3-6-2015

Fuzzy Modeling and Control Based Virtual Machine Resource Management

Lixi Wang
lwang007@fiu.edu

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

FUZZY MODELING AND CONTROL BASED VIRTUAL MACHINE RESOURCE MANAGEMENT

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Lixi Wang

2015

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Lixi Wang, and entitled Fuzzy Modeling and Control Based Virtual Machine Resource Management, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

_____
Jason Liu

_____
Raju Rangaswami

_____
Gang Quan

_____
Ming Zhao, Major Professor

Date of Defense: March 6, 2015

The dissertation of Lixi Wang is approved.

_____
Dean Amir Mirmiran
College of Engineering and Computing

_____
Dean Lakshmi N. Reddi
University Graduate School

Florida International University, 2015

DEDICATION

To my beloved parents.

ACKNOWLEDGMENTS

Foremost, I would like to express my deepest gratitude to my advisor, Professor Ming Zhao, who accepted me as his first Ph.D. student. His enthusiasm and persistence to doing great research in computer system has set me an excellent example during my pursuit of Ph.D. Without his guidance and support, I would never have been able to finish my dissertation. I would also like to thank his wife, Dr. Jing Xu for helping me a great deal in developing my background in control theory and applying fuzzy logic to computer systems.

I want to thank my committee members, Professor Jason Liu, Professor Raju Rangaswami and Professor Gang Quan for reviewing my proposal and dissertation and offering helpful comments to improve my work.

I would also thank Yun Lv who helped to collect important experimental data during our collaboration which was a joyful experience to me. Many gratitude to my lab-mates, Dulcardo Arteaga and Yiqi Xu who assisted me in setting up experimental environment, Douglas Otstott and Rachel Chavez who helped to enable my remote attendance to research seminars during my absence. I want to thank Hsinyu Ha and Yimin Yang who as my good friends, were always willing to help. A special thank goes to my friend Ting Li; it would have been quite a lonelier and longer journey without her.

Finally, I would like to thank my husband and my parents, who provides me with enormous support and encouragement during my entire Ph.D. life.

ABSTRACT OF THE DISSERTATION

FUZZY MODELING AND CONTROL BASED VIRTUAL MACHINE RESOURCE MANAGEMENT

by

Lixi Wang

Florida International University, 2015

Miami, Florida

Professor Ming Zhao, Major Professor

Virtual machines (VMs) are powerful platforms for building agile datacenters and emerging cloud systems. However, resource management for a VM-based system is still a challenging task. First, the complexity of application workloads as well as the interference among competing workloads makes it difficult to understand their VMs' resource demands for meeting their Quality of Service (QoS) targets; Second, the dynamics in the applications and system makes it also difficult to maintain the desired QoS target while the environment changes; Third, the transparency of virtualization presents a hurdle for guest-layer application and host-layer VM scheduler to cooperate and improve application QoS and system efficiency.

This dissertation proposes to address the above challenges through fuzzy modeling and control theory based VM resource management. First, a fuzzy-logic-based nonlinear modeling approach is proposed to accurately capture a VM's complex demands of multiple types of resources automatically online based on the observed workload and resource usages. Second, to enable fast adaption for resource management, the fuzzy modeling approach is integrated with a predictive-control-based controller to form a new Fuzzy

Modeling Predictive Control (FMPC) approach which can quickly track the applications' QoS targets and optimize the resource allocations under dynamic changes in the system. Finally, to address the limitations of black-box-based resource management solutions, a cross-layer optimization approach is proposed to enable cooperation between a VM's host and guest layers and further improve the application QoS and resource usage efficiency.

The above proposed approaches are prototyped and evaluated on a Xen-based virtualized system and evaluated with representative benchmarks including TPC-H, RUBiS, and TerraFly. The results demonstrate that the fuzzy-modeling-based approach improves the accuracy in resource prediction by up to 31.4% compared to conventional regression approaches. The FMPC approach substantially outperforms the traditional linear-model-based predictive control approach in meeting application QoS targets for an oversubscribed system. It is able to manage dynamic VM resource allocations and migrations for over 100 concurrent VMs across multiple hosts with good efficiency. Finally, the cross-layer optimization approach further improves the performance of a virtualized application by up to 40% when the resources are contended by dynamic workloads.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE PAGE

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. Served as powerful platforms for hosting systems, VMs allow applications to be encapsulated along with their execution environments and easily deployed on different systems. 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, particularly 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.

Autonomic resource management promises to address these problems for such a VM based hosting system. The goal of such a system is two-fold. First, it should be able to automatically allocate resources to a VM according to the hosting application's demand for satisfying desired QoS. Second, it should be able to automatically adapt to dynamic changes

in the VM's behavior and timely adjust the resource allocation, so that both resource efficiency and QoS can be sustained. However, the complexity and dynamism in the virtualized applications and system pose several key challenges for the VM based resource management system, which makes it challenging to host application on shared resources without compromising the QoS of applications or wasting the resources of the system.

- First, the complexity of application workloads which often consist of a variety of requests with distinct resource usage may lead to not only different levels of but also multiple types of virtualized resource demands. The interference between multiple consolidated application workloads which compete for resources that cannot be strictly portioned may also lead to complex nonlinear resource usage behaviors,.
- Second, the dynamics in both the applications (e.g., changes in an application workload or variation in its QoS target) and the system (e.g., changes in service-level objectives) require timely control actions in response to the environment changes. The control actions should consider both the performance tracking accuracy and the system stability, in order to not only maintain the desired QoS target for individual applications but also sustain an optimized overall performance for system-level objectives.
- Third, the transparency of virtualization presents a hurdle for guest-layer application and host-layer VM scheduler to cooperate and improve application QoS and system efficiency. Without any knowledge about the guest-layer application, it is difficult for the host-layer scheduler to understand the application's workload composition and detect the intrinsic workload changes; without any knowledge of the host-layer scheduler's resource allocation decisions, it is also difficult for the guest-layer

application to adapt its application-specific configuration and improve its performance as the resource availability changes.

1.1. Fuzzy Modeling Based VM Resource Management

In the first resource management approach, fuzzy modeling method is proposed to learn and predict a VM's demands of multiple types of resources based on the observed workload intensity and resource usages. This method does not require any a priori knowledge of the system's internal structure and it can efficiently describe complex and nonlinear system behaviors through a VM's fuzzy model which can be learned and updated online. A prototype of this fuzzy modeling based resource management approach is built on Xen-based VM environment for a database hosting system. Databases often serve complex and dynamic workloads which consist of a variety of queries with different types and amounts of resource demands. Therefore, virtualized databases can be an excellent case study of the proposed approach.

The main contribution of this approach lies in two aspects: first, it can accurately and efficiently allocate multiple typs of resources, i.e., both CPU and disk I/O bandwidth, for a database VM that is serving CPU and I/O intensive queries while delivering the same level of QoS as using peak-load-based resource allocation; second, it can perform the resource adjustments online at fine granularity (every 10s) and adapt to dynamic changes in the workloads served by the virtualized database. To the best of our knowledge, this is the first to study fuzzy modeling for virtualized applications with dynamic, multi-type resource needs.

The experimental evaluations demonstrate that the fuzzy-modeling-based approach improves the accuracy in resource prediction by up to 31.4% and 5.2% compared to the conventional regression approaches. Both CPU and disk I/O bandwidth can be efficiently allocated online to a VM serving resource intensive workloads. As a result, the QoS target is met for 97% of the time and at the same time substantial resources (about 62.6% of CPU and 76.5% of disk I/O bandwidth) are saved in comparison to peak-load-based allocation.

1.2. Fuzzy Model Predictive Control (FMPC) Based VM Resource Management

In the above fuzzy-modeling-based resource management approach, a supplementary strategy is employed to deal with the situations where the VM's resource demand is misestimated. However, such an adaptation strategy requires sufficient qualified data to be collected within a short time period to update the system model. To address this limitation, we propose to integrate the fuzzy modeling approach with a predictive control based resource management system which allows a VM's resource allocation to be directly adjusted based on the difference between the application's performance feedback and the QoS target.

This approach is architected to answer two key questions: *first*, how to accurately capture the complex relationship between resource allocation and application performance, and *second*, how to adaptively optimize the resource allocations for competing VMs as changes occur dynamically in the system. The first question is answered by employing the fuzzy-logic based modeling method proposed above to learn the relationship between VM resource allocation and application performance, which can efficiently capture system

behaviors without requiring any a priori knowledge. The second question is addressed by using a new predictive controller to predict the resource demands for all VMs and take the resource control actions that enable the system to quickly reach its optimization objective. These two phases work in a closed-loop manner where the model is constructed and updated online and resource allocations are adjusted dynamically in order to track the QoS target and adapt to the changes in the system in a timely manner.

