Application-aware Cross-layer Virtual Machine Resource Management

Lixi Wang Jing Xu Ming Zhao

School of Computing and Information Sciences
Florida International University, Miami, FL, USA
{lwang007,jxu,ming}@cs.fiu.edu

ABSTRACT

Existing resource management solutions in datacenters and cloud systems typically treat VMs as black boxes when making resource allocation decisions. This paper advocates the cooperation between VM host- and guest-layer schedulers for optimizing the resource management and application performance. It presents an approach to such cross-layer optimization upon fuzzy-modeling-based resource management. This approach exploits guest-layer application knowledge to capture workload characteristics and improve VM modeling, and enables the host-layer scheduler to feedback resource allocation decision and adapt guest-layer application configuration. As a case study, this approach is applied to virtualized databases which have challenging dynamic, complex resource usage behaviors. Specifically, it characterizes query workloads based on a database's internal cost estimation and adapts query executions by tuning the cost model parameters according to changing resource availability. A prototype of the proposed approach is implemented on Xen VMs and evaluated using workloads based on TPC-H and RUBiS. The results show that with guest-to-host workload characterization, resources can be efficiently allocated to database VMs serving workloads with changing intensity and composition while meeting Quality-of-Service (QoS) targets. For TPC-H, the prediction error for VM resource demand is less than 3.5%; for RUBiS, the response time target is met for 92% of the time. Both significantly outperform the resource allocation scheme without workload characterization. With host-to-guest database adaptation, the performance of TPC-H-based workloads is also improved by 17% when the VM's available I/O bandwidth is reduced due to contention.

Categories and Subject Descriptors

C.4 [**Performance of System**]: Modeling techniques;
I.5.1 [**Pattern Recognition**]: Models – *fuzzy set*

Keywords

Autonomic computing, Fuzzy modeling, Resource management, Virtualization

1. INTRODUCTION

With the rapid growth of computational power on compute servers and the fast maturing of x86 virtualization technologies, virtual machines (VMs [1][2]) are becoming increasingly important in supporting efficient and flexible application and resource provisioning. Virtualization is the key enabling technology for building agile datacenters and emerging cloud systems [3][4]. It allows a single physical server to be carved into multiple virtual resource containers, each delivering a powerful, secure, customizable, and portable execution environment for applications. As the level of VM-based consolidation continues to grow, there is an increasingly urgent need for virtualized systems to deliver better Quality-of-Service (QoS) guarantees, so that users are comfortable in running their applications on the shared infrastructure. However, currently such systems cannot meet stringent performance requirements, particular not for applications with dynamic and complex behaviors. Consequently, examples such as cloud systems cannot support QoS-based Service Level Agreements (SLA), whereas users often have to purchase unnecessary resources for their VMs.

Existing resource management solutions typically treat VMs as black boxes when making resource allocation decisions. The host-layer VM scheduler is agnostic of the guest-layer application-specific resource scheduling, whereas a guest-layer application scheduler is unaware of the host-layer VM resource allocation. Although such transparency is important for reasons such as portability and legacy support, it also prevents the resource management effectively providing application-desired QoS. On one hand, the knowledge of an application's workload characteristics can help the host-layer resource management to better understand the VM's resource demand and meet the application's QoS target. On the other hand, the knowledge of the host's VM allocation decision can help the guest-layer resource management understand the actual resource availability and adapt its scheduling to improve application performance.

Therefore, this paper proposes cross-layer optimization in VM resource management which allows certain awareness and cooperation between host and guest in order to improve application performance and meet its QoS target. Specifically, this paper studies two aspects of such cross-layer optimization. First, *guest-to-host optimization* exploits guest-layer application knowledge to capture dynamic workload characteristics and improve the modeling of VM resource usage. Second, *host-to-guest optimization* enables the host-layer scheduler to feedback resource allocation decision and adapt guest-layer application configuration. These two aspects of cross-layer optimization are integrated into a fuzzy-modeling-based resource management system [5] which uses fuzzy logic to model VM resource demand online and allocate resource dynamically according to application QoS requirement.

This paper considers virtualized databases as an interesting and challenging case study. Databases often serve complex and dynamic workloads which consist of a variety of queries with different types and amounts of resource demand. Moreover, databases typically employ sophisticated optimization schemes which adapt query executions according to their resource availability. Hence, applying cross-layer optimization to the resource management of virtualized databases can be a convincing showcase of our proposed approach. Specifically in this case study, the proposed cross-layer optimization approach performs workload characterization based on database's internal cost model and adapts query executions by tuning the cost model parameters according to changing resource availability.

This proposed system is prototyped on Xen-based VM environments and evaluated by experiments using typical database workloads created based on TPC-H [6] and RUBiS [7] benchmarks. The results show that the fuzzy-modeling-based resource allocation with guest-to-host workload characterization can accurately predict the resource needs for complex application workloads. For TPC-H, it achieves less than 3.5% error for predicting VM resource demand; for RUBiS, it meets the response time target for 92% of the time. Both substantially outperform the resource allocation scheme without workload characterization, in terms of both application QoS and resource efficiency. Moreover, the results also show that our proposed approach of host-to-guest application adaptation effectively optimizes the database's query execution when the VM's resource availability changes due to I/O contention. The performance of a TPC-H workload is improved by about 17% compared to the scheme without such adaptation.

To the best of our knowledge, this paper is the first to study cross-layer optimization in VM resource management, considering both guest-to-host workload characterization and host-to-guest application adaptation. The case study demonstrates the effectiveness of this approach and provides an experimental evaluation. Compared to existing VM resource management solutions, this approach can accurately capture complex resource usage behavior for virtualize applications, timely adapt to dynamic changes in workloads, and optimize their performance under varying resource availability. In the rest of the paper, Section 2 presents the motivating examples, Section 3 introduces the background on fuzzy-modeling-based resource management, Section 4 and 5 present the general approach of cross-layer optimization and its case study on virtualized databases, Section 6 discusses the evaluation, Section 7 examines the related work, and Section 8 concludes the paper.

2. MOTIVATING EXAMPLES

In this section, we use several examples to motivate the need of cross-layer optimization in VM resource management, including both guest-to-host workload characterization and host-to-guest application adaptation.

2.1 Guest-to-Host Workload Characterization

For the first aspect of cross-layer resource management, we use an example to demonstrate that it is necessary for the host-layer VM scheduler to use the knowledge from guest-layer for workload characterization. Coarse-grained workload information such as the request rate or number of concurrent users can be easily obtained without knowledge about application internals. However, this information is no longer sufficient when the application workload consists of different types of requests with diverse usage of multiple types of resources. Here we use a concrete example based on a typical multi-tier OLTP benchmark, RUBiS [7] to demonstrate this limitation (Figure 1 and 2).

