Cross-layer Optimization for Virtual Machine Resource Management

Ming Zhao
Arizona State University
Tempe, AZ, USA
mingzhao@asu.edu

Lixi Wang
Amazon.com
Seattle, WA, USA
lwang007fiu@gmail.com

Yun Lv
Beihang University
Beijing, China
lvyuncn@outlook.com

Jing Xu
Google
Moutain View, CA, USA
jingxu97@gmail.com

Abstract—**Virtualized systems (e.g., public and private clouds) are playing an increasingly vital role to support the computing of applications from different domains. Existing resource management solutions in such systems typically treat virtual machines (VMs) as black boxes, which presents a hurdle to achieving application-desired Quality of Service (QoS). 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 by enabling the host-layer scheduler to feedback resource allocation decisions and adapt guest-layer application configurations. As case studies, the proposed approach is applied to virtualized databases and map services which have challenging dynamic and complex resource demands as well as sophisticated configurations. Specifically, for databases, the proposed approach adapts query executions by tuning the cost model parameters according to the available storage bandwidth and memory capacity. For map services, it adapts the quality of returned map imagery in order to meet the response time target as the workload intensity and available network bandwidth change over time. A prototype of the proposed approach is implemented on Xen and Hyper-V VMs, and evaluated using a TPC-H based database workload and a TerraFly-based map service workload. The results show that with the proposed host-to-guest application adaptation, the TPC-H workload improves its performance by 33.5%, and the TerraFly workload improves the map imagery quality by 40% and always meets its response time target, compared to the schemes without adaptation.**

Keywords—cross-layer optimization, virtual machine, resource management

I. Introduction

Virtualized systems are playing an increasingly vital role to support the computing of applications from different domains. For example, cloud computing is a promising platform for delivering computing as a utility to users [1][2]. Public clouds allow public users to rent resources and run a wide variety of applications; private clouds allow users from the same organization to run business-related applications on shared internal resources. In such virtualized systems, applications are consolidated to shared physical resources via their dedicated virtual machines (VMs). As the level of consolidation quickly grows, there is an increasingly urgent need for virtualized systems to deliver better Quality-of-Service (QoS), so that users are comfortable to run their applications on a shared infrastructure. However, current systems cannot meet stringent performance requirements, particularly not for applications with dynamic and complex behaviors. Consequently, examples such as clouds 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 application scheduling inside of a guest, whereas a guest-layer application scheduler is also unaware of the VM's resource allocation on a host. Although such transparency is important for reasons such as portability and legacy support, it also presents a hurdle to achieving application-desired QoS on virtualized systems. On one hand, the knowledge of an application's workload characteristics could be exploited by the host-layer scheduler to better understand the VM's resource demands. On the other hand, the knowledge of the host's resource allocation decisions can help the guest-layer scheduler adapt to the VM's actual resource availability.

Therefore, this paper proposes cross-layer optimization for VM resource management which allows certain awareness and cooperation between the VM host and guest layers in order to improve application performance and meet its QoS target. Specifically, this paper focuses on the problem of enabling the host-layer scheduler to feedback resource allocation decisions to the guest layer and adapt the application configurations. Such cross-layer optimization are integrated into a fuzzy-modeling-based resource management system [3] which uses online fuzzy modeling and prediction to allocate resources dynamically according to application QoS requirements. Compared to related works on paravirtualization [4], this paper's solution does not require any change to the existing VM interfaces and thus supports unmodified applications and operating systems.

This paper considers virtualized databases and web-based map services as representative case studies. Databases serve complex and dynamic workloads consisting of different queries, whereas they also employ sophisticated query optimization which needs to be tuned according to the resource availability. Map services need to serve dynamic map requests with both good responsiveness and imagery quality, which is a tradeoff that should also be adjusted based on the resource availability. Hence, applying cross-layer optimization to the resource management of virtualized databases and map services can be a convincing showcase of the proposed cross-layer optimization. Specifically, in the case study of virtualized databases, the proposed approach adapts query executions by tuning the cost model parameters according to the changing storage bandwidth and memory availability. For virtualized map services, it adapts the imagery resolution of returned maps based on the workload intensity and available network bandwidth in order to meet the response time target.

This proposed approach is prototyped on Xen and Hyper-V based VM environments, and evaluated using both typical database workloads based on TPC-H [5] and typical web map service workloads based on TerraFly [6]. The results show that the proposed approach of host-to-guest application adaptation effectively optimizes the database's query executions when the VM's resource availability changes due to disk I/O and memory contention. The TPC-H workload improves its query time by about 33.5% compared to the scheme without such adaptation. For the TerraFly map service, the proposed approach adapts the quality of returned map imagery according to the changing workload intensity and network contention. It improves the imagery quality by 40% while always meeting the service's response time target. In comparison, the static schemes either deliver a poor image quality and waste the available network bandwidth, or miss the response time target when trying to deliver a high imager quality.