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

This proposed approach was prototyped on Xen-based virtualized systems and evaluated using typical benchmarks. The results demonstrate that FMPC can accurately estimate the resource demand for a VM running dynamically changing workload and quickly achieve the desired QoS target. FMPC can also capture the complex behaviors of resource competing VMs and optimize the resource allocations according to their QoS targets. It substantially outperforms the traditional linear model predictive control (LMPC) approach. Furthermore, the proposed two-level resource management framework can effectively optimize the performance for more than 100 concurrent VMs running dynamic workloads across multiple hosts.

1.3. Cross-Layer Optimization Based VM Resource Management

Based on the above fuzzy-modeling and control based resource management framework, the third component of this dissertation proposes cross-layer optimization which allows certain awareness and cooperation between a VM's host and guest in order to improve application performance and meet its QoS target. Specifically, two aspects of such cross-layer optimization are explored. First, guest-to-host optimization exploits guest-layer application knowledge to capture dynamic workload characteristics and improve modeling of VM resource usage. Second, host-to-guest optimization enables host-layer scheduler to feedback resource allocation decision and adapt guest-layer application configuration. These two aspects of cross-layer optimization are integrated into the aforementioned fuzzy-modeling-based resource management system which uses fuzzy logic to model VM resource demands online and allocate resources dynamically according to application QoS requirement.

As case studies, the proposed approach is applied to virtualized databases and map services which have challenging dynamic, complex resource demands and sophisticated configurations. Specifically, for databases, the proposed approach characterizes query workloads based on a database's internal cost estimation and 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. These case studies demonstrate the effectiveness of this approach and provides an experimental evaluation.

This approach 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. With the guest-to-host workload characterization, resources can be efficiently allocated to database VMs serving workloads with changing intensity and composition while meeting the QoS targets, improving the database performance by 17% compared to the allocation scheme without workload characterization. With the host-to-guest application adaptation, the performance of TPC-H-based workloads is improved by 17% while a map request workloads is improved by 15% in response time and 40% in map imagery quality, compared to schemes without adaptation

1.4. Organization of the Dissertation

The rest the dissertation presents the details of the three research components mentioned above. Chapter 2 introduces the background and related work. Chapter 3 presents the fuzzy modeling based resource management approach and discusses the management of virtualized database applications as a case study. Chapter 4 discusses the FMPC approach which integrates fuzzy modeling with predictive control for adaptive resource management in a dynamic system. Chapter 5 presents the cross-layer optimization approach which enables cooperation between a VM host and guest in order to improve application performance and resource usage efficiency. Finally, Chapter 6 concludes the dissertation with an outline of the future work.

2. BACKGROUND AND RELATED WORK

2.1. VM Based Computing System

The emergence of VMs is driven by the fast maturation and wide availability of virtualization technologies, as well as the rapid growth of computing power on modern computer systems. On one hand, VM technologies are already efficient and reliable enough to host mission-critical applications, and they are widely available for the virtualization of various types of system; on the other hand, the ever increasing computing power of today's computers has provided the necessary resources to host VMs. In particularly, multi-core and many-core CPUs are quickly emerging on not only high-end systems but also consumer products. VMs are particularly suited to provide space-sharing of resources for such systems.

The system-level VMs [1][2], which are based on the virtualization of an entire physical host's resources, including CPU, memory, and I/O devices, presenting virtual resources to the guest operating systems and applications. Such VMs are mainly implemented by the layer of software called Virtual Machine Monitor (VMM, a.k.a. hypervisor). Although our proposed techniques can also be applied to some other types of virtualization (e.g., OS-extension based VMs [5][6]), those are not the focus of this dissertation.

This dissertation considers the use of dedicated VMs to host different applications and allow them to transparently share the underlying resources. Because the multiplexing of applications to resources is provided at a lower level of the system, it has the following advantages compared to traditional OS-based resource sharing:

- VMs provide strong isolation for resource sharing, allowing applications on one VM to be protected from failures and security breaches occurred on another concurrently hosted VM;
- Virtualization supports flexible allocation of various types of resources to VMs, and VM migration further enables dynamic balancing of resource usages across physical hosts;
- VMs allow application-tailored customization of their execution environments, including OSes and libraries, and enable applications to be seamlessly deployed onto resources with heterogeneous configurations.

Virtualization provides promising platforms for building agile datacenters and emerging cloud systems [3][4]. In such a virtualized system, physical servers can be carved into multiple virtual resource containers, each delivering a powerful, secure, customizable, and portable execution environment for applications by hosting applications on dedicated VMs. 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.

2.2. Autonomic VM Resource Management

VM-based application hosting allows dynamic resource allocations based on the demands from applications, thereby improving the overall resource utilization. However, a key challenge to the success of this approach is how to allocate resources to a VM to achieve both the application desired QoS and the system desired resource efficiency, and how to do so for all the VMs automatically and continuously. To address this challenge, autonomic computing techniques can be employed to realize self-managing of VM resource configurations according to the high-level application performance and resource utilization objectives [7]. A Monitor-Analyze-Plan-Execute (MAPE) control loop [8] can be deployed to monitor the VM's workload demand, analyze its resource needs, plan its resource configuration, and then execute it accordingly. This dissertation follows this approach to build autonomic systems for the resource management of VM based hosting systems.

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. We classify the related work into three categories: (1) Queuing model based approach, (2) control theory based system, and (3) machine learning techniques. In this dissertation, our proposed resource management solutions belong to the second and third categories.

2.2.1. Queuing Model Based Resource Management

The first category of solutions employs queuing theory to construct analytical performance models for virtualized applications. For example, Doyle *et al*. derive analytical models from basic queuing theory to predict response times of Internet services under

different load and resource allocation [9]; Bennani *et al*. consider using multiclass queuing networks to predict the response time and throughput for online and batch workloads on VM based application environments [10]; Gulati *et al*. apply queueing model to build approximate IO performance model in a storage management system for virtualized data center[73]. However, solutions of this type are restricted by their often simplified assumptions on a virtualized system's internal structure, and are difficult to capture the system's complex resource usage behavior. Although Gandhi *et al*. employs a statistical technique to adapt the parameters for a queueing theoretic model to capture dynamics in the system without offline benchmarking, it focuses on more coarse-grained application scaling in the cloud [69].

2.2.2. Control Theory Based Resource Management

The second category of solutions applies control theory to adjust VM resource allocation and achieve the desired application performance or system-level objective. Such solutions often assume a linear relationship between QoS parameters and control parameters and involve a system identification phase to train the model parameters. In addition, the control parameters typically must be specified or configured offline on a per-workload basis. For example, Liu *et al*. consider the complex interactions and dependencies among different application tiers hosted on VMs and optimize their CPU allocations in order to achieve QoS differentiation among the multi-tier applications [11]. Its follow-up work [12] builds an online ARMA model for each application to represent the relationship between the allocations of multiple resources and normalized performance when the application tiers are hosted on VMs spanning across physical nodes. Linear MPC has also been studied to

capture the last-level cache interference between concurrent VMs and compensate its performance impact [13], which also points out that a nonlinear model can model such interference much more accurately. In a typical linear-model-based MPC approach, a linear model is assumed to approximate the nonlinear behavior within a limited region of an operation point while it can be updated adaptively as the system moves from one operating point to another. However, it remains challenging to perform optimized control continuously over the entire operating space.

In the related work on other aspects of system management, Wang *et al*. uses MPC to optimize the power consumption for multiple servers [51]; Lu *et al*. applies MPC to the control of CPU utilization in a highly coupled distributed real-time system [52].

In comparison, we combine the strength of machine learning with control theory, which does not require any a priori knowledge of the VM's system model, and can efficiently model a nonlinear system with dynamically changing resource usage behaviors. Compared to adaptive linear models in traditional control system, we build continuous nonlinear models to capture the system's entire behavior more accurately and allow optimized resource allocation over the entire operating space.

2.2.3. Machine Learning Based Resource Management

In the third category of solutions, machine learning techniques are extensively studied to address several major problems in VM-based resource management system.