We fix the RUBiS' database tier's query workload intensity by running 300 concurrent client sessions in RUBiS. But we vary the composition of the query workload by increasing the ratio between bidding and browsing requests to the web tier, which corresponds to the ratio between read and write queries to the database tier. The entire experiment lasts for 600 seconds, starting with a browsing-only mix and then shifting to a 30%-bidding mix from the 300th second. The QoS target for this workload is set to 800ms. Without being aware of the changes in workload composition, the amount of resources needed by the RUBiS VM is estimated based solely on the workload intensity. Hence only 60KB/s I/O bandwidth is allocated to the RUBiS VM throughout the entire experiment (Figure 1). This allocation is enough for the workload to meet the QoS target in the first 300 seconds when the workload is not I/O intensive; but it leads to many QoS violations in the second 300 seconds due to the under-provisioning of I/O bandwidth (Figure 2). To address this problem, this paper proposes to exploit application-specific knowledge of workload characteristics in terms of different types of requests in order to make more accurate allocation decisions.

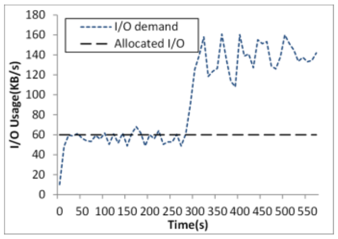

Figure 1 I/O Allocation for a changing mix in RUBiS

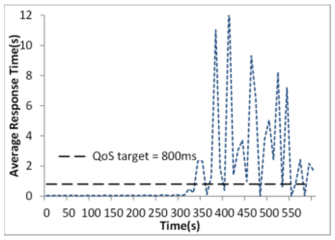

Figure 2 Performance for a changing mix in RUBiS

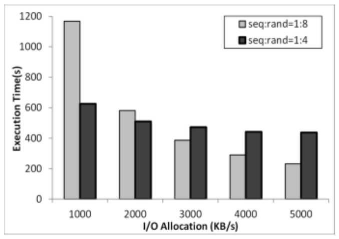

Figure 3 Execution time of TPC-H Q8 with varying I/O allocation

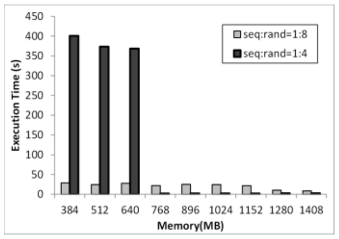

Figure 4 Execution time of TPC-H Q8 with varying memory allocation

2.2 Host-to-Guest Application Adaptation

We use other examples from virtualized databases to show the advantage of feeding back the information of resource availability from host- to guest-layer. We run a workload consisting of single copy of TPC-H query Q8 on a 3GB database VM, and manually set the database cost model parameters given different resource capacity. Figure 3 and 4 compare the query performance using two representative settings of the cost model parameters, *seq_page_cost* and *random_page_cost*. Both parameters characterize the database's execution environment: the former defines the cost of fetching a page from disk using sequential reads whereas the latter, usually more costly, defines the cost of a non-sequential disk page fetch. Changing these parameters affects the database performance indirectly by influencing the database's internal query cost estimation. Lower value of *seq_page_cost* reduces the cost of a plan with more sequential scans on the tables; lower value of *random_page_cost* reduces the cost of a plan with more random scans, e.g., index scans. Therefore, changing the ratio between these two parameters affects the database's preference on different execution plans.

In the first example (Figure 3), Q8 runs on a cold database VM, as the I/O bandwidth allocated to the VM is reduced from 5000 to 1000 KB/s. Both database configurations suffer from performance degradation with reduced available I/O bandwidth. However, when the available I/O bandwidth is high, the configuration that

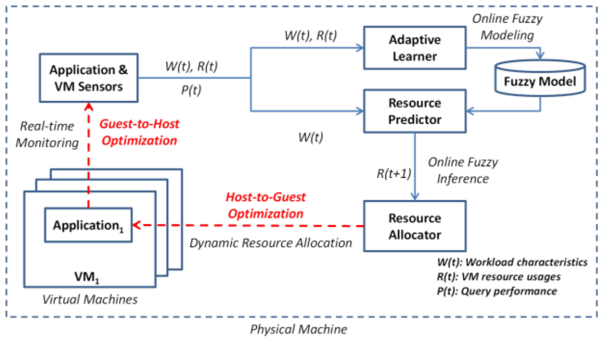

Figure 5 Architecture of cross-layer optimization on fuzzy-modeling-based resource management system

favors sequential scan outperforms that favors random scan (by 89% at 5000KB/s). When the I/O bandwidth is throttled, the latter's performance is only slightly affected and as a result it is 1.9 times better than the former at 1000 KB/s. The second example performed in a warm database VM shows similar behavior of Q8 performance but with respect to changing memory availability (Figure 4). When the available memory is low, the sequential-preferable DBMS configuration is drastically faster (by 14 times) because only a small amount of indices or tables can be cached in memory. As the memory allocation increases from 640 to 768MB, the indices can be effectively cached and consequently the index-preferable configuration's performance substantially outperforms the sequential-preferable configuration (by 3 times).

The above examples show strong evidence of the importance of host-to-guest optimization. If the database cost parameters can be adjusted dynamically to reflect the actual resource availability to the VM, the database performance can be further improved.

3. Fuzzy-modeling-based VM Resource Management

The main challenges to VM resource management are how to efficiently allocate resources to VMs and how to do so automatically and continuously. To address these challenges, our previous work [5][8] proposed fuzzy-modeling-based resource management to learn a VM's resource demand and allocate resources according to its QoS target in an autonomic manner. Fuzzy logic is used to create a VM's resource usage model automatically from data observed from the system without assuming any *a priori* knowledge about the system's structure. It is shown to be able to effectively capture complex, nonlinear resource usage behaviors in a virtualized system.

Figure 5 illustrates the architecture of our fuzzy-modeling-based resource management system. It consists of four key modules. As a workload executes on the VM, the *Application and VM Sensors* monitor the workload $W(t)$, its performance $P(t)$, and the VM's resource usage $R(t)$. The *Adaptive Learner* creates and updates a fuzzy model that represents the relationship between a workload and its VM's resource needs. With this model and the current workload $W(t)$, the *Resource Predictor* estimates the resource needs for time $t+1$ and the *Resource Allocator* adjusts the allocation accordingly. Together, these modules form a closed-loop for the VM's resource control and optimization.

Fuzzy logic is employed to build the model based on the qualified input-output data pairs, $<W(t), R(t)>$ whose workload performance $P(t)$ meet the desired QoS target. Both the workload input $W(t)$ and the resource usage output $R(t)$ can be vectors with multiple dimensions. This model captures the relationship between the application's workload and the VM's resource demand for meeting the QoS target. With the fuzzy model created by the *Adaptive Learner*, the *Resource Predictor* performs fuzzy inference to generate an estimate of the resource needs R given the workload input W. This estimation is then sent to the *Resource Allocator* to guide the VM's resource allocation. More details on fuzzy modeling can be found in our previous work [5][8].