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 to cross-layer optimization and its case studies, Section 6 discusses the evaluation, Section 7 examines the related work, and Section 8 concludes the paper.

II. Motivating Examples

In this section, several examples are used to motivate the need of cross-layer optimization by feeding back the host-layer's resource allocation information to the guest-layer. In the first two examples, the workload consists of a single copy of TPC-H query Q8 on a 3GB database VM. Figures 1 and 2 compare the query performance using two representative settings of the cost model parameters, *sequential_page_cost* and *random_page_cost*, denoted by *seq* and *rand* respectively. 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 defines the cost of a non-sequential disk page fetch. Changing these parameters affects the database's performance indirectly by influencing its internal query cost estimation. Lower value of *seq* reduces the cost of a plan with more sequential scans on the tables; lower value of *rand* reduces the cost of a plan with more random scans, e.g., index scans. Therefore, when the ratio of *seq* vs. *rand* is lower, the database favors execution plans that use more sequential scans; whereas when the ratio is high, the database favors execution plans that use more random scans.

Figure 1 shows the performance of Q8 on a database VM when its memory cache is cold. As the VM's I/O bandwidth allocation reduces from 5000 to 1000 KB/s, the performance of Q8 drops in both database configurations. However, when the available I/O bandwidth is high, the sequential-scan-preferred configuration outperforms the random-scan-preferred one (by 89% at 5000KB/s). When the available bandwidth is reduced,

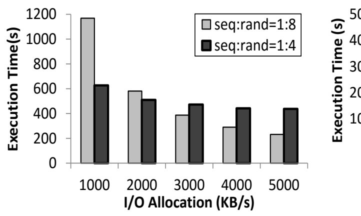

Figure 1: Execution time of TPC-H Q8 with varying I/O bandwidth allocation

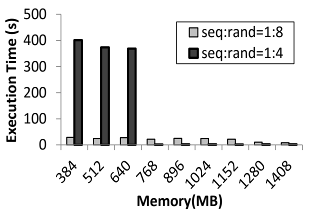

Figure 2: Execution time of Q8 with varying memory allocation

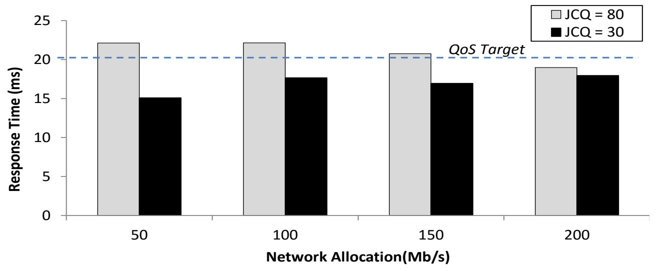

Figure 3: Response time of TerraFly workload with varying network bandwidth allocation

the latter's performance is much less affected and becomes faster than the former (by 1.9 times at 1000 KB/s).

Figure 2 shows similar behavior of Q8's performance but with respect to changing memory availability when performed in a warm database VM. When the available memory is low, the sequential-scan-preferred configuration is drastically faster than the random-scan-preferred one (by 14 times at 384MB), because the query performance is bound by disk I/Os where sequential I/Os are much more efficient than random I/Os. When the available memory becomes large enough to cache the queried data, the random-scan-preferred configuration starts to outperform the sequential-scan-preferred one (by 3 times at 1048MB), because the former touches less data (indexes are much smaller than tables).

The third example is demonstrated using a virtualized web-based map service. On one hand, such a service needs to meet the response time target for map requests; on the other hand, it is also desirable that the returned map imagery resolution to be as high as possible. In Figure 3, two different service configurations are used to process a workload, by changing the JPEG Compression Quality (JCQ) parameter which affects the quality and size of the returned map imagery. When the available network bandwidth is sufficient, both configurations can meet the response time target, but the one with a higher JCQ is more desirable because of its higher image quality. But as the available network bandwidth reduces, the configuration with lower JCQ becomes more suitable because it can lower the response time by transferring less data.

The above examples show strong evidence of the importance of adapting virtualized applications according to their actual resource availability. Cross-layer optimization is key to enabling such adaptation, and the rest of the paper details how it is accomplished with the proposed solution.

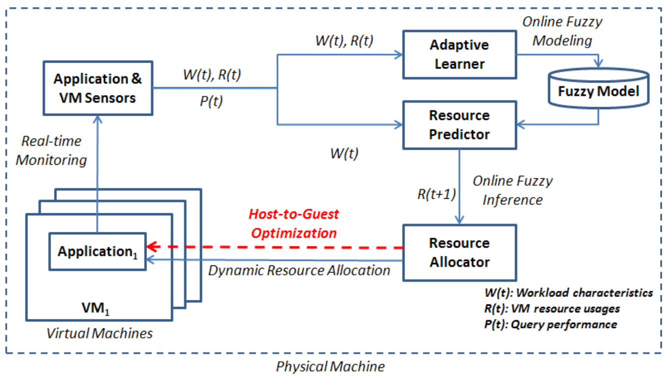

Figure 4: Architecture of cross-layer optimization on fuzzy-modeling-based resource management system

III. FUZZY-MODELING-BASED VM RESOURCE MANAGEMENT

The key questions to VM resource management are how to efficiently allocate resources to VMs and how to do so automatically and continuously. To address these questions, the authors' previous work [3][7] 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 using data observed from the system without assuming any *a priori* knowledge about the system's structure. It is shown to be able to capture complex, nonlinear resource usage behaviors of a virtualized system.