A variety of machine learning techniques are applied to system modeling for prediction-based resource management. For example, simple regression method is used to predict the performance impact of VM memory allocation [14]; Regression method is also employed

to map a resource usage profile obtained on a physical system to that on a virtualized system [15]. However, these solutions often unable to capture the nonlinearity in a virtualized system's behaviors. Specially, their modeling accuracy is shown to be poor in modeling the performance of complex hosting applications [17]. Artificial neural network (ANN) and support vector machine (SVM) are then explored to build multi-dimension performance models to predict the resource needs of hosting applications given certain performance target [17][72]. These solutions identify three control knobs, CPU, memory limits, and disk I/O latency as the inputs of the model and collects performance measurements under various allocation configurations to build offline non-linear model to capture complex application behaviors. Compared to such a typical performance modeling approach, we focus on the autonomous management of both CPU and disk I/O allocations for virtualized applications in an online, adaptive way. The performance model used in our fuzzy model predictive control approach can be initialized using a small set of training data collected as the system starts. Online adaptive control is then enabled to adapt the model continuously to reflect the system changes by feed backing recent observations to the controller. Instead of evaluating the overall accuracy for modeling static application workloads, we demonstrate the effectiveness of our approach in tracking online performance target for representative workloads which change dynamically over time.

Other popular machine learning techniques also have been widely studied for online management scenarios. Reinforcement learning technique is used to automatically tune VM resource configuration such as CPU and memory to achieve good performance for hosting applications [16]; Signal processing technique is first employed to predict repeating resource usage patterns for applications and hosts in a cloud [18][19] and later used to

achieve online adaptive padding when resource needs are under-estimated [74]. Compared to those solutions which treat a VM as a black box, our application-aware management solutions takes advantage of application-specific knowledge to effectively capture the workload patterns and proactively optimize guest level performance. Clustering and classification methods are utilized in [66][67] to adapt resources allocations for dynamic workloads on the fly but these solutions still rely on offline profiling on small set of representative workloads.

In the related research on workload-aware resource management, k-means clustering combined with queuing models is employed to characterize workload with changes in both volume and mix for predicting server capacity [46]. In comparison, our workload characterization is performed more efficiently by leveraging the knowledge on resource estimation directly from the hosted application to cluster its workloads

Other related works have shown promising results for VM provision and configuration from a long-term prospective; mathematical models and clustering techniques are combined to detect interference between co-hosted VM and therefore guide VM placement [68]; Markov Decision Process (MDP)-based algorithm is used to make efficient VM migration decisions for long-term load balance [71]. While our solution focuses on the fine-grained resource allocations for VMs within a host, e.g., allocating CPU time slices and I/O bandwidth at short time scale, we also supports resource optimization across hosts in the units of VMs at a larger time scale through VM migration based on a two-level resource management framework.

2.3. Virtualized Database Hosting Systems

In the related research of virtualized database hosting systems, Farooq *et al*. experimentally evaluated VM-based databases and showed that the overhead is very small compared to natively hosted databases [24], which also confirms the feasibility of such approaches. Soror *et al*. address the problem of automatic resource configuration for database VMs by calibrating database's internal query cost model [25]. However, this work treats a workload as a static entity with a fixed set of queries, so the performance considered is the overall runtime and the VM configuration is done statically for the entire workload. The offline calibration process considers VM's use of CPU, memory, and I/Os as independent from each other, which may not hold due to the complexity of resource virtualization. When the database's cost model is inaccurate, this work employs online refinement by assuming a linear resource usage model. Therefore, it is unclear how this approach would apply to and how well it would perform for a workload with complex resource usage and dynamically changing behavior. In contrast, our application-aware approach uses database cost model only as a tool to discover workload composition, but not for directly estimating VM resource demands, thereby avoiding the well-known inaccuracy inherent to database cost models. In our solution, we more realistically treat 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.

Xiong *et al*. build probability-based classification model for incoming queries to make admission control decisions for database system to meet expected performance target [75]. Salomie *et al*. exploit ballooning technique to reallocate RAM for database system to

preserve SLAs while maximizing utilization[76]. Other related autonomous database work [26][27][28] focuses on a database's internal tuning and query optimization. Those solutions are all orthogonal and complementary to the problem addressed by this dissertation, which focuses on the resource allocation to an entire database VM.

Previous work on workload characterization [29][30] also considers it as the key to understanding the resource intensity of a database workload. In these studies, a workload is often described with time-invariant structure and parameters, which is far from the real-world situations. We incorporate both of these two aspects in 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.

3. FUZZY-MODELING BASED RESOURCE MANAGEMENT

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

To address the above challenges, this chapter presents a fuzzy-modeling based approach for on-demand allocation of multiple types of resources to a VM running dynamic and complex workloads while meeting the QoS requirement. Without any *a priori* knowledge of the system's internal structure, the fuzzy modeling approach can accurately describe complex and nonlinear system behaviors and can dynamically adapt to the changes in workload.

3.1. Motivation

3.1.1. Virtualized Hosting System

Traditionally, applications 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 [20] and stream data management [21], where the workloads are intrinsically dynamic in terms of their bursty arrival patterns and ever-changing unit processing costs. Even under domains where traditional static workload exists, it can dynamically switch from one workload to another at runtime. For example, an online vendor database that serves large number of user queries during the day may switch to internal bookkeeping jobs early in the morning.

Therefore, the limitations of the traditional application hosting approach are two-fold. First, peak-load based resource provision leads to overprovision and thus underutilization of resources for normal state workloads. This can cause considerable infrastructural and operational overhead. Second, as a steady-state workload demand exceeds its previously expected peak value, the application performance may drop dramatically due to overload, unless it can be moved to a more powerful server through a lengthy relocation process.

Using VMs to host applications can effectively address the above limitations, because virtualized resources, including CPU, memory, and I/O, are decoupled from their physical infrastructure and can be flexibly allocated to the application as needed. Virtualization can consolidate many dedicatedly provisioned physical servers into a small number of shared ones, where each of them can be carved into multiple virtual resource containers to provision

resources to applications. By hosting the applications on dedicated VMs separately, it allows the application to share the consolidated resources with others with strong isolation. It also allows the resource allocation to the application VM elastically grow and shrink based on the application workload's demand. In addition, application VMs can be dynamically migrated across physical machines for resource optimization.

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

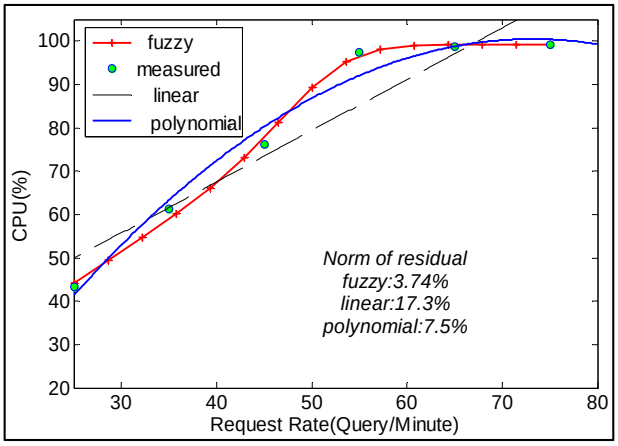

Figure 3-1 CPU models for TPC-H experiment

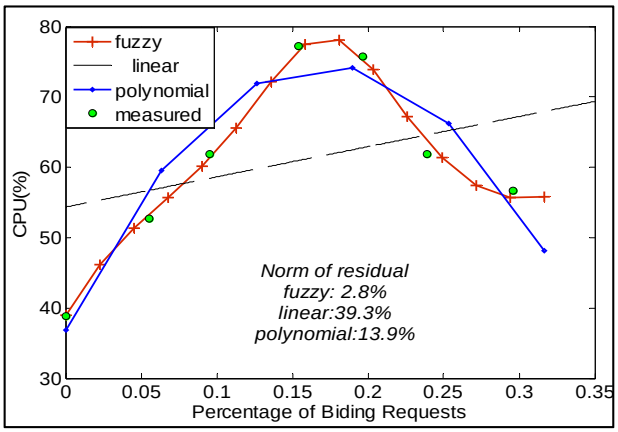

Figure 3-2 CPU models for RUBiS experiment

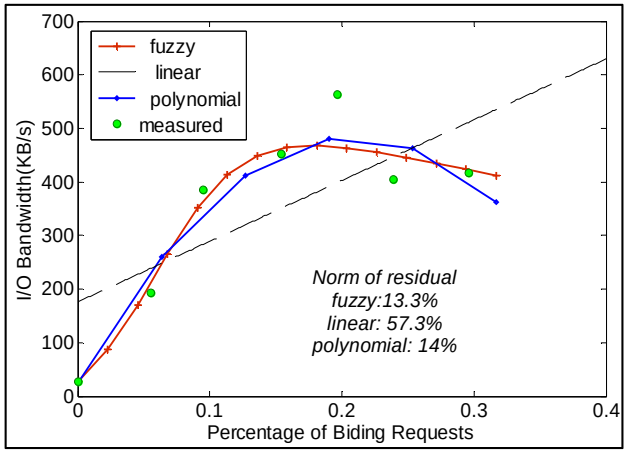

Figure 3-3 I/O models for RUBiS experiment

3.1.2. Non-linearity in Virtualized System

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

In the first example, a synthetic database workload based on a sequence of TPC-H [39] queries is executed on a database VM. We gradually increase the workload intensity by adjusting the query request rate until the virtualized database becomes saturated. Figure 3-1 plots the observed average CPU usage of the database VM as the request rate is increased from 35 to 75 request/minute. The nonlinearity in such an OLAP database is evident as the request rate exceeds around 55 query/minute and the system becomes saturated.