In this paper we propose to further improve this existing fuzzy-modeling-based resource management system by incorporating cross-layer optimization between the VM host and guest, which is discussed in detail in the rest of this paper.

4. GENERAL APPROCAH TO CROSS-LAYER OPTIMIZATION

The goal of cross-layer optimization is to enable VM host- and guest-layer resource schedulers to communicate scheduling-related information and collaboratively improve the performance of a virtualized application and satisfy its QoS requirement. Existing resource management solutions typically treat VMs as black boxes when making resource allocations. The host-layer VM scheduler is agnostic of the guest-layer application-specific resource scheduling, whereas a guest-level application scheduler is also unaware of the host-layer VM resource allocation. Such transparency is important for reasons such as portability and legacy support, but for applications requiring strong QoS guarantees, a tradeoff can be made to allow certain awareness and cooperation between host and guest for meeting the QoS target.

Such cross-layer optimization is two-fold. First, the host-layer scheduler can leverage the guest-layer application-specific knowledge to improve the VM resource allocation decisions. Second, the guest-layer scheduler can adapt its application-specific scheduling based on the host-layer VM resource allocation to improve the application performance under changing resource availability. We will describe the general approach to both of these aspects of cross-layer optimization in this section.

4.1 Guest-to-Host Optimization

The guest-to-host aspect of our proposed cross-layer optimization is to exploit the guest-layer application-specific information to improve the understanding of the VM workload's resource usage patterns. Such knowledge will enable the host-layer resource scheduler to more accurately estimate the VM's resource demands and more agilely adapt to its workload changes. We propose to analyze an application's workload by describing it in terms of the characteristics that are relevant to its VM resource usage behaviors. Such characteristics provide important inputs to the effective modeling and prediction of the VM's resource needs. A commonly used workload characteristic is its overall intensity such as the total request rate or total number of online users. As shown in the motivating examples (Section 2.1), this characteristic alone is not sufficient for a real-world workload that consists of requests with diverse use of resources. As another example, a web workload consisting of only static web page has distinct resource needs versus one containing also considerable dynamic web page requests, even if their total request rates are exactly the same (the former consumes mainly CPU while the latter requires also substantial I/O bandwidth). Hence, it is important to characterize a workload's composition of different types of requests in terms of their resource usage patterns. But such characterization is difficult to do in existing resource management solutions which treat VMs as black boxes where application-specific knowledge is hidden.

To address this problem, we propose cross-layer optimization which allows a host-layer scheduler to exploit a guest-layer application's knowledge to understand the resource usage patterns of its received requests in the workload. For example, for web workloads, the web server's knowledge can be exploited to understand whether the received HTTP requests are targeting static or dynamic content. Such characterization of workload composition is key to understanding the VM's demands of CPU and I/O resources. For the workloads that contain more complex requests, such as in Online Analytical Processing (OLAP), more sophisticated application knowledge is required to analyze their resource usage patterns. We propose to characterize such workloads by leveraging the application's internal cost model, which is discussed in detail in Section 5.

The characterization of each individual request's resource usage pattern can be aggregated to describe the entire workload's resource usage characteristics. However, for workloads containing vast diversity of requests, it is impractical to describe all requests in the workload characterization. A concise representation is needed to effectively compress the information of all requests, which is critical to ensure low overhead and high robustness of the characterization. We propose to use data clustering techniques to group a workload's queries into clusters, so that those within a cluster are more similar in terms of their resource requirements to each other than the ones from different clusters. Assuming after the clustering a workload consists of *m* different groups of requests ($r_1, \ldots r_m$), the entire workload can then be characterized by the request rates of all these groups ($W_{r1}, \ldots, W_{rm}$), where each group represents a distinct resource usage pattern.

Many well established offline clustering algorithms are available for use, such as *K*-means, hierarchical clustering, subtractive clustering, etc. However, because of the dynamic nature of real-world workloads, the request cluster analysis should be carried out in an online fashion. To achieve this, we propose online, adaptive request clustering for an online, dynamic VM system, in which the clustering is performed in a way that is self-learning and self-adapting, without needing the number of clusters to be pre-specified. The basic idea is to perform one-pass, non-iterative clustering of a stream of requests using a method such as subtractive clustering. The procedure starts with an empty set of clusters and creates the first cluster with the first request sample assumed to be the cluster center. As more request samples come in, either a new cluster is added with the center based on the new data, or an existing cluster is removed or updated based on certain criteria (e.g., the radius set in subtractive clustering [9]). Such a clustering approach has the ability to gradually adapt to the changing data patterns. It can be applied to the data set of any size and allows flexible clustering with an evolving shape so that it can better match the current data distribution.

The above proposed workload characterization process will be performed online periodically (e.g., every 10s), in which the recently received requests will be used to update the workload's current clustering results. In this way, the characterization does not need *a priori* knowledge about all the queries that compose the workload, and it can dynamically adapt to the changing workload composition.

4.2 Host-to-Guest Optimization

The host-to-guest aspect of our proposed cross-layer optimization is to feed back the host-layer VM resource allocation decision and enable the guest-layer application-specific scheduling to adapt for better performance. Many applications need to be tuned to optimize their performance based on the resource availability of the hosting system. For example, a web server needs to tune parameters such as the number of concurrent threads based on its host's available memory. A database needs to tune its internal cost model (e.g., the CPU and I/O costs of processing a tuple) based on its host's resource availability so that it can correctly estimate the costs of different query execution plans and select the most efficient one to use. A web search engine may change its crawling, indexing, or searching strategies as the resource availability varies. When resource is constrained, it may crawler over only a portion of available web pages, restrict the depth of parsing and indexing on the searched contents, and return a limited number of best matching results to the users. Another example application is a simulator that can tune the modeling resolution based on its host's resource availability to increase the simulation accuracy or speed up the simulation progress [10].

When such an application is hosted on a physical machine, it needs to be tuned only once during the initial deployment. However, on a VM, the resource availability can vary over time, because of *1)* changing resource contention from other co-hosted VMs as they come and go dynamically and their workloads vary over time; *2)* changing resource allocation policy such as VM priorities or Service-level Agreements (SLAs). Nonetheless, the changing resource availability to a VM is hidden to the application in existing VM resource management solutions. As a result, the application is stuck with the initial configuration assuming a resource availability that is no longer valid. It cannot adapt itself to use a configuration that is more efficient in application performance and/or resource utilization when the VM's resource becomes either under pressure or abundant.