Figure 4 illustrates the architecture of the 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 usages $R(t)$. 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 that is executed iteratively online for VM resource management.

Fuzzy logic is employed to build the model based on the qualified input-output data pairs, $<W(t), R(t)>$ whose workload performance $P(t)$ meets 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 demands 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 the authors' previous work [3][7].

Note that while this paper's work is built upon the above mentioned fuzzy-modeling-based resource management system, it does not rely on the use of fuzzy modeling. In fact, any effective online modeling methods can be employed by the *Adaptive Learner*; fuzzy modeling is preferred because of its fast speed and ability to capture VMs' complex resource usage behaviors. More importantly, this paper's work significantly improves the existing resource management system by enabling cross-layer optimization between the VM host and guest layers.

IV. GENERAL APPROACH TO CROSS-LAYER OPTIMIZATION

The goal of cross-layer optimization is to enable VM host- and guest-layer schedulers to communicate scheduling-related information and collaboratively improve the performance of a virtualized application and satisfy its QoS requirement. Existing resource management solutions do not support such cross-layer optimization, because they treat VMs as black boxes. This paper proposes to trade off the transparency of virtualization for certain awareness and cooperation between host and guest in order to optimize the VM scheduling and application performance. The cross-layer optimization considered in this paper focuses on enabling the guest-layer scheduler to adapt its application-specific configuration based on the host-layer VM resource allocation to improve the application performance. This section describes the general approach to such cross-layer optimization.

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 can also change its crawling, indexing, or searching strategies as the resource availability varies. When resource is constrained, it may crawl 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 [8].

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: first, changing resource contention from other co-hosted VMs as they come and go dynamically and their workloads vary over time; second, changing resource allocation policy such as VM priorities or 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 resource utilization as the VM's resources become either under pressure or abundant.

To address this problem, the cross-layer optimization enables 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. Specific implementations of this approach will be explained using case studies in the next section.

Assume that there are M different types of resources, such as CPU cycles, memory capacity, and I/O bandwidth. $R_i = [R_{i1}, \cdots, R_{iM}]$ represents the amount of resources of different types available for application i's workload W_i. The goal of the optimization is to find a feasible set of configuration parameters, denoted as C_i of the application i, with which the performance of workload P_i is optimized, given the VM's current resource availability R_i. For applications that have a large number of parameters, commonly used techniques such as Principal Component Analysis (PCA) and Independent Component Analysis (ICA) [9] can be employed to identify a smaller set of parameters that have strong correlations with the application performance [10] and consider only them for application adaptation in the cross-layer optimization.

In order to enable such adaption, a mapping needs to be built between different resource 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 $C_i = [c_{i1}, \cdots, c_{ik}, \cdots, c_{in}]$ that contribute to the application i's performance. For each parameter c_{ik}, a mapping needs to be determined, which defines the optimal c_{ik_opt} as a function, $f_{ik}(R_i)$, of the resource availability 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's 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 $\langle c_{ik\_opt}, R_i \rangle$ for each allocation, and perform regression analysis on the set of 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 by changing its configuration parameters accordingly.

The aforementioned cross-layer optimization is integrated with the fuzzy-modeling-based VM resource management introduced in Section 3 (Figure 4). As *Resource Allocator* adjusts the allocation based on the prediction given by the fuzzy model, it also feeds back this decision to the guest for the application to tune its parameters for better performance. Specifically, this adaptation can be implemented using a daemon running on the guest which periodically obtains resource allocation decisions from the *Resource Allocator*, computes the optimal parameter settings, and adjusts the parameters through the application's configuration interface.

The resulting autonomic resource management system is able to not only automatically allocate resources to VMs based on their dynamic workload demands but also adaptively optimize the application configuration as the resource availability changes over time. The stability of the system is ensured by two factors: *1)* guest-layer application adaptation occurs at a much coarser time granularity (e.g., every minute) than host-layer resource adjustment (e.g., every 10 seconds); *2)* the host layer is able to quickly update its fuzzy model to capture a VM's new behaviors and continue to accurately predict its demands when the guest-layer application adapts.

The next section presents two concrete case studies using two different and representative applications, databases and web-based map services, to demonstrate this cross-layer optimization approach.