The second example considers a typical multi-tier OLTP benchmark, RUBiS [40]. We fix the database tier's query workload intensity by running 1000 concurrent client sessions in RUBiS. But we vary the composition of the query workload by increasing the ratio of bidding and browsing requests to the web tier which correspond to read and write queries, respectively, to the database tier. Nonlinearity is apparent in the CPU and disk bandwidth usages (Figure 3-2 and Figure 3-3) of such an OLTP database's behavior, even though the system is not under saturation.

We then study the accuracy of applying fuzzy modeling to the database VMs in the above two examples and compare it to another two commonly used modeling methods, the simple linear regression and the more complex second-order polynomial fitting. The models created by these different methods along with the measured data are shown in Figure 3-2. The figure also shows their *norm of the residuals*, a common metric for evaluating the goodness of a model, which is defined as the square root of the sum of the squares of the differences between the predicted values and the actual values. The results show that linear model (*linear*) poorly fits the data points; the polynomial model (*polynomial*) can only reflect the trend of the resource need but cannot predict accurately the amount of necessary resources; only the fuzzy model is accurate regarding the entire data set which represent the complete resource usage behaviors of the database VMs. As we will further demonstrate in Section 3.3, our proposed fuzzy-modeling-based approach outperforms others in terms of its accuracy and efficiency.

Note that such modeling-based resource management is different from a typical feedback-control-based approach in which the application's actual performance is used to directly adjust the resource allocation in order to achieve the QoS target. In our modeling-based approach, a model is first built to capture the relationship between the application workload and its resource needs for the QoS target, and then used to predict the necessary resource allocation for the current workload demand. Although fuzzy-logic-based feedback controllers also exist [41], the key difference between our approach and those still lies in the fact that fuzzy logic is used to build a model for the managed system instead of to directly decide how to control the system.

3.2. Background in Fuzzy-logic Based Modeling

Fuzzy modeling combines fuzzy logic with mathematical equations to describe the discovered patterns of system behavior and to guide the control strategies of the system [31]. A fuzzy model is a rule base which consists of a collection of *fuzzy rules* in the form of "If x is A then y is B", where A and B are determined by fuzzy sets with associated membership functions. Contrast to a crisp set, a fuzzy set allows partial set memberships which can be quantified into numeric values based on a membership function. Commonly used membership functions are *Gaussian*, *Sigmoidal*, *Triangular*, *Trapezoidal* function, etc. The fuzzy rules in a fuzzy model are trained using the input (x) and output (y) observed from the system and together they compose the model representing the system behavior.

While building a fuzzy model, *data clustering* techniques (e.g.,[32]) are often employed to discover the important features of the system and derive a concise representation of the system's behavior. Each cluster is treated as a fuzzy set and then each set is associated with a fuzzy rule. As a result, only a small number of fuzzy rules are needed in the fuzzy model. For example, the model for a database VM from the experiment discussed in Section 3.3.2 is as follows,

- R^1: If $[C_1, C_2]^T$ is in $cluster_1$, then $r_{CPU} = [8.8\ 6.3][C_1, C_2]^T + 3.1$
- R^2: If $[C_1, C_2]^T$ is in $cluster_2$, then $r_{CPU} = [-0.5\ 1.5][C_1, C_2]^T + 88$
- R^3: If $[C_1, C_2]^T$ is in $cluster_3$, then $r_{CPU} = [12.8\ 0.5][C_1, C_2]^T + 41$
- R^4: If $[C_1, C_2]^T$ is in $cluster_4$, then $r_{CPU} = [8.3\ 2.1][C_1, C_2]^T - 68$

The input of the model is the query workload described by a vector of request rates of two types of queries, $[C_1\ C_2]^T$, while the output of the model is the CPU resource usage, r_{cpu}.

Given a total of 225 input-output data pairs measured in the experiment, clustering technique is used to produce only 4 clusters which can effectively represent the entire dataset. Each cluster is then treated as a fuzzy set and associated with a fuzzy rule as part of the database VM's model.

The mapping from a given input to an output on a fuzzy rule base is called *fuzzy inference*, which entails the following steps: 1) Evaluation of antecedents: the input variables are *fuzzified* to the degree to which they belong to each of the appropriate fuzzy sets via the corresponding membership functions, 2) Implication to consequents: implication is performed on each fuzzy rule by modifying the fuzzy set in the consequent to the degree specified by the antecedent; 3) Aggregation of consequents: the outputs of all the fuzzy rules are aggregated into a single fuzzy set which is then inversely translated into a single numeric value through a *defuzzification* method. Following the above example, given a specific workload input $[C_1, C_2]^T$, the fuzzy model learned from the TPC-H based experiment can be used to predict the CPU demand r_{cpu} following the above steps. Note that this fuzzy-modeling approach is fundamentally different from traditional rule-based system management approach [37][38]. The latter is based on the use of a set of event-condition-action rules that are triggered only when certain events happen and some preconditions are met. In such an approach, the rules are typically specified by system experts, which is often intractable to apply to a complex system because of the difficulty in defining thresholds and corrective actions for all possible system states. In contrast, a fuzzy model is built for the entire input space of the system and can be used for continuous control, where the fuzzy rules representing the model are created automatically from the observed input-output data.

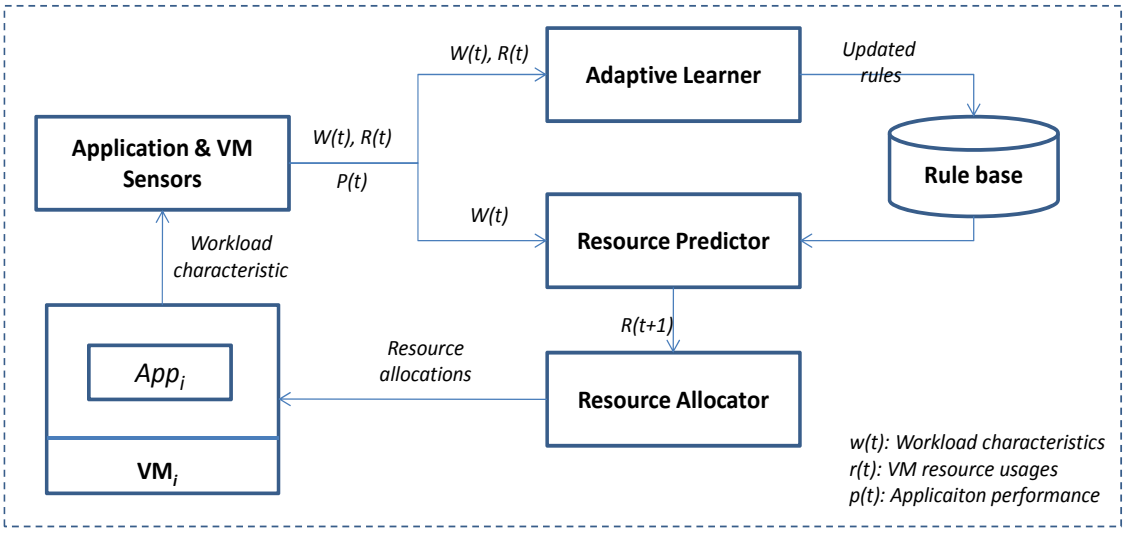

Figure 3-4 Architecture of the autonomic resource management system

3.3. Fuzzy Modeling Based VM Management

Figure 3-4 illustrates the architecture of our proposed resource management for VM-based applications based on the aforementioned fuzzy modeling approach. This system consists of four key modules. As a workload executes on the application VM, the *Application and VM Sensors* monitor the workload $W(t)$, its performance $P(t)$, and the VM's resource usage $R(t)$. With this model and the current workload $W(t)$, the *Resource Predictor* estimates the resource need for time $t+1$ and the *Resource Allocator* adjusts the allocation accordingly. Together, these modules form a closed-loop for the VM's resource control and optimization. They are described in detail in the rest of this section.