To address this problem, we propose cross-layer optimization for the host-layer scheduler to feedback the resource allocation decision to the guest-layer and automatically adapt the latter's configuration for improved performance given the current resource availability. The general approach to this host-to-guest optimization can be formally described as follows. Assuming that there are *M* different types of resources, such as memory, CPU capacity, or I/O bandwidth, $\boldsymbol{R}_i=[R_{i1}, \ldots R_{iM}]$ represents the amount of resource of different types available for workload W_i of application *i*. The goal of the optimization is to find a feasible set of configuration parameters, denoted as C_i, of the application *i* that the performance of the workload P_i is optimized, given the VM's current resource availability R_i. In order to enable such adaption, we need to have a means of mapping different recourse allocations to the corresponding optimal parameter settings. Although this mapping is application specific, there are some general steps.

1) Find out the set of possible parameters $C_i = [c_{i1}, .. c_{ik}, c_{in}]$ that contributes to the application *i*'s performance. For each parameters c_{ik}, we need to determine a function that defines c_{ik} as a function of R_i, i.e., $f_{ik}(R_i)$.
2) Given a certain resource allocation, run a general workload of the virtualized application for the mapping process. Iterate a variety of settings for c_{ik} over its value range and measure the application performance. Collect the setting c_{ik_opt} with the best performance.
3) Repeat Step 2 under different candidate resource allocations over the possible range.
4) Collect the data pairs $<c_{ik\_opt}, R_i>$ for each allocation, and perform regression analysis on the set of the data to fit the function $c_{ik\_opt} = f_{ik}(R_i)$.

Once such a mapping is built for an application, the resource availability to the VM can be directly fed back to enable the application's adaptation.

The aforementioned two aspects of cross-layer optimization are integrated with our existing fuzzy-modeling-based VM resource management middleware. For guest-to-host optimization, the workload is characterized by *Application Sensor* based on application-specific knowledge, which is used by the *Adaptive Learner* for better modeling and predicting the VM's resource usage behavior. For host-to-guest optimization, as *Resource Allocator* adjusts the allocation based on the prediction given by the fuzzy model, it also feeds back this decision to the VM for the application to tune its parameters for better performance. The resulting autonomic resource management system can not only automatically allocate resources to VMs based on their dynamic workload demand but also adaptively improve application performance even when the system is overloaded.

5. CASE STUDY

In this section, we take virtualized databases as an interesting and challenging case study of our proposed cross-layer resource management approach. 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 [11] and stream data management [12], where the workloads are intrinsically dynamic in terms of their bursty arrival patterns and ever-changing unit processing costs. Using VMs to host databases can effectively address this limitation. It allows a database to transparently share the consolidated resources with other applications, where a database's resource usage can elastically grow and shrink based on the dynamic demand of its workload.

The cross-host-guest cooperation for a virtualized database is implemented as follows. For guest-to-host optimization, a database proxy served as the *Application Sensor* is deployed on the host to intercept the incoming query requests to the database VM and characterize the workload composition by classifying the queries. For host-to-guest optimization, a daemon running on the guest periodically obtains resource allocation decision from the *Resource Allocator*, looks up the corresponding optimal database parameters, and sends an administrative query to the database to change the parameters accordingly.

5.1 Guest-to-Host Workload Characterization

Databases are a challenging application 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. Nonetheless, a database's internal query optimizer has intimate knowledge of a query resource usage pattern. Such knowledge can be extracted from the database and used to classify queries for characterizing the entire workload in terms of its resource demands. The result of the workload characterization can be then used as input to the VM's fuzzy model to improve its accuracy and adaptability under dynamic changes of the workload. 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. 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 [13], and second, it does not capture the entire VM's resource needs.

Specifically, we use PostgreSQL database system as an example to demonstrate our proposed guest-to-host optimization on workload characterization. The internal cost model in PostgreSQL is defined as a function of a set of database cost parameters, denoted as $Cost_D(C)$ where $C=[c_1, c_2,.., c_m]$. Each cost parameter represents the unit cost of either CPU or I/O usage associated with an operation in the database. For example, *sequential_page_cost* and *random_page_cost* represent the overhead of a single sequential and non-sequential I/O to fetch a page from disk, respectively; *cpu_tuple_cost* estimates the CPU cost of processing each row in a table. The total cost that aggregates the costs of all operations in a query plan can be broken down into two parts: the total CPU cost and the total I/O cost. Each query can be expressed as a 2-dimention cost vector $<Cost_{CPU}, Cost_{I/O}>$.

To characterize a workload, the *Application Sensor* first extracts the cost vector for all unique queries in a workload from database and then performs subtractive clustering [9] on the set of query cost vectors collected. By setting the radius of a cluster r, any pair of the query vectors with distance $d<r$ will fall into the same cluster indicating queries with similar resource usage patterns. Finally, as the workload runs, the *Application Sensor* measures query intensity online by counting the request rate for each individual cluster. For example, a workload mix W consists of N queries, and after clustering only K clusters are generated where $K<<N$. The workload can be abstracted as a vector of arrival rates of these clusters $<C_1, C_2, \dots, C_K>$. Then the above arrival rate vector that reflects the current characteristics of the workload is periodically fed to the *Adaptive Learner* as an input for modeling the VM's current usage behavior. At the same time, the workload characterization of current time t is also used as the input for the *Resource Predictor* to estimate the resource demand of the next time step t+1 based on the assumption that no abrupt change happens to the workload within one period of time.

5.2 Host-to-Guest Database Adaptation

Databases are a typical application that has a complex internal self-scheduling and self-optimization mechanism which can optimize its performance based on its knowledge about the outside environment. Based on the given resource capacity, a database's query optimizer can automatically evaluate different query execution plans and choose the most efficient one to execute queries. As the availability of resources changes, critical parameters on which the query optimizer depends on for cost evaluation should also be updated accordingly, which will lead to better resource utilization and more efficient query executions. Specifically, a database usually uses the aforementioned cost model $Cost_D(C)$, defined as a function of a set of parameters C, to estimate the costs for all possible query execution plans. Each parameter c_k in the cost model serves as a cost factor related to a certain type of operation in query processing such as table scanning and tuple processing. Appropriate values on these parameters that reflect the actual VM resource availability will help the query planner choose the most efficient operations. Taking PostgreSQL as an example, as shown in Section 2.2, the query optimizer will switch from a sequential scan to an index scan for processing the TPC-H query Q8 as the relative value of *rand_page_cost* to *sequential_page_cost* decreases. Such tuning is necessary when the I/O contention happens and more efficient scanning method is desired given the limited I/O bandwidth.