V. Case Studies

A. Virtualized Databases

Databases represent a typical type of applications that have sophisticated internal mechanisms to optimize their performance based on their knowledge about the hosting environments. Based on the host's resource capacity, a database's query optimizer can automatically evaluate the costs of 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 often uses an internal cost model *CostD*(C), defined as a function of a set of parameters C, to estimate the costs for 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 resource availability will help the query planner choose the most efficient operations. Taking PostgreSQL as an example, as shown in Section 2, the query optimizer switches from using sequential scans to random scans for processing the TPC-H query Q8 as the ratio between *seq* and *rand* increases. Such tuning is necessary when, e.g., disk 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, a mapping needs to be created 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 result in alternative query execution plan. Therefore, the mapping needs to be built only between the optimal ratio of the cost parameters and the resource allocation to the VM.

For example, to investigate the impact of I/O bandwidth allocation on scanning methods, the ratio of the aforementioned two I/O cost parameters is considered. A simple query is used to benchmark this ratio, which reads all the rows from a large table. The query is executed by different plans (sequential scan vs. random scan) with different amount of I/O allocations. The performance is observed for each scanning plan under different I/O allocations. Since the cost of executing this simple query is

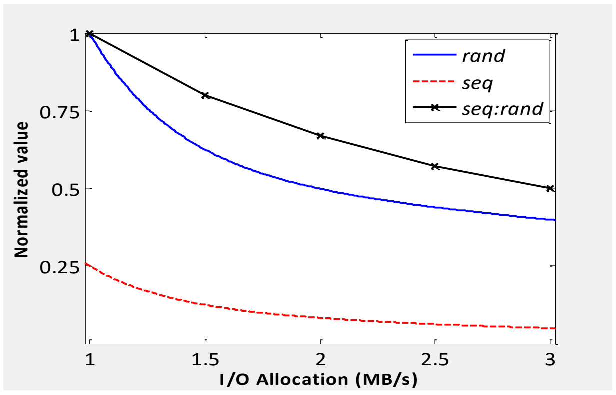

Figure 5(a): The mapping between database cost parameters and VM I/O bandwidth allocation

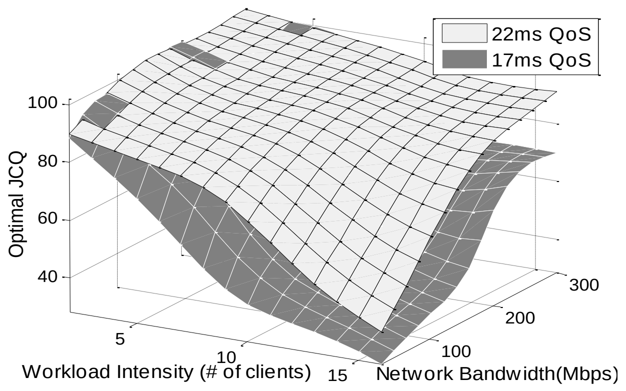

Figure 5(b): The mapping between map service JCQ and workload intensity and VM network bandwidth allocation

mainly from the scanning operations, the performance of different plans (sequential scan vs. random scan) can be considered as the estimate of the I/O cost parameters (*seq* vs. *rand*) for different I/O allocations. In this way, a mapping is built between the I/O allocation and the I/O cost parameters (Figure 5(a)). When the VM's I/O allocation changes, the ratio between these two parameters can be then adapted accordingly so that the database can choose the most efficient query execution plan under the given resource allocation.

In addition to parameters that reflect the knowledge about the database's execution environment, there are also other types of parameters that define the database's own limit for certain types of resource usage. Such parameters should also be adapted according to the database VM's actual resource availability. For instance in PostgreSQL, the parameter *shared_buffers* changes the amount of memory that the database uses for caching data. A reasonable setting of *shared_buffers* should be proportional to (e.g., ¼) the amount of memory allocated to its VM.

B. *Virtualized Map Services*

Another interesting case study of this paper's cross-layer optimization is web-based map services. Map services are the most important applications of modern geographic information systems, which serve requests for maps and related geographic information for a variety of clients over Internet. Map services represent applications that can tune their QoS based on the resource availability (other examples include search engines and streaming services). The configurations that need to be tuned on a map service include the resolution and comprehensiveness of the returned maps and the selection of different search strategies for geographic information. The settings of these configurations affect different aspects of a map service's QoS and need to be carefully tuned according to its host's resource capacity. Hence, automatic adaptation becomes important for a virtualized web map service when its resource availability changes dynamically.

Specifically, this paper focuses on one key tunable parameter in a map service, the JPEG compression quality (JCQ), which affects two different aspects of the QoS—response time and imagery quality. JCQ determines the compression level of a map image returned to a request. Setting a higher JCQ value results in returning maps with a better resolution which also require more data transfer. This case study assumes a typical service-level objective which is to meet the response time target while delivering maps with the highest possible resolution. As illustrated in Section 2, this objective cannot be met using a fixed JCQ setting in a virtualized web map system where the available network bandwidth varies over time. It is necessary to adapt the JCQ setting automatically based on the VM's network bandwidth availability.