3.3.1. Application and VM Sensors

In order to modeling resource usage for the application workload, first of all the workload needs to be abstracted as one components of the inputs for the model. *Application Sensor* is responsible for extracting the characteristics of a workload that is relevant to its resource

usage behaviors when executed on an application VM. Such characteristics provide important inputs to the effective modeling and prediction of an application VM's resource needs. A commonly used workload characteristic is the request rate which describes the workload's overall intensity and is often strongly correlated with its resource demand. However, the characterization of an application workload is more challenging, because it can consist of different request with diverse use of multiple types of resources. To address this challenge, we propose to characterize a workload by first classifying its requests into a small number of groups based on their resource usage patterns and then describing the whole workload as a vector of arrival rates of these groups. This workload characterization process can be done by leveraging the intimate knowledge of application, for example we make advantage of a database's internal cost model to clustering queries according to the estimates on their resource usage, which will be discussed in details based on a cross-layer optimization approach in Chapter 4.3.

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

The *VM Sensor* monitors a VM's resource consumption, which is the other key piece of information for modeling the VM's resource usage behavior. The monitoring has to be done

outside of the VM, because the application's resource usage inside of the VM does not truthfully represent its entire VM's resource usage which entails overhead from both the guest operating system and the use of virtualization. The *VM Sensor* in our system monitors multiple types of resources including CPU, memory, and disk and network I/Os, as a database VM can make intensive use of multi-type resources.

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

3.3.2. Adaptive Learner and Resource Predictor

The *Adaptive Learner* creates and updates the model that represents the relationship between an application workload and its VM's resource need. It employs the fuzzy modeling approach to automatically discover this relationship, where fuzzy rules are constructed based on the input and output data pairs, $<W(t), R(t)>$, collected by the Application and VM Sensors. Both the workload input $W(t)$ and the resource usage output $r(t)$ can be vectors with multiple dimensions. For $W(t)$, each dimension represents a certain

characteristic of the workload and for $R(t)$ each dimension maps to one type of resources. In order to learn a model that represents the resource needs of the VM for a specific QoS target, only qualified input-output data pairs $<W(t), R(t)>$ whose workload performance $P(t)$ meet the QoS target are fed to the *Adaptive Learner*. In this way, the resulting model trained based on the filtered data can capture the VM's resource needs in order to meet the given QoS target. When the QoS target changes, the model will be different as the qualified training data change.

While creating a fuzzy rule base from the qualified input-output data, it is inefficient to generate one rule for every specific data pair. In order to build a concise fuzzy rule base with a small number of rules that can still effectively represent the VM's behavior, a clustering method is used to group similar data points into clusters. In particular, the *Adaptive Learner* adopts an efficient one-pass clustering algorithm, subtractive clustering [32]. Each resulting cluster exemplifies a representative characteristic of the system behavior and can be used to create a fuzzy rule accordingly.

The *Adaptive Leaner* generates *Sugeno*-type fuzzy rules [31] from the clustered data for modeling the application VM. This type of fuzzy rules uses a crisp, linear or constant function as the membership function, which is suitable for mathematical analysis. Suppose for input the workload $W(t)$ is described by N different characteristics, $[C_1, C_2, \ldots, C_N]$ and for output, two types of resources, CPU and I/O, $[R_{CPU}, R_{IO}]$, are consumed. If K clusters are formed from all the data pairs, then K rules are produced for this fuzzy model. The rule base is constructed as follows:

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

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

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

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

modeling, this whole model updating process can be completed quickly (typically under a second) for fine-grained resource control interval.

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

3.3.3. Resource Allocator

In a virtualized system, a VM serves as a resource container to the hosted database, where different types of resources can be dynamically allocated to this container for serving its workload. This is in contrast to traditional, non-virtualized hosting, where an application's resource availability is statically defined by its physical machine's configuration. The *Resource Allocator* periodically (e.g., every 10 seconds) adjusts the multi-type resource allocation to VMs based on the *Resource Predictor*'s estimate. The *Resource Allocator* also needs to deal with situations where the resource prediction is inaccurate and causes the application performance to diverge from the QoS target. This happens when the application

workload is first started or when its resource usage behavior changes so the *Adaptive Learner* cannot properly model the VM's current behavior.

In our approach, a backup resource allocation policy is employed to quickly recover from performance loss resulted from QoS violations when the VM's resource need is underestimated due to inaccurate workload modeling. This backup policy is invoked based on the recent information on the application's performance measurement $P(t)$, for instance, after the QoS target is missed for several (e.g., two) consecutive periods of time. This backup policy increases the current resource allocation by a fixed percentage (e.g., 100%) in order to satisfy the VM's unknown resource need which is beyond its previous resource allocation level. (The choice of how soon to invoke the backup policy when a QoS violation happens is studied in section 3.4.4.) This fixed increment of resource allocation is accumulated until the QoS comes back to the target value, and afterwards the resource allocation is sustained at that level until the target is met for several (e.g., two) consecutive periods of time. Because the VM resource usage can be controlled at a fine granularity (in the matter of seconds), this mechanism allows the performance loss to be quickly recovered. Meanwhile, it also allows qualified data points to become quickly available so that the model can be timely updated to correctly reflect the VM's current resource needs.

However, the backup policy is only a supplemental method to our proposed fuzzy-modeling-based resource allocation. Although in the form of a traditional event-condition-action rule, it cannot substitute for the fuzzy model. The event-condition-action rules have to be predefined based on experts' knowledge, while the fuzzy model is automatically learned from the controlled system. Further, the event-condition-action rules are often statically defined, while the fuzzy model can be updated online to adapt to the changes in

the system. Hence, the backup policy is only triggered when the model is inaccurate and unable to get qualified data to update itself. With the assumption that a workload's stable phases are much longer than its transition phases, the fuzzy model should be able to correctly predict the resource needs for most of the time therefore the backup policy would only be used infrequently.

3.4. Evaluation

This section evaluates our proposed approach on a virtualized database system which is considered as challenging and interesting case study for applying the fuzzy-logic based modeling due to its dynamic and complex behaviors [22]. Although our previous work successfully applied fuzzy modeling to control CPU allocation for VMs hosting CPU-intensive applications [23], the evaluation of the management of database VMs answers the following unique, important research questions: 1) How to effectively manage a VM with correlated, multi-type resource need, including not only CPU cycles but also I/O bandwidth? 2) How to timely adapt to the dynamic changes in a VM's resource need in terms of not only varying intensity but also shifting demand across different resource types?

3.4.1. Setup

The testbed is a quad-core Intel Q6600 2.4GHz physical machine with 4GB RAM and 142GB SATA disk. Xen 3.3.1 is installed to provide the VMs, where the operating system for both Dom0 and DomU VMs is Ubuntu Linux 8.10 with paravirtualized kernel 2.6.18.8. The evaluated databases are hosted on DomUs, while our resource management system is hosted on Dom0. In all the experiments, the management system monitors and controls the database VM's usage of *both* CPU cycles and disk I/O bandwidth every *10* seconds. In the

VM Sensor, resource monitoring is done using xentop and iostat, where the I/O bandwidth usage is considered as the sum of reads and writes per period of time. In the *Application Sensor*, a database proxy deployed on Dom0 is used to measure the performance of the database VM. The *Resource Allocator* uses Xen's sEDF CPU scheduler to assign CPU allocations and Linux's dm-ioband I/O controller to set the cap for disk I/O bandwidth [42]. The sEDF scheduler uses 100ms period in the work-conserving mode. Another DomU VM running a CPU-intensive program is pinned on the same core assigned to the database VM to consume the surplus CPU cycles. Other VMs involved in our experiments are served as clients running outside of our testbed.