To tune the cost parameters given changing resource availability, we need to find the mapping from the resource allocation to the optimal parameter values. Because all the cost parameters in a cost model are factors normalized on the same scale, only the changes in their relative values will result in alternative query execution plan. Therefore we focus on building the mapping between the ratio of those cost parameters and the resource allocation to the VM. For example, to investigate the impact of I/O allocation on the scanning methods, the ratio of the aforementioned two I/O cost parameters is considered. We generate a simple query that needs to read all the rows from a large table. The query is executed by different plans using sequential scan vs. random scan iteratively with different amount of I/O allocation. The performance obtained from the changing I/O allocation is observed for each scanning plan. Since the total cost of each plan is mainly resulted from the scanning operations, other types of processing overhead can be ignored. We then normalize the performance of different plans and consider them as the estimation of the I/O cost parameters for different I/O allocation. In this way, a mapping is built between the I/O allocation and I/O cost parameters, which is consistent with the actual performance observed with the corresponding plans.

In addition to those parameters that reflect the knowledge about the database's execution environment, there are also other types of parameters used in database-level scheduling that defines the database's own limit for certain type of resource usage. For instance in PostgreSQL, the parameter *shared_buffers* changes the amount of memory that the database uses for caching data. A reasonable setting value of *shared_buffers* should be proportional to (e.g., ¼) the amount of memory allocated to its VM.

6. EVALUATION

6.1 Setup

This section evaluates our approach using representative database workloads hosted on a typical VM environment. The testbed is a Dell PowerEdge 2970 server equipped with two six-core 2.4GHz AMD Opteron CPUs, 32GB of RAM, and one 500GB 7.2 RPM SAS 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 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 credit CPU scheduler to assign CPU allocations and Linux's dm-ioband I/O controller to set the cap for disk I/O bandwidth [14].

Two typical database benchmarks, TPC-H and RUBiS, are used in our experiments for different purposes. Experiments designed on TPC-H are aimed to show the accuracy of our approach in modeling resource consumption behaviors for highly complex workloads. For RUBiS, it is to show the effectiveness of our solution in adapting to more random changes in the system. The performance metrics is average query response time measured every 10s. Three different resource allocation schemes are compared: 1) The fuzzy-modeling-based allocation with cross-layer optimization which includes guest-to-host workload characterization and host-to-guest database tuning; 2) The fuzzy-modeling-based allocation without cross-layer optimization; 3) The traditional peak-load-based allocation which statically allocates a fixed amount of resources based on the peak workload demand. By comparing the VM's resource usage and the benchmark's performance between these cases, we evaluate whether our proposed cross-layer optimization approach can allocate resources more efficiently while meeting the desired QoS target or improving its performance.

6.2 Guest to Host Optimization

6.2.1 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 10MB/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 6.2.2. The database used here is PostgresSQL 8.1.3 with 2GB of data on a VM with one CPU and 1GB RAM.

To characterize the TPC-H workload, subtractive clustering is performed on 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 queries, 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 5.1 that the database's query cost estimation cannot be used directly to infer the VM's resource needs.

6.2.1.1 CPU-intensive Workload

The first experiment is based on a CPU-intensive workload consisting of Cluster I and II queries, Q1 and Q18. The workload's total request rate is varied from 20 to 50 request/minute while the percentage of Cluster I is also varied from 0% to 80%. About 20 data points with different combinations of request rate and cluster ratio evenly selected from both input ranges are used to train the VM's fuzzy model. With workload characterization (*fuzzy modeling w/ char*), both the request rate and cluster ratio are considered as the input for the CPU usage modeling. In contrast, without workload characterization (*fuzzy modeling w/o char*), only the request rate is used for the input and the ratio factor is ignored. To evaluate these two models, the workload is run with a different set of request rate and cluster ratio combinations (totally 60 data points) while the models are used to control the VM's resource allocation.

Figure 6(a) compares the VM CPU allocations given by these two models against the actual CPU usage of the VM when the resource is allocated based on peak load. Figure 6(b) compares the workload performance under these two CPU allocation schemes against the ideal performance under peak-load-based allocation. The result shows that the CPU allocation given by the fuzzy model created with workload characterization closely follows the VM's actual demand; the average error is below 2.3%. The model created without workload characterization can lead to significant under- or over-provision; the average error is about 36.7%. The difference in CPU allocation accuracy leads to significant difference in the query workload's performance. When using the

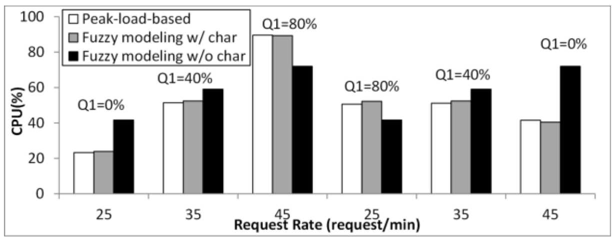

Figure 6(a) CPU allocations for a CPU-intensive TPC-H workload

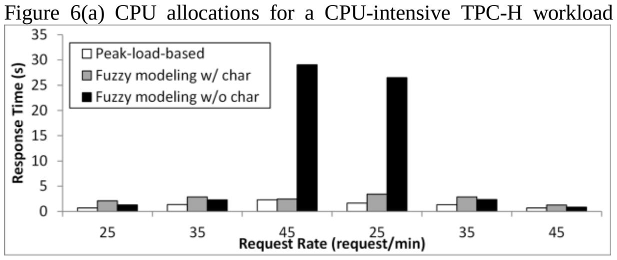

Figure 6(b) Performance for a CPU-intensive TPC-H workload

model created with workload charactrization, the query response time is always at the same level as the peak-load-based allocation; the difference is less than 2s. When using the model created without workload characterization, in some case it leads to up to 27s delay in response time with a 15% under-provision of CPU; in another case, it results in an over-provision of CPU by 15.7% but achieves a response time only 0.6s better than the former scheme.

6.2.1.2 CPU/IO-intensive Workload

In the second experiment, we study a more interesting and challenging workload which includes not only CPU-intensive (Q1 from Cluster I) but also I/O-intensive queries (Q18 from Cluster II and Q6 from Cluster III). As the workload runs, the total percentage of Cluster I+II in the entire workload is varied from 0.1 to 0.9 (the ratio between Cluster I and Cluster II is fixed) and the total request rate also varies from 20 to 80 request/minute. Similarly, different sets of data points are evenly taken from these data ranges for training (450 data points) and testing (120 data points). The experiment is performed separately using fuzzy-modeling-based resource allocation w/ and w/o characterization. The former captures the workload using a vector [*Request rate, Percentage of Cluster I+II*] as the input, while the latter considers only the total request rate of the workload. Both CPU and I/O are controlled in the two cases.