In order to use the host-to-guest map service adaptation for JCQ tuning, a mapping needs to be created from the network bandwidth allocation to the optimal JCQ value. The optimal JCQ depends on the workload intensity, the available network bandwidth, and the response time target. To build the mapping, the map service's performance is profiled using a synthetic workload under different JCQ settings while the workload's intensity and the VM's network bandwidth allocation are varied. Based on these collected performance data, the optimal JCQ can be then found by searching for the highest JCQ value with which the corresponding performance satisfies the given response time target.

In this way, the mapping is built from the network bandwidth availability and workload intensity to the optimal JCQ for the given response time target. The profiling time can be reduced by collecting only a subset of the data and using regression to build the rest of the profile. Figure 5(b) illustrates two of such mappings for the response time targets of 22ms and 17ms. A total of 144 data points are collected to build a mapping in this figure and the fitting error is 2.95% on average. With these mappings, the JCQ value can be then adjusted automatically and always set to optimal as the network bandwidth availability or the workload intensity changes.

VI. Evaluation

A. *Setup*

This paper's cross-layer optimization approach is evaluated using both databases and web map services discussed in the above case studies. The testbed is a physical machine equipped with two six-core 2.4GHz AMD Opteron CPUs, 32GB of RAM, and one 500GB 7.2 RPM SAS disk.

To evaluate the virtualized database, 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

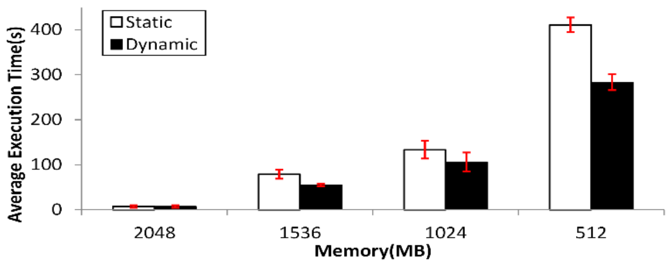

Figure 6(a): Performance of a TPC-H workload with 50 request/s

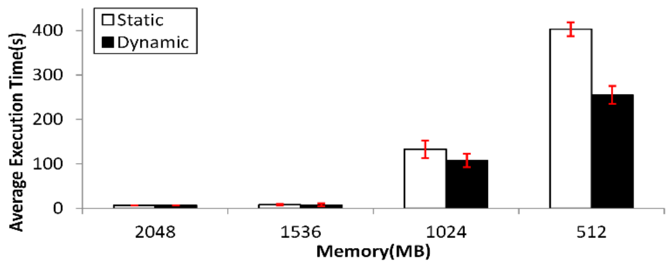

Figure 6(b): Performance of a TPC-H workload with 30 request/s

kernel 2.6.18.8. The evaluated databases are hosted on DomUs, while the resource management system is hosted on Dom0. 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 [11]. A typical database benchmark, TPC-H [5], is used in these experiments.

To evaluate the virtualized map service, Microsoft Hyper-V 6.2 [12] is deployed to provide the virtualization environment. The operating systems on parent and child partitions are Windows Server 2012 and Windows Server 2008 R2 Datacenter respectively. The map service application is hosted on the child partition configured with 1 CPU core and 4GB memory. The resource management system deployed on the parent partition monitors and controls the network I/O bandwidth to the child partition through the Hyper-V's bandwidth management tool. The specific map service considered here is TerraFly [6], a production web-based map system serving requests from over 125 countries and regions and providing users with customized aerial photography, satellite imagery, and various overlays. The real workload traces collected from production TerraFly system are used in the evaluation.

B. *TPC-H*

This experiment demonstrates the effectiveness of the host-to-guest optimization by automatically tuning a database system under varying memory availability. An I/O intensive workload consisting of a mix of duplicated copies of the Q4, Q6, Q8 and Q14 queries from TPC-H is run on a database with warm memory, where the query processing can be done mostly using