Two typical database benchmarks, TPC-H [39] and RUBiS [40], are used in our experiments. The performance metrics considered in TPC-H include both average query throughput and average query response time measured every 10s. But in RUBiS only response time is considered because it is strongly correlated with throughput for this benchmark. Two different resource allocation schemes are compared: 1) The *peak-load-based resource allocation*, where the database VM is statically allocated sufficient resources based on its peak-load demand; 2) The *fuzzy-modeling-based resource allocation*, where the VM's resources are dynamically allocated based on our proposed approach. By comparing the VM's resource usage and the benchmark's performance between these two cases, we evaluate whether our proposed approach can achieve the same level of QoS while saving resources compared to peak-load based static resource allocation.

3.4.2. TPC-H Experiments

TPC-H provides 22 representative queries of business decision support systems, which involve the processing of large volumes of data with a high degree of complexity. Based on these queries, we construct synthetic workloads with varying demands of different types of resources. With peak-load based allocation, 100% CPU and 12MB/s or 10 MB/s I/O are allocated to the database VM statically. With fuzzy-modeling-based allocation, there are two phases involved. In the training phase, the fuzzy model is learned without resource restrictions, while in the testing phase the model is applied to predict the resource demand and control the resource allocation. The evaluation of more realistic workloads with online training is discussed in Section 3.4.3. The database used here is PostgresSQL 8.1.3 with 2 GB of data, hosted on a VM with one CPU and 1GB RAM.

We characterize a TPC-H workload by classifying its 22 standard queries into four clusters. Each cluster identifies one type of query with similar resource usage pattern. *Cluster I* containing single query Q1 and *Cluster II* containing single query Q18 represent highly and moderately CPU-intensive query, respectively. *Cluster III* including Q4, Q6, Q15 and Q12 represents highly I/O-intensive queries. *Cluster IV* including most of the remaining queries represents simple queries which are neither CPU nor I/O intensive. This workload characterization can be performed based on the cost estimation extracted from PostgresSQL using a cross-layer optimization approach which will be illustrated in the following chapter. The resulting clusters are experimentally verified by the actual resource usages when running the queries separately on the database VM.

a) CPU-intensive Workload

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

The CPU allocation and workload performance from using the fuzzy-modeling-based resource allocation and the peak-load-based resource allocation are compared in Figure 3-5, Figure 3-6 and Figure 3-7. Note that both the workload performance and resource allocation shown from fuzzy-modeling-based approach are average values calculated from the measurements for each specific request rate. The performance obtained in the fuzzy-modeling-based allocation is always at the same level as the peak-load-based allocation even when the system becomes saturated after the request rate exceeds 55 query/minute. This demonstrates that our proposed fuzzy model is able to capture complex system behaviors over large region of the operating space. The throughput is within 96.4% to 100% of the peak-load-based allocation, while the average response times only increase by at most two seconds. (The throughput is expressed in terms of number of completed queries every 10s, because these queries are complex and time-consuming.) At the same time, substantial amount of CPU allocation is saved when the workload is below the peak load. Note that,

because of the difference in CPU intensity between Cluster I and II queries, the VM's CPU need changes as the ratio of these two clusters varies. Our

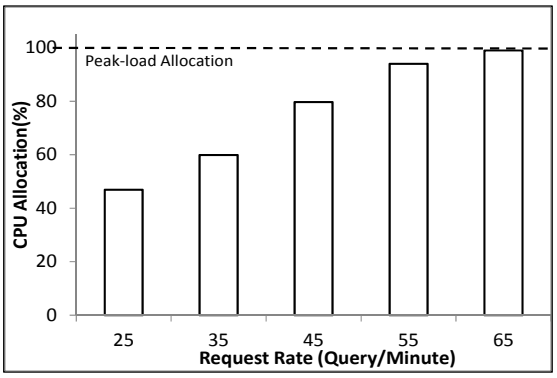

Figure 3-5 CPU allocation for the CPU-intensive TPC-H workload

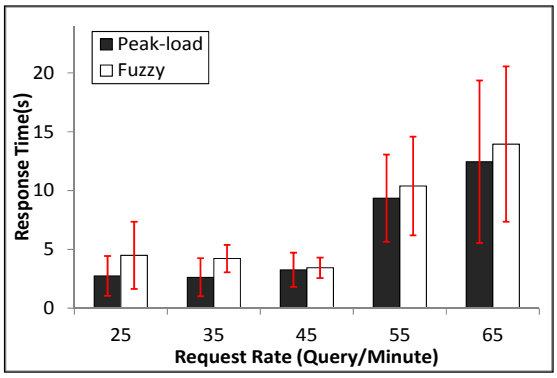

Figure 3-6 Response time for the CPU-intensive TPC-H workload

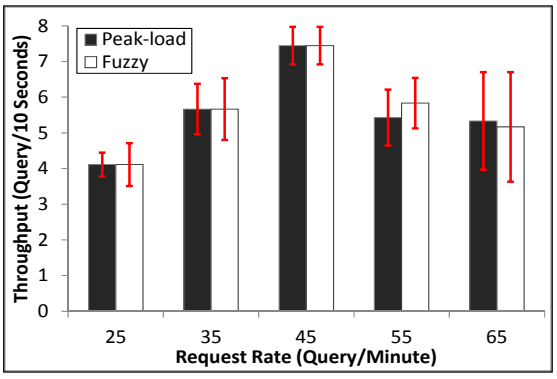

Figure 3-7 Throughput for the CPU-intensive TPC-H workload

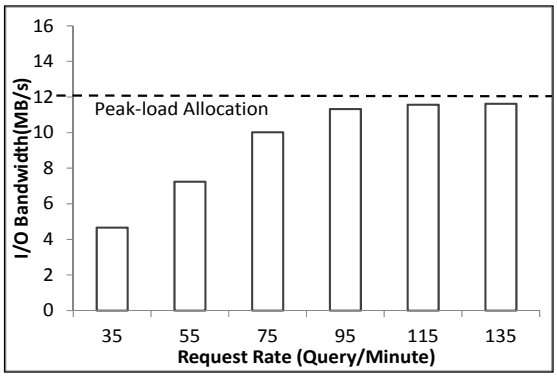

Figure 3-8 I/O bandwidth allocation for the I/O-intensive TPC-H workload

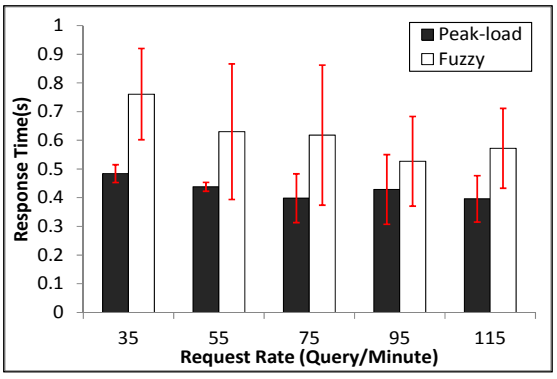

Figure 3-9 Response time for the I/O-intensive TPC-H workload

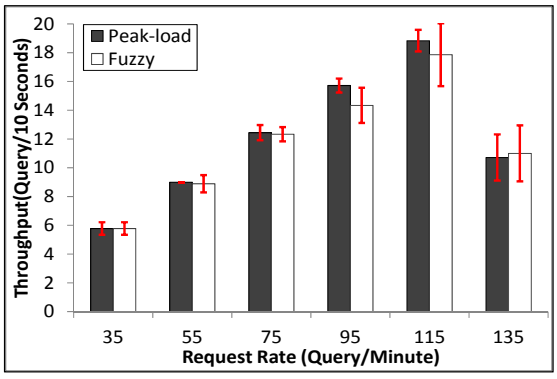

Figure 3-10 Throughput for the I/O-intensive TPC-H workload

approach can also properly model this behavior and accurately predict the VM's CPU need by taking this ratio as another input to the modeling. These results are omitted due to the limited space.

b) I/O-intensive Workload

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

Figure 3-8, Figure 3-9 and Figure 3-10 compare the I/O bandwidth allocation, query response time, and query throughput between using fuzzy-modeling-based and peak-load-based resource allocation. (The response times for the request rate of 135, not shown in the figure due to the large magnitude, are 50.6s and 52.9s for peak-load-based and fuzzy-modeling-based allocations respectively. The CPU allocations are also omitted because this experiment is not CPU-intensive.) The results also demonstrate that our approach can accurately model the database VM's I/O bandwidth need for such an I/O intensive workload. The throughput is within 89.5% to 100% of the as the peak-load-based allocation, but up to 30% increase in response time is observed. We believe that this overhead is due to the non-work-conserving nature of the dm-ioband I/O bandwidth controller, which increases the queuing delay of the queries, affecting only the query response time but not the throughput. We will investigate how to improve dm-ioband for query response time in

