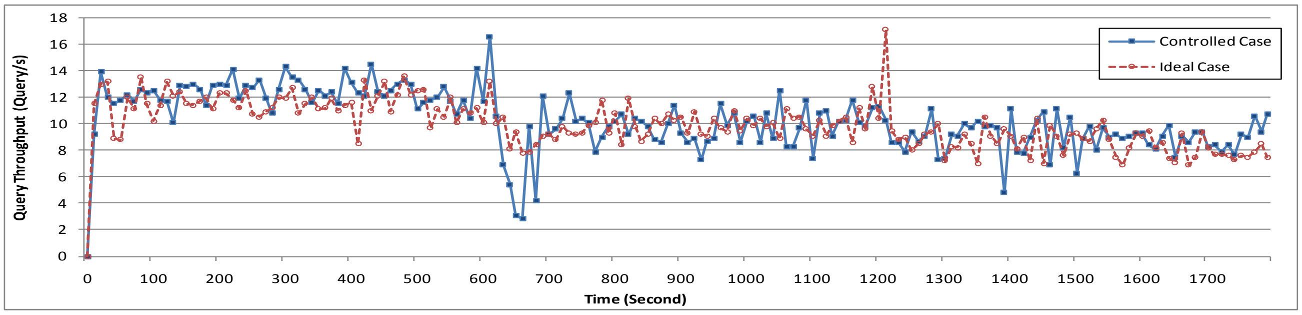

Figure 15. Query throughput for the RUBiS workload with changing mix

average models have also been considered to represent a more general relationship between application performance and its VM's multi-type resource usages [30]; The "1000 islands" work is an integrated, control-theory based approach to virtualized datacenter resource management using both dynamic resource allocation and VM migration [31].

In comparison, the third category of solutions considers machine learning techniques to automatically learn the complex resource model for a virtualized system based on data observed from the system. For example, the CRAVE project employs simple regression analysis to predict the performance impact of memory allocation to VMs [13]; Wood *et al.* also use regression method to map a resource usage profile obtained on a physical system to one that can be used on a virtualized system [14]; The VCONF project has studied using reinforcement learning to automatically tune the CPU and memory configurations of a VM in order to achieve good performance for its hosted application [15]; Kund *et al.* employs artificial neural networks to build performance models that consider both resource allocation to VMs and resource interference between VMs [16].

Our proposed fuzzy-logic based VM modeling solution also falls into the third category. It is advantageous in that it does not require any *a priori* knowledge of the VM's resource usage behaviors, and it can efficiently model a nonlinear system with dynamically changing operating conditions. Our previous work [18] has studied fuzzy modeling for the CPU resource management of VMs hosting CPU-intensive Web servers. This paper takes this approach to address the new challenges in VM-based hosting of applications that have more complex, multi-type resource usage behaviors. Specifically, our system is able to manage and optimize VM's use of not only CPU but also I/O resource, which is known to be more difficult to model and control. It is also capable of handling dynamic workloads with shifting demands across different resource types.

Research on the resource management of virtualized database hosting systems is still in its infancy. Farooq *et al.* experimentally evaluated VM-based databases and showed that the overhead is very small compared to natively hosted databases, thus justifying the feasibility of such approaches [31]. Soror *et al.* addressed the problem of automatic VM resource configuration for hosting databases by leveraging DBMS query cost model, which is calibrated to reflect the cost when queries are executed on VMs [32]. 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 DBMS cost model is inaccurate, this work employs online refinement by assuming a linear model between workload performance and VM resource allocation. Therefore, it is unclear how this approach would apply to and how well it would perform for a workload with complex resource usages and dynamically changing behaviors. In comparison, our proposed approach is capable of addressing such dynamism and complexity which are typical in database workloads and virtualized systems.

VI. Conclusion 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 demands and QoS targets. This paper presents an autonomic resource management system that can achieve this goal through fuzzy modeling based approach, which learns 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 continuously online to guide dynamic resource allocation and adapt to dynamic changes in the system. Experiments based on typical database benchmarks demonstrate that our system can accurately estimate a database VM's resource needs for dynamic and complex query workloads, and it can save substantial resources compared to peak-load based static allocation while achieving the desired query QoS. Finally, the approach taken by this research is also generally applicable to address the virtual resource management for other types of applications that have dynamic and complex resource usage behaviors.

In our future work, we will consider to further improvements along the following directions. First, our current approach treats an application's VM as a black box in resource management, which has the advantage of being agnostic to application specifics and thereby applicable to different applications. However, application knowledge can be carefully leveraged to further improve its effectiveness and efficiency. In particular, for database VMs, online workload characterization can take advantage of the query cost estimation from DBMS, whereas more efficient I/O bandwidth allocation can be based on the knowledge of DBMS buffer management. Second, our current system manages each VM separately based on its individual workload and QoS target. Nonetheless, for VMs hosting dependent applications, e.g., different tiers of a multi-tier application, their resource management can be considered holistically, in order to further optimize the overall resource usage and achieve the end-to-end QoS goals.

REFERENCES

[1] Carl A. Waldspurger, "Memory resource management in VMware ESX server", Proceedings of the 5th symposium on Operating systems design and implementation, December 2002.

[2] P. Barham et al., "Xen and the Art of Virtualization", in Proc. of the ACM Symposium on Operating Systems Principles, October 2003

[3] TPC-H Benchmark Specification, URL: http://www. tcp. org.