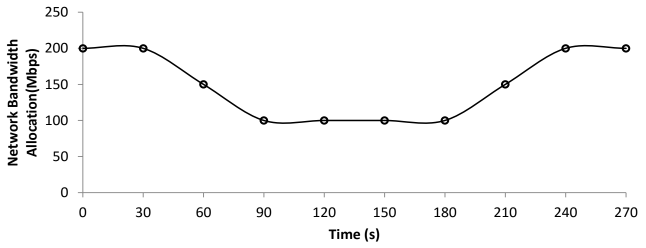

Figure 7(a): Network bandwidth allocation to TerraFly VM

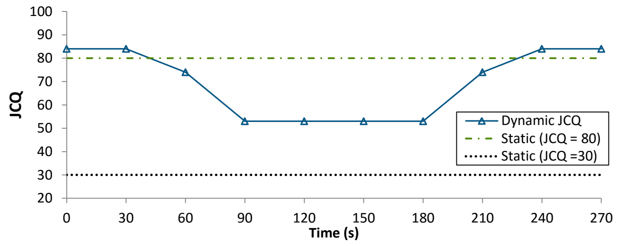

Figure 7(b): TerraFly's JCQ settings

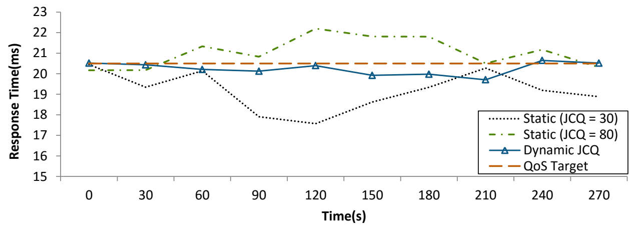

Figure 7(c): TerraFly's performance with different JCQ settings

data cached in memory. The intensity of the workload can be varied by changing the inter-arrival rate of the queries from 4.8s to 8s with a corresponding request rate of 50 and 30 queries per minute. To simulate different levels of memory contention, the database VM's memory allocation is varied from 2048MB, 1536MB, 1024MB to 512MB while the workload is running at a given request rate.

Figure 6 compares the performance of two TPC-H workloads with different intensities from the scheme that uses host-to-guest optimization (*Dynamic*) vs. without it (*Static*). The former dynamically adapts the ratio between *seq* and *rand* as the memory availability changes; the latter uses a static ratio of 1:4. The result shows that the adaptation improves the database performance for both workloads as the available memory reduces. For example, an average of 33.5% improvement in query execution time is achieved when the VM's memory is 512MB. The improvement increases as the workload becomes more intensive because the memory contention gets worse. For the workload with 50 request/s, as soon as the memory allocation is reduced to 1.5GB, substantial speedup is observed; while for the workload with 30 request/s, the advantage of optimization becomes evident only when the available memory is reduced to 1GB and less.

The host-to-guest optimization achieves the above performance improvement because it enables the database to adapt its query execution strategy as the memory availability varies. Specifically, it allows the database to switch from a random-scan-preferred configuration to a sequential-scan-preferred one by tuning its ratio of *seq* vs. *rand* from the default

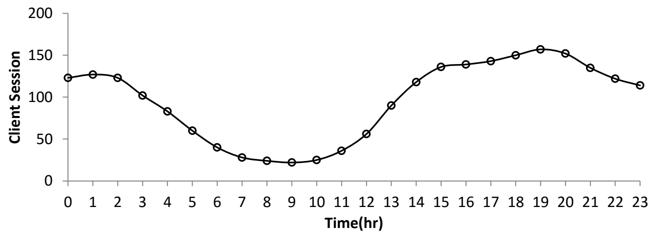

Figure 8(a): A real TerraFly workload with changing intensity

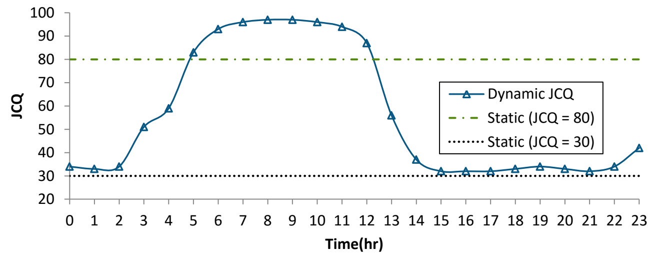

Figure 8(b): TerraFly's JCQ settings

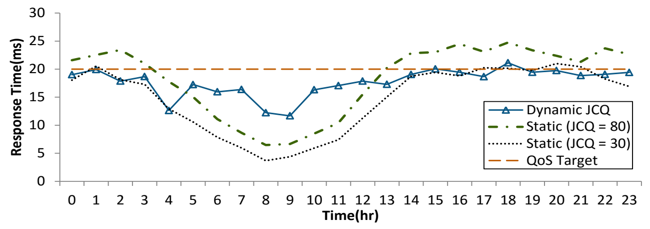

Figure 8(c): TerraFly's performance with different JCQ settings

1:4 ratio to 1:16 as the available memory decreases from 2GB to 512MB. When the memory is sufficient to cache all the queried data, a random-scan-preferred configuration is advantageous because it scans indexes and accesses less data. When the memory is not sufficient to cache the queried data, the query processing becomes disk I/O bound where sequential scans are more efficient. The cross-layer optimization takes advantage of this application-specific knowledge to make sure that the database's performance is always optimal given its VM's current memory availability.

C. *TerraFly*

To demonstrate the effectiveness of the host-to-guest adaptation for TerraFly-based map service, two scenarios are considered in this experiment. In the first scenario, the amount of available network bandwidth to TerraFly is contended by another VM which runs an FTP server. Figure 7(a) shows that the network bandwidth allocated to the TerraFly VM is first reduced from 200 to 100 Mbps as a file transfer starts on the FTP VM, sustained at 100 Mbps during the transfer, and finally increased back to 200 Mbps when the transfer completes. With the host-to-guest adaptation, the network resource availability is explicitly fed back to the TerraFly VM and used to adapt the JCQ for the map service.