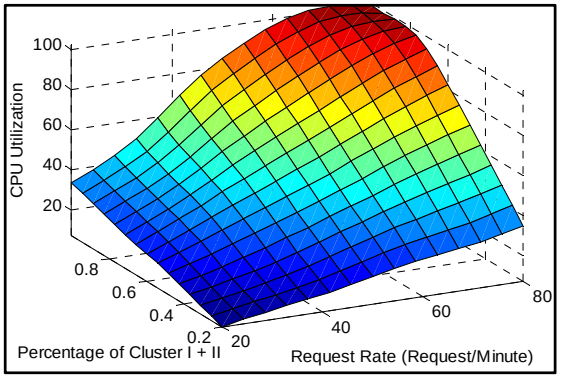

Figure 3-11 CPU model for the CPU/IO-intensive TPC-H workload

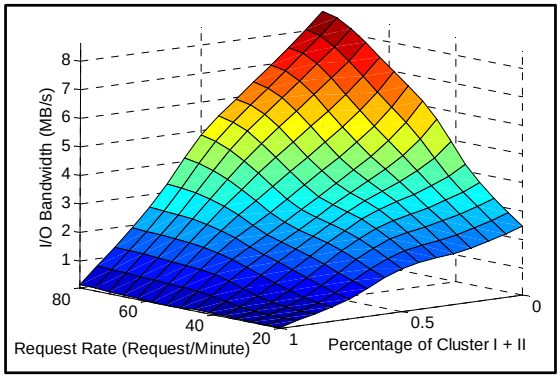

Figure 3-12 I/O model for the CPU/IO-intensive TPC-H workload

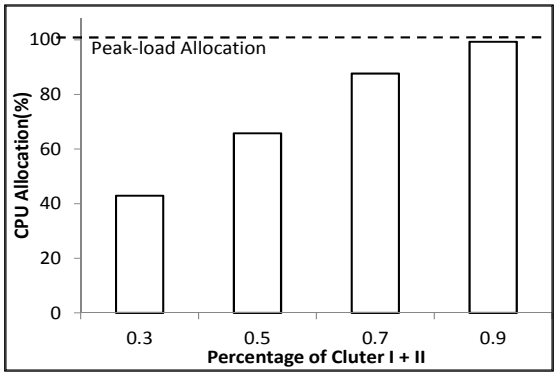

Figure 3-13 CPU allocation for the CPU/IO-intensive TPC-H workload

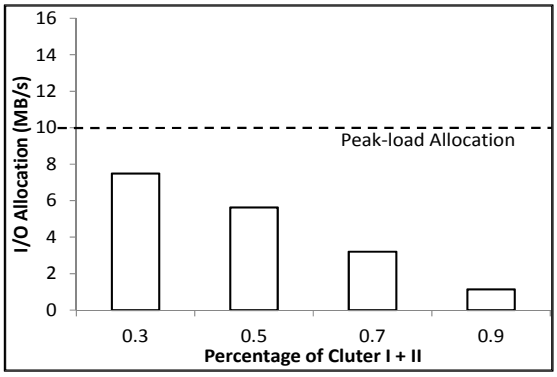

Figure 3-14 I/O allocation for the CPU/IO-intensive TPC-H workload

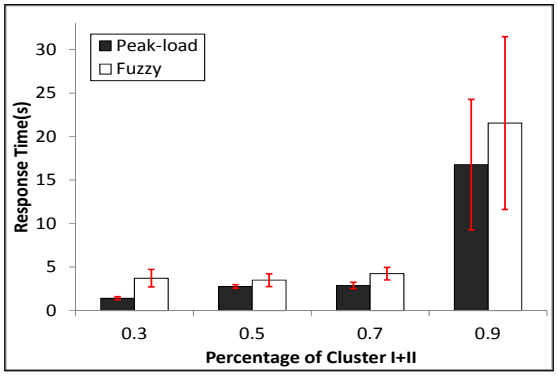

Figure 3-15 Response time for the CPU/IO-intensive TPC-H workload

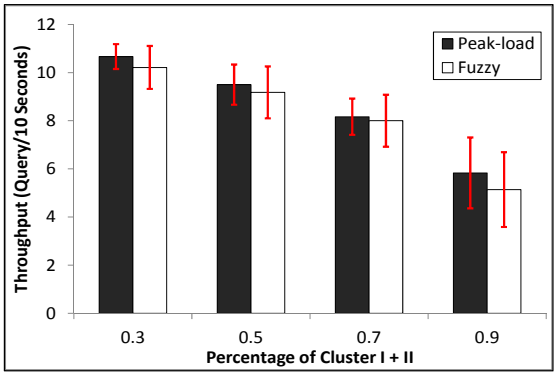

Figure 3-16 Throughput for the CPU/IO-intensive TPC-H workload

our future work. Nonetheless, substantial amount of I/O bandwidth is still saved using the fuzzy-modeling-based approach when the workload is below the peak load.

c) CPU/IO-intensive Workload

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

This experiment is designed to evaluate our approach's ability to model a both CPU- and I/O-intensive workload with both changing intensity and changing composition. The model's input, the workload is characterized by both the total request rate and the ratio of Cluster I+II and Cluster III queries. The resulting model is illustrated by two 3-D sub-models each consisting of 12 fuzzy rules. The results show that our approach can properly capture such complex behaviors of the database VM. From Figure 3-13 to Figure 3-16 show the resource allocation and workload performance when the request rate is fixed at 75 requests per minute but the Cluster I+II/Cluster III ratio varies. Compared to using peak-load-based resource allocation, the performance degradation from using fuzzy-modeling-based allocation is less than 5s in average response time and less than 10% in throughput, while saving both CPU and I/O bandwidth allocations. (The results from other request rates are similar and omitted here.) These results show that the VM's fuzzy model can accurately predict both its CPU and I/O need and the resource management system can effectively control them simultaneously, delivering good QoS to such a both CPU- and I/O-intensive workload.

3.4.3. RUBiS Experiments

RUBiS models an online auction site that supports the core functionalities such as browsing, selling, and bidding [40]. A typical two-tier setup is used to set up RUBiS, where the Web tier and database tier are deployed on separated VMs. The Web-tier VM hosts Apache Tomcat 4.1.40 with RUBiS and its clients while the database-tier VM hosts MySQL 5.0 with 1.1 GB of data. Both VMs are configured with one CPU and 1GB RAM. Since these experiments are performed *completely online*, only the first 10 data points collected are used to initialize the model. Afterwards the model is used to allocate resources right away and in the meantime it is updated with new observed data every 10s. By interposing a MySQL proxy before the database tier, our system characterizes its query workload online in terms of intensity and composition. The composition can be captured by the ratio of two types of queries, the SELECT queries, which are read-only, and the INSERT and UPDATE queries, which are writes to the database.

a) Simulation of Real-world Workload

Compared to the synthetic workloads used in the above TPC-H experiments, here we constructed two more realistic workloads, one with changing intensity and the other with changing composition, based on real traces from the 1998 World Cup site [43]. This method is similar to those used by the related work for creating realistic workloads [44][45].

The first workload with changing intensity is a browsing-only mix (Figure 3-17) derived from a typical one-day hourly trace from the World Cup site. We first vertically scale the range of request rate to what our RUBiS setup can handle, i.e., mapping [50000, 100000] request/hour in the World Cup trace to [0, 1000] request/second in the RUBiS workload.