Figure 7(a) and (b) compare the VM CPU and I/O allocations in these two cases against the actual CPU and I/O usages of the VM when the resource is allocated based on peak load. Figure 7(c) compares the workload performance of these two allocation schemes against the ideal performance under peak-load-based allocation. The results show that the fuzzy modeling with workload characterization method can predict the VM's actual demand with an average error of 3.5% for both CPU and I/O allocations. It is more accurate than the case without characterization in which the average error is about 37% for CPU and 73% for I/O. As a result, in the former case it always achieves the same level of performance as the peak-load-based allocation, with only a 1.5s delay in average response time; while in the latter case, the response time is always worse than the peak-load-based case. In the worst case, it produces either a 36% under-provision of CPU which causes a 15s delay or a 27% under-provision of I/O for 11s additional delay. Noticed that the performance in the without characterization case is always worse than the other two cases due to the misprediction of VM resource demand: although

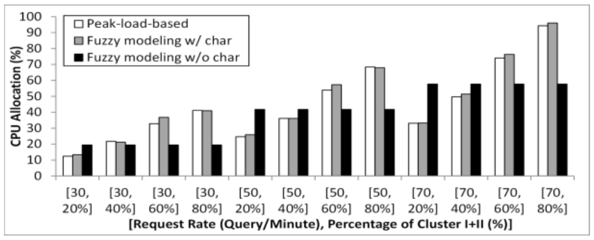

Figure 7(a) CPU allocations for a CPU/IO-intensive TPC-H workload

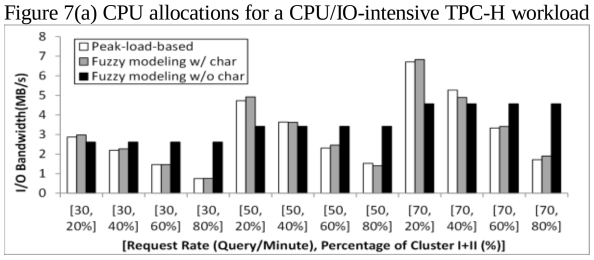

Figure 7(b) I/O allocations for a CPU/IO-intensive TPC-H workload

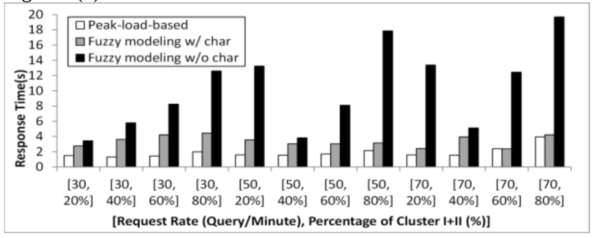

Figure 7(c) Performance for a CPU/IO-intensive TPC-H workload

over-provision of either CPU or I/O does happen, the demands for CPU and I/O cannot be both met at the same time.

6.2.2 RUBiS Experiments

RUBiS models an online auction site that supports the core functionalities such as browsing, selling, and bidding [7]. 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.

Compared to the synthetic workloads used in the above TPC-H experiments, here we constructed more realistic workloads, based on real traces from the 1998 World Cup site [15], one of the public traces widely used in related research for creating realistic workloads [16][17]. In order to emphasize of the variation in workload composition, the workload's intensity is kept constant (the number of concurrent client sessions to the web tier is fixed at 800) while its composition is varied based on the pattern in a typical one-day hourly trace from the World Cup site. We identify the read and write requests in the World Cup trace based on the "Get" and "Post" methods, 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 of its database workload.

We compare the performance of the fuzzy model created with workload characterization versus without it. The former considers both the workload's intensity and composition as the input to the modeling whereas the latter considers only the intensity. The composition can be captured by the ratio of two types of queries, the SELECT queries, which are read-only, and the INSERT and

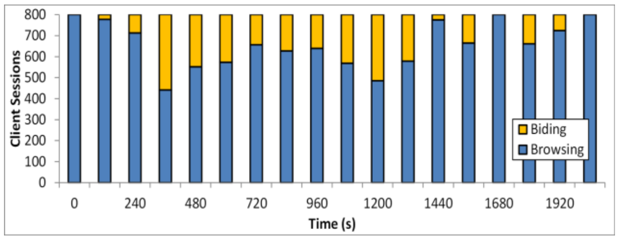

Figure 8(a). Trace for RUBiS with changing composition

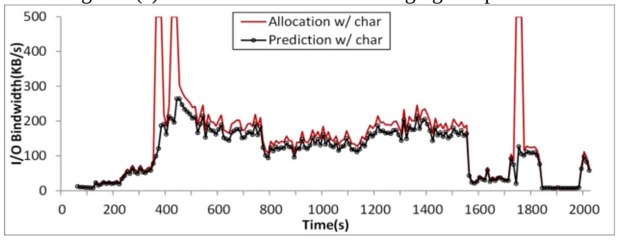

Figure 8(b). I/O allocations with workload characterization

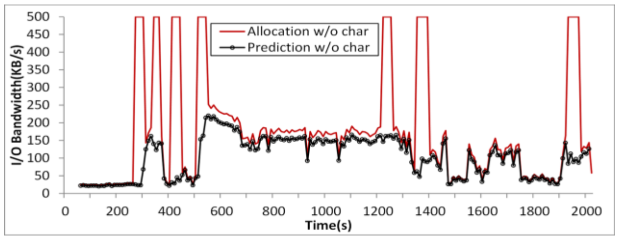

Figure 8(c). I/O allocations without workload characterization

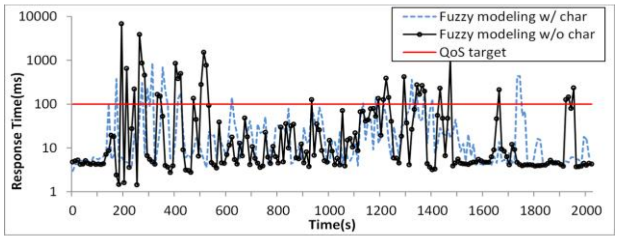

Figure 8(d). Performance comparisons for RUBiS workload

UPDATE queries, which are writes to the database. These characteristics are captured by interposing a MySQL proxy before the database tier. Since this experiment is performed completely online, only the first 10 data points collected are used to initialize the VM's fuzzy model. Afterwards the model is used to allocate resources right away and in the meantime it is updated with new observed data every 10s.

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. In the experiment, the QoS target is set to 100ms for the average response time within each period. A 10% margin is added to the resource allocation predicted by the fuzzy model. When the QoS target cannot be met due to inaccuracy in the model, a backup policy is invoked to allocate a fixed amount of I/O bandwidth (500KB/s) to the VM temporarily. This backup mechanism allows the performance loss to be quickly recovered and ensures that the model can be timely updated to reflect the VM's current resource needs. It is invoked when two consecutive QoS violations occur and revoked after the QoS target are met again for three consecutive periods of time. Afterwards, the fuzzy model updated with the new measurements is used again for guiding the resource allocation.