Figure 7(c) compares the performance of TerraFly using three different JCQ settings shown in Figure 7(b): one with a dynamic JCQ adapted by host-to-guest optimization (*Dynamic*) versus two using static JCQ settings (*Static*). The results show that the host-to-guest adaptation allows the response time target (20.5ms) to be met throughout the experiment. In contrast, using a static high JCQ misses the response time target most of the time and causes up to 15% delay in response time. Although using a static low JCQ can meet the response time target, it fails to provide a good image quality to map requests and wastes the available network bandwidth when it is sufficient. Compared to it, the host-to-guest adaptation is able to fully utilize the available network resources and improve image quality by 40% in average.

In the second scenario, a fixed amount of network bandwidth (50 Mbps) is allocated to the TerraFly VM while a real workload collected from the production TerraFly system (shown in Figure 8(a)) is replayed with a 60-fold speedup. Although the network contention does not change in this experiment, the host-to-guest adaptation still enables TerraFly to adapt its JCQ based on the knowledge of its network bandwidth availability and workload intensity.

Figure 8(c) compares the performance of TerraFly using the three different JCQ settings shown in Figure 8(b). Similar to the previous experiment, the result shows that the dynamic JCQ setting adapted by host-to-guest optimization outperforms the static JCQ settings in terms of imagery quality and response time of the map requests. Using a high JCQ statically is not able to meet the response time target when the workload intensity becomes high; the scheme with a static low JCQ cannot provide good quality images even when there is abundant network bandwidth to be used. In contrast, the host-to-guest JCQ adaptation approach always meets the response time target and delivers an average improvement of 26.3% in imagery quality.

VII. 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, 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 [13]; Reinforcement learning is used to automatically tune VM resource configuration [14]; Artificial neural networks are used to model the nonlinear behaviors for a variety of applications when their VMs are under I/O contention [15]; The authors' previous work [3][7] studied the use of 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, time-varying behaviors.

In addition, predictive resource controllers have been used to automatically adjust VM resource allocations based on the VM performance models in order to meet their applications' QoS targets. For example, linear multi-input-multi-output (MIMO) models have been used by predictive controllers to allocate CPU resources to multiple virtualized web servers [16], and in more complicated cases, to allocate multiple types of resources to virtualized multi-tier applications [17]. Fuzzy modeling based predictive control has also been proposed to better capture the nonlinear behaviors in VMs' resource contention and meet applications' performance needs [18].

This paper builds upon the fuzzy-modeling-based resource management framework and benefits from its fast speed and ability to capture nonlinear behaviors. More importantly, This paper's cross-layer optimization approach complements these solutions in that: fuzzy modeling and the other effective modeling methods can be employed to estimate VMs' resource demands and make resource requests, whereas this paper's approach allows the virtualized applications to adapt their configurations and optimize their performance when the available resources cannot satisfy their requests.

There are related paravirtualization works that also adopt a cross-layer approach to bridge the semantics gap between VM guests and host and improve various aspects of virtualized systems, including reducing virtualization overhead [4], increasing memory utilization [19], and improving intra-host VM communications [20]. However, these works all require changes to the virtualization interfaces and the guest operating systems in VMs. In contrast, this paper's approach does not require any change to the guest systems: it keeps unmodified virtualization interfaces for running VMs, and leverages applications' existing mechanisms to adapt their configurations according to the current resource availability.

Related autonomous database works [21][22][23] are focused mainly on a database's internal tuning and query optimization, which however do not work when the database is virtualized and unware of its actual resource availability. This paper's cross-layer optimization approach will bridge this gap by making the database aware of its resource availability and able to tune itself properly in a systematic manner.

Finally, compared to the authors' previous work on application-aware VM resource management [24], this paper makes substantial new contributions on host-to-guest application adaptation by considering different types of VM resources (memory, disk bandwidth, and network bandwidth) and different types of virtualized applications (databases and map services).

VIII. Conclusions 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. Specifically, the guest-layer scheduler uses the host-layer feedback to understand the changing resource availability and adapt its configuration accordingly. Virtualized databases and map services are considered as interesting case studies of this cross-layer optimization approach. A database's cost model parameters are adapted according to its VM's resource availability. A map service's imagery quality is also adapted according to its workload intensity and resource availability in order to sustain the response time target.

A prototype of this approach is implemented on Xen- and Hyper-v based VMs and evaluated using TPC-H based database workloads and TerraFly-based map service workloads. The results demonstrate that the proposed cross-layer optimization significantly improves the performance and QoS of virtualized applications compared to the traditional approaches which treat VMs as black boxes.