[4] C. Amza et al., "Specification and Implementation of Dynamic Web Site Benchmarks", In Proceedings of WWC-5: IEEE 5th Annual Workshop on Workload Characterization, October 2002.

[5] Linux Vserver, http://linux-vserver.org/.

[6] OpenVZ, http://wiki.openvz.org/.

[7] A. Chen et al., "Heuristics for Selecting Robust Database Structures with Dynamic Query Patterns", European Journal on Operational Research, 168(1):200–220, January 2006.

[8] M. Zhang et al., "Data Mining Meets Performance Evaluation: Fast Algorithms for Modeling Bursty Trafic", In Proceedings of the 18th ICDE Conference, pages 507–516, Feburary 2002.

[9] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling", Transactions on Networking, 3(3):226–244, 1995.

[10] J. O. Kephart, D. M. Chess, "The Vision of Autonomic Computing", IEEE Computer, 36(1): 41-50, 2003.

[11] Steve White, James Hanson, Ian Whalley, David Chess, and Jeffrey Kephart., "An Architectural Approach to Autonomic Computing", In Proc. 1st International Conference on Autonomic Computing, 2004.

[12] J. Matthews, et al., "Quantifying the performance isolation properties of virtualization systems", in Proceedings of the 2007 workshop on Experimental computer science, 2007.

[13] Jonathan Wildstrom, Peter Stone, Emmett Witchel, "CARVE: A Cognitive Agent for Resource Value Estimation", in Proc. of 5th IEEE International Conference on Autonomic Computing, 2008.

[14] T. Wood, L. Cherkasova, K. Ozonat, P. Shenoy, "Profiling and Modeling Resource Usage of Virtualized Applications", Proc. of the 9th International Middleware Conference, December, 2008.

[15] Jia Rao et al., "VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration", in Proc. of 6th IEEE International Conference on Autonomic Computing (ICAC), 2009.

[16] Sajib Kundu et al., "Application Performance Modeling in a Virtualized Environment", in Proc. of 16th IEEE International Symposium on High-Performance Computer Architecture, 2010.

[17] L. A. Zadeh, "Fuzzy Sets", Information and Control, vol. 8, no. 3, pp. 338-353, June 1965.

[18] Jing Xu et al., "Autonomic Resource Management in Virtualized Data Centers Using Fuzzy-logic-based Control", Cluster Computing, Vol. 11, No. 3, Pages: 213-227, September 2008.

[19] Stefan Krompass, Umesh Dayal, Harumi Kuno and Alfons Kemper, "Dynamic Workload Management for Very Large Data Warehouses: Juggling Feathers and Bowling Balls", In Proc. of International Conference on Very Large Data Bases, pages 1105-1115, 2007.

[20] Patrick Martin, Said Elnaffar, and Ted J. Wasserman, "Workload Models for Autonomic Database Management Systems", In Proc. of Intl. Conference on Autonomic and Autonomous Systems, 2006.

[21] S. Chaudhuri, "Relational Query Optimization – Data Management Meets Statistical Estimation", Communications of ACM 52(10):86-86, October 2009.

[22] S. Chiu, "Fuzzy Model Identification Based on Cluster Estimation", Journal of Intelligent and Fuzzy Systems, Vol. 2, No. 3, 1994.

[23] M. Sugeno and T. Yasukawa, "A Fuzzy-Logic-Based Approach to Qualitative Modeling", IEEE Trans. on Fuzzy Systems, 1(1), 1993.

[24] I. Leslie et al., "The Design and Implementation of an Operating System to Support Distributed Multimedia Applications", IEEE Journal of Selected Areas in Communications, 1996.

[25] dm-ioband, URL: http://sourceforge.net/apps/trac/ioband.

[26] R. Doyle, J. Chase, O. Asad, W. Jin, and A. Vahdat., "Model-Based Resource Provisioning in a Web Service Utility", in Proc. of the 4th USENIX Symposium on Internet Technologies and Systems, 2003.

[27] M. Bennani and D. Menascé, "Resource Allocation for Autonomic Data Centers using Analytic Performance Models", in Proc. of 2nd IEEE International Conference on Autonomic Computing, 2005.

[28] X. Liu, X. Zhu, S. Singhal, M. Arlitt, "Adaptive Entitlement Control of Resource Containers on Shared Servers", in Proc. of 9th IFIP/IEEE International Symposium on Integrated Network Management, 2005.

[29] Z. Wang, X Zhu, S. Singhal, "Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions", in Proc. of 16th IFIP/IEEE Distributed Systems: Operations and Management (DSOM), 2005.

[30] Pradeep Padala et al., "Automated Control of Multiple Virtualized Resources", In Proc. of the 4th ACM SIGOPS/EuroSys European Conference on Computer Systems, 2009.

[31] X. Zhu et al., "1000 Islands: Integrated Capacity and Workload Management for the Next Generation Data Center", Proc. of the 5th IEEE International Conference on Autonomic Computing, 2008.

[32] Umar Farooq Minhas, Jitendra Yadav, Ashraf Aboulnaga, Kenneth Salem, "Database Systems on Virtual Machines: How Much do You Lose?", Intl. Workshop on Self-Managing Database Systems, 2008.

[33] Ahmed Soror et al., "Automatic Virtual Machine Configuration for Database Workloads", Proceedings of ACM SIGMOD International Conference on Management of Data, pages 953-966, June 2008.