Figure 8(b) and (c) show the I/O predictions and allocations using a fuzzy model created with/without workload characterization, respectively, for the changing composition RUBiS workload. Figure 8(d) compares the corresponding performance in both cases with the pre-set QoS target. For the fuzzy modeling with workload characterization, it is able to predict the VM's resource needs throughout most of the experiment and require only a few (3 times) invocations of the backup allocation policy. It can quickly react to the changes in workload composition and deliver the desired QoS for 92% of the time; the average response time is 44.9ms throughout the entire experiment. However, without characterization, the QoS target is violated for 15% of the time, and the backup policy is triggered twice more often (7 times). The resulting average response time of 119.5ms cannot meet the QoS target, almost 3 times worse than the one with characterization.

6.3 Combining both Guest-to-Host and Host-to-Guest Optimizations

This experiment demonstrates the effectiveness of our cross-layer resource management by combining guest-to-host workload characterization and host-to-guest database tuning for an OLAP-like database workload.

To construct a more interesting workload, we mix multiple copies of Q1, Q4, Q6, and Q14 from TPC-H queries. To make these queries more diverse in resource usage patterns, distinct query copies are derived from Q4, Q6, and Q14 by modifying the condition in the *where* cause in the original query statements. Each copy touches a different section of the involved tables and the data accessed by different copies is evenly distributed within the range of a table. In this way, the intensity in I/O can be easily varied by changing the total number of these copies, while the CPU intensity is varied by changing the number of copies of original Q1. The experiment is performed in two phases. In *Phase 1*, the workload intensity is fixed by running 18 copies of queries in total but the composition is varied by changing the percentage of Q1's copies from 17% then to 50% and finally to 83%. In *Phase 2*, an I/O cap from 3000 to 1000KB/s is set to the VM to simulate different levels of I/O contention from other VMs while the workload composition is kept constant with 83% of Q1.

Using our proposed approach with cross-layer optimization, during Phase 1, it models the VM's resource demand using the workload characterization result, [*Request rate*, *Percentage of Cluster I*], as the input (two types of queries are classified: Q1 as the CPU-intensive query and the others as the I/O intensive). This model is then used to allocate resources to the VM in the absence of contention. When the experiment transits to Phase 2 and I/O contention is introduced into the system, our approach feeds the I/O bandwidth pressure back to the guest layer by tuning the database parameters according to the resource availability as discussed in Section 5.2. In comparison, we repeat the experiment using fuzzy-modeling-based resource allocation without cross-layer optimization. In this case, during Phase1, only the workload intensity is used to create the fuzzy model; during Phase 2, the database configuration is not adapted and kept static as in Phase 1.

Figure 9 compares the database performance under fuzzy-modeling-based resource allocation with cross-layer optimization (*Cross-layer Opt*) and without it (*Non-Opt*) versus the ideal performance under peak-load-based resource allocation (*Peak-load-based*). From the result we can see that in Phase 1, the

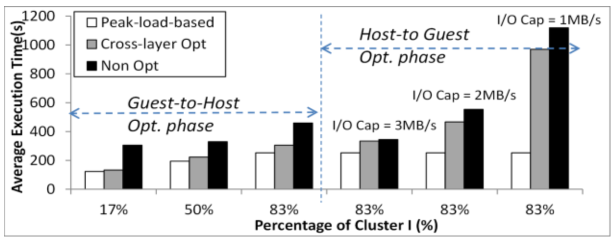

Figure 9 Performance of a TPC-H workload with both guest-to-host and host-to-guest optimizations

performance in *Cross-layer Opt* closely follows the one under peak-load-based allocation. It is as much as seven times better than that in *Non-Opt* which results in a 98% average performance degradation. In Phase 2, both approaches suffer from the reduced I/O bandwidth. However, the *Cross-layer Opt* case still achieves about 17% performance improvement than the *Non-Opt* case. The host-to-guest feedback enables the database query optimizer to switch from a bitmap-scan preferable plan to an index-scan preferable plan for all I/O intensive queries by tuning the I/O cost parameters *sequential_page_cost* vs. *random_page_cost* from the original 1:4 ratio to 1:1 as the I/O cap decreases from 3MB/s to 1MB/s. This adaptation improves the query performance significantly because the index-scan preferable query plan requires less I/O bandwidth than the bitmap-scan preferable one.

7. RELATED WORK

Various solutions have been studied in the literature to address the problem of automatically deciding a VM's resource allocation based on its hosted application's demand and QoS requirement. In particular, different machine learning algorithms have been considered to model VM resource usages. For example, a simple regression method is used to predict the performance impact of VM memory allocation [18]; Reinforcement learning is used to automatically tune VM resource configuration [19]; Artificial neural networks are used to model the nonlinear behaviors for a variety of applications when their VMs are under I/O contention [20]; our previous work [5][8] proposed to use *fuzzy logic* to model the relationship between application workload and VM resource demand, which is shown to be both fast and accurate for modeling systems with complex behaviors. Unlike the traditional solutions which treat VM as a black box, this paper's application-aware approach takes advantage of application-specific knowledge to capture the workload patterns so that the VM model can accurately predict its demand and quickly adapt to the changes in application workload.

Classical feedback control theory has also been used to adjust VM resource allocations according to the application's performance requirement. Linear multi-input-multi-output (MIMO) models are often used for performance modeling, for example, to allocate CPU resource for Web servers [21][22], to manage the performance interference of competing VMs [25], and in more complicated cases, to allocate multiple types of resources to multi-tier applications [23][24]. Although such linear models can be updated adaptively as the system moves from one operating point to another, their accuracy and speed are limited for capturing complex nonlinear behaviors existing in VMs. In contrast, the fuzzy-modeling-based approach proposed in this paper can effectively capture such behaviors and the modeling is shown to be fast. In fact, our previous work as well as others [26] proposed 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 [27].

In the related research on workload-aware resource management, R. Singh *et al.* proposed a mix-aware provisioning solution for allocating server capacity to the bottleneck-tier in datacenters serving workloads with changes in both volume and mix [28]. It employs *k-means* clustering to characterize workload mix and queuing models for predicting server capacity. In comparison, our proposed approach exploits application-specific knowledge to effectively characterize workloads, employs fuzzy modeling to accurately capture VM resource usage, and supports finer-grained allocations of multiple types of resources including both CPU and I/O. Furthermore, in addition to extracting knowledge from the guest to host for better understanding of application resource demand, our approach also delivers the allocation decisions from host to guest for the application to better adapt to resource availability changes.

In the related work on virtualized database resource management, Soror *et al.* proposed using calibrated database query cost model to estimate the VM resource demand [29]. It considers a workload as a static entity consisting of a fixed set of queries, so the resource allocation decision is made for the entire workload according to the overall performance requirement. To deal with the inaccuracy in database cost models, it employs online refinement by assuming a linear VM resource usage model. In contrast, our proposed approach uses database cost model only as a tool to discover workload composition, but not for directly estimating VM resource demand, thereby avoiding the well-known inaccuracy inherent to database cost models. Our approach also more realistically treats a workload as a non-stationary time series and considers fine-grained query performance needs. The VM's complex resource usage model is automatically learned and adapted online without any *a priori* assumption.