This cross-layer optimization approach requires certain awareness between the virtualization software and virtualized applications. Such awareness breaks the transparency offered by traditional full virtualization, but this paper advocates that such a 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 the experimental results reported in this paper. The underlying argument is the same as that drives the success of paravirtualization [4] which sacrifices complete transparency for lighter-weight and more efficient virtualization. Although not every virtualized application is capable of adapting its behaviors, the authors believe that it will become a necessity for critical applications as virtualization becomes pervasive.

Based on this cross-layer optimization framework, future works will be conducted along two possible directions. First, VM migrations will be considered in addition to application adaptations to handle situations where resources on a host are not sufficient to satisfy all the VMs' requests. These two techniques will complement each other: VM migration can harness idle resources on other hosts to satisfy the VMs' resource needs; application adaptation is necessary when there is no suitable host for migration or when migration is not supported by the hosts. When migrations are used, the authors' previous work [25] can be employed to migrate a VM's performance model together with the VM so that it does not have to be learned from scratch after the migration. Second, cross-layer optimization will also be applied to applications that are distributed across multiple VMs, including both parallel applications that make use of several VMs to speed up the execution, and multi-tiered applications that distribute their tiers across multiple VMs. For such distributed applications, adaptations may have to be applied across the involved VMs in a coordinated manner, which is an interesting topic for future research.

Acknowledgement

The authors thank the anonymous reviewers for their helpful comments. This research is sponsored by National Science Foundation under the National Science Foundation CAREER award CNS-1619653, CNS-1629888, IIS-1633381, and CMMI-1610282.

References

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

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

[3] L. Wang, J. Xu, M. Zhao, Y. Tu, and J. A.B. Fortes, "Fuzzy Modeling Based Resource Management for Virtualized Database Systems", in Proceedings of International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, 2011.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization", ACM SIGOPS Operating Systems Review, vol. 37, no. 5, pp. 164-177, ACM, 2003.

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

[6] N. Rishe, S. Chen, N. Prabakar, and M. Weiss, "TerraFly: A High-performance Web-based Digital Library System for Spatial Data Access", in Proceedings of International Conference on Data Engineering, 2001.

[7] J. Xu, M. Zhao, and J. Fortes, “Autonomic Resource Management in Virtualized Data Centers Using Fuzzy-logic-based Control”, Cluster Computing, 11(3), pp.213-227, 2008.

[8] J. Liu, R. Rangaswami, and M. Zhao, “Model-Driven Network Emulation with Virtual Time Machine”, in Proceedings of Winter Simulation Conference, December 2010.

[9] J. Stone, “Independent Component Analysis: An Introduction”, Trends in Cognitive Sciences, vol. 6, no. 2, pp. 59–64, 2002.

[10] N. Park, W. Xiao, K. Choi, and D. Lilja, “A Statistical Evaluation of the Impact of Parameter Selection on Storage System Benchmarks”, in Proceedings of International Workshop on Storage Network Architecture and Parallel I/Os, 2011.

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

[12] Hyper-V, URL: http://msdn.microsoft.com/en-us/library/cc768520%28v=bts.10%29.aspx

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

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

[15] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao, “Application Performance Modeling in a Virtualized Environment”, in Proceedings of International Symposium on High-Performance Computer Architecture, 2010.

[16] X. Liu, X. Zhu, S. Singhal, and M. Arlitt, “Adaptive Entitlement Control of Resource Containers on Shared Servers”, in Proceedings of International Symposium on Integrated Network Management, 2005.

[17] P. Padala, K. Hou, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, “Automated Control of Multiple Virtualized Resources”, in Proceedings of European Conference on Computer Systems, 2009.

[18] L. Wang, J. Xu, M. Zhao, and J. A.B. Fortes, “Adaptive Virtual Resource Management with Fuzzy Model Predictive Control”, in Proceedings of International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks, 2011.

[19] C. A. Waldspurger, “Memory Resource Management in VMware ESX Server”, ACM SIGOPS Operating Systems Review, 36(SI), 181-194.

[20] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam. “Xen and Co.: Communication-aware CPU Scheduling for Consolidated Xen-based Hosting Platforms”, in Proceedings of the 3rd International Conference on Virtual Execution Environments, pp. 126-136, 2007.

[21] G. Weikum, A. Moenkeberg, C. Hasse, and P. Zabback, “Self-tuning Database Technology and Information Services: From Wishful Thinking to Viable Engineering”, in Proceedings of International Conference on Very Large Databases, 2002.

[22] S. Chaudhuri, “Relational Query Optimization – Data Management Meets Statistical Estimation”, Communications of ACM, 2009.

[23] B. Schroeder, M. Harchol-Balter, A. Iyengar, and E. Nahum, “Achieving Class-based QoS for Transactional Workloads”, in Proceedings of International Conference on Data Engineering, 2006.

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

[25] L. Wang, J. Xu, and M. Zhao, “QoS-driven Cloud Resource Management through Fuzzy Model Predictive Control”, in Proceedings of Proceedings of the 12th International Conference on Autonomic Computing (ICAC), July 2015.