Other related autonomous database work [30][31][32] focuses only on a database's internal tuning and query optimization, but not the resource allocation to an entire database VM. Previous work [33][34] points out workload characterization as the key to understanding the resource intensity of a database workload. This paper incorporates both of the two aspects of work in the fuzzy-modeling-based resource management of virtualized databases. It improves the static workload characterization method by allowing online and adaptive characterization and optimizes the performance of virtualized databases by further tuning database parameters according to the adjustment in resource allocations.

8. CONCLUSION AND FUTURE WORK

This paper proposes a new VM resource management approach based on fuzzy modeling and cross-host-guest optimization. It enables the communication between VM host- and guest-layer schedulers and allows them to collaboratively optimize the resource allocation and application performance. The host-layer scheduler exploits guest-layer application-specific information to characterize VM workload and model its resource demand. The guest-layer scheduler uses the host-layer feedback to understand the changing resource availability and adapt its configuration accordingly. As a challenging case study, this cross-layer optimization approach is applied to the resource management of virtualized databases. It uses a database's cost estimation for workload characterization and adapts the database by tuning its cost model parameters according to its resource availability.

A prototype of this approach is implemented on Xen-based VMs and evaluated using typical database workloads based on TPC-H and RUBiS. The results demonstrate that the cross-layer optimization approach significantly outperforms the application-unaware one which treats VMs as black boxes. It can efficiently allocate both CPU and I/O resources to database VMs serving workloads with dynamically changing intensity and composition while meeting QoS targets or improving the performance when under resource pressure.

The proposed cross-layer optimization requires certain awareness between virtualization software and virtualized application. Such awareness breaks the transparency offered by traditional full virtualization, but we advocate that this tradeoff is necessary for business- and mission-critical applications to achieve their desired QoS on virtualized systems. The benefit of this tradeoff is demonstrated by our initial results reported in this paper. The underlying argument is the same as that drives the success of paravirtualization [2] which sacrifices complete transparency for lighter-weight and more efficient virtualization. Although not every virtualized application is capable of adapting its behavior according to changing resource availability, we believe it will become a necessity for critical applications as virtualization becomes pervasive. In our future work, we will study how to create a concise and generic interface for cross-layer optimization that can support diverse guest OSes and applications.

9. ACKNOWLEDGMENTS

This research is sponsored by National Science Foundation under grant CCF-0938045 and Department of Homeland Security grant 2010-ST-062-000039. The authors are also thankful to the anonymous reviewers for their useful comments to improve the quality of the paper.

REFERENCES

[1] VMware, URL: http://www.vmware.com.

[2] P. Barham, Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A, "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] L. Wang, J. Xu, M. Zhao, Y. Tu and J. A.B. Fortes, "Fuzzy Modeling Based Resource Management for Virtualized Database Systems", MASCOTS, 2011.

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

[7] 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.

[8] J. Xu, M. Zhao and J. Fortes, "Autonomic Resource Management in Virtualized Data Centers Using Fuzzy-logic-based Control", Cluster Computing, 2008.

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

[10] J. Liu, R. Rangaswami, and M. Zhao, "Model-Driven Network Emulation With Virtual Time Machine", Winter Simulation Conference, December 2010.

[11] A. Chen, P. Goes, A. Gupta and J. Marsden, "Heuristics for Selecting Robust Database Structures with Dynamic Query Patterns", EJOR, 2006.

[12] M. Wang, T. Madhyastha, N. Chan, S. Papadimitriou and C. Faloutsos, "Data Mining Meets Performance Evaluation: Fast Algorithms for Modeling Bursty Traffic", ICDE, 2002.

[13] S. Chaudhuri, "Relational Query Optimization – Data Management Meets Statistical Estimation", Communications of ACM, 2009.

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

[15] M. Arlitt and T. Jin, "Workload Characterization of the 1998 World Cup Web Site," in HP Technical Report, 1999.

[16] Z. Gong and X. Gu, "PAC: Pattern-driven Application Consolidation for Efficient Cloud Computing", MASCOTS, 2010.

[17] G. Jung, M. Hiltunen, K. Joshi, R. Schlichting and C. Pu, "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures", ICDCS, 2010.

[18] J. Wildstrom, P. Stone and E. Witchel, "CARVE: A Cognitive Agent for Resource Value Estimation", ICAC, 2008.

[19] J. Rao, X. Bu, C. Xu, L. Wang and G. Yin, "VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration", ICAC, 2009.

[20] S. Kundu, R. Rangaswami, K. Dutta and M. Zhao, "Application Performance Modeling in a Virtualized Environment," HPCA, 2010.

[21] X. Liu, X. Zhu, S. Singhal and M. Arlitt, "Adaptive Entitlement Control of Resource Containers on Shared Servers", IM, 2005.

[22] Z. Wang, X. Zhu and S. Singhal, "Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions", DSOM, 2005.

[23] 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.

[24] X. Liu, X. Zhu, P. Padala, Z. Wang and S. Singhal,"Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform", CDC, 2007.

[25] R.Nathuji and A. Kansal, "Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds", Eurosys, 2010.

[26] P. Lama and X. Zhou, "PERFUME: Power and Performance Guarantee with Fuzzy MIMO Control in Virtualized Servers", IWQoS, 2011.

[27] L.Wang, J. Xu, M. Zhao and J. A.B. Fortes, "Adaptive Virtual Resource Management with Fuzzy Model Predictive Control" FeBID, 2011.

[28] R. Singh, U. Sharma, E. Cecchet, and P.J. Shenoy, "Autonomic Mix-Aware Provisioning for Non-Stationary Data Center Workloads", ICAC. 2010

[29] A. Soror, U. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis and S. Kamath, "Automatic Virtual Machine Configuration for Database Workloads", SIGMOD, 2008

[30] G. Weikum, A. Moenkeberg, C. Hasse and P. Zabback, "Self-tuning Database Technology and Information Services: From Wishful Thinking to Viable Engineering", VLDB, 2002.

[31] S. Chaudhuri and G. Weikum, "Foundations of Automated Database Tuning", ICDE, 2006.

[32] B. Schroeder, M. Harchol-Balter, A. Iyengar and E. Nahum, "Achieving Class-based QoS for Transactional Workloads", ICDE, 2006.

[33] P. Martin, S. Elnaffar and T. Wasserman, "Workload Models for Autonomic Database Management Systems", ICAS, 2006.

[34] T. Wasserman, P. Martin and D. Skillicorn, "Developing a Characterization of Business Intelligence Workloads for Sizing New Database Systems", DOLAP, 2004.