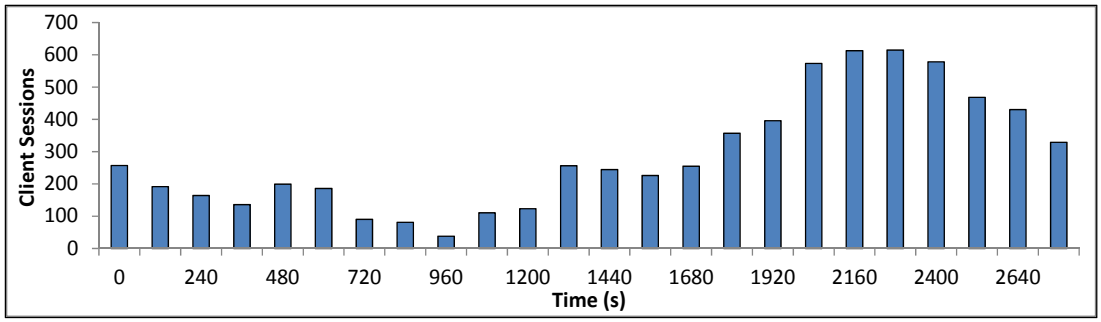

Figure 3-17 Trace for RUBiS with changing intensity

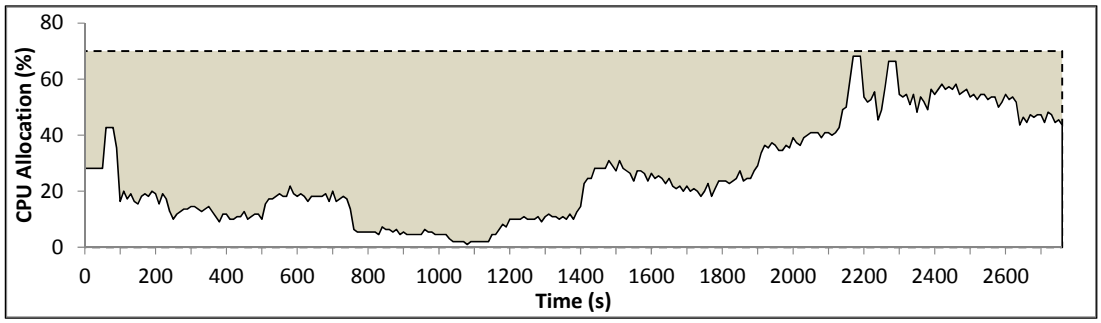

Figure 3-18 CPU allocation for changing intensity workload

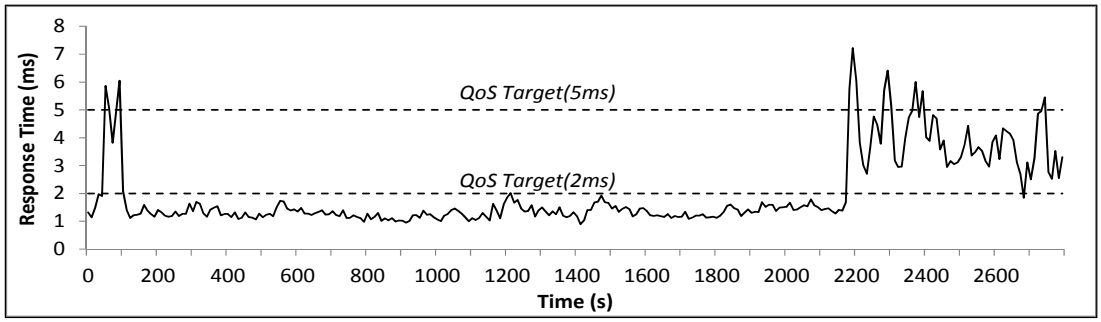

Figure 3-19 Performance for changing intensity workload

Second, we horizontally scale the duration of workload from 24 hours to 2880 seconds, to speed up the replay of the trace. Since the workload intensity in RUBiS is controlled by the number of concurrent client sessions to the web tier, another mapping is created from the desired request rates to the number of client sessions.

The second workload is constructed in a similar way but we place emphasis on the variation in workload composition while keeping its intensity constant (the number of

concurrent client sessions to the web tier is fixed at 800). Another one-day hourly trace with a stable request rate is chosen to derive this workload. We identify the read and write requests in the World Cup trace based on the "Get" and "Post" method, respectively, used in each request. The ratio of the read and write requests in this trace is then mapped to the ratio of the browsing and bidding requests in the RUBiS workload (Figure 3-17), which corresponds to the SELECT to INSERT/UPDATE ratio to its database workload.

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

b) Results

Figure 3-18 and Figure 3-19 show the CPU allocation and query performance of the database VM for the changing intensity workload (the I/O allocation result is omitted because this workload is not I/O intensive). As soon as the model is initialized through the first ten data points, it is able to accurately predict the VM's resource need throughout most of the experiment even when the burst occurs at time 480s, 1450s, and 1930s without using the backup resource allocation policy. At time 2100s, the system is under its peak load, current model underestimates the CPU need and the backup resource allocation policy is triggered to ensure the availability of qualified data for model adaption. After two control periods (20s), the model is adapted to the new system behavior and able to correctly predict the new resource need while the backup policy is stopped. The shaded area in Figure 3-18 illustrates the amount of resource saved (62.6%) in fuzzy-modeling-based resource

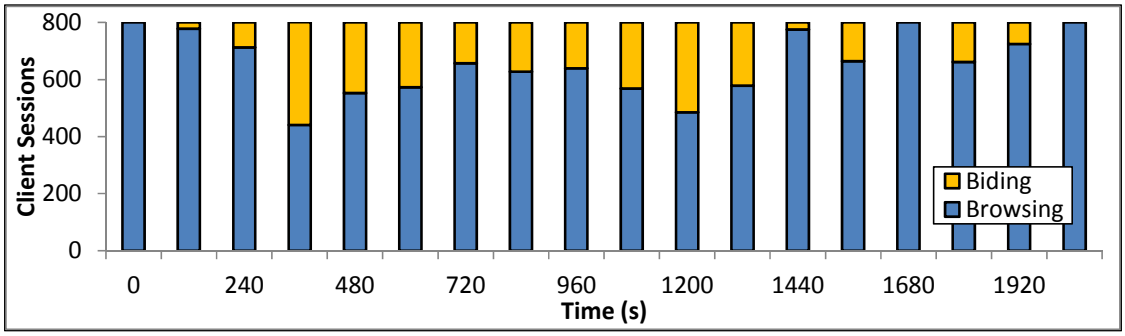

Figure 3-20 Trace for RUBiS with changing composition

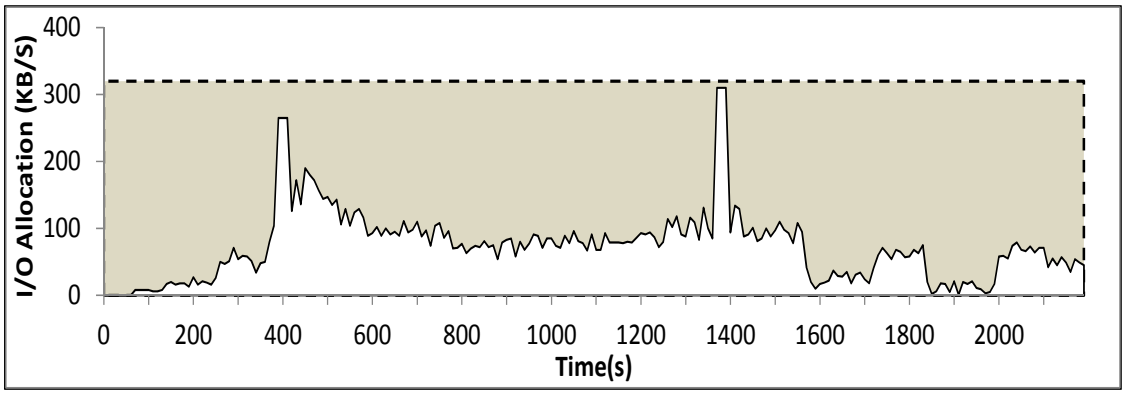

Figure 3-21 I/O allocation for changing composition workload

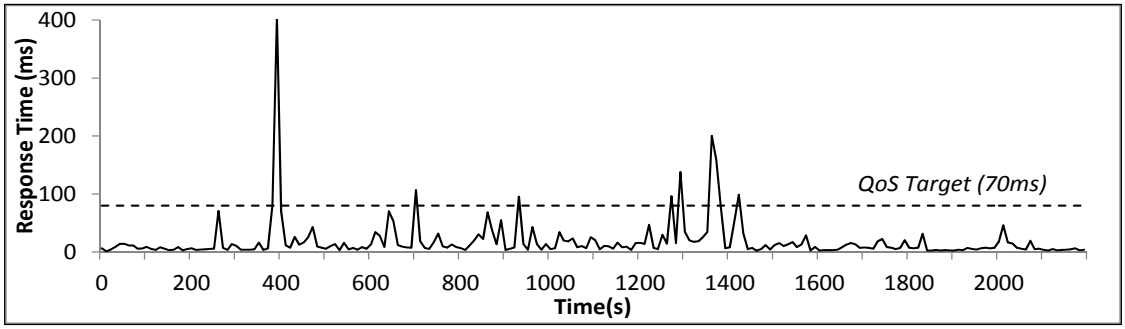

Figure 3-22 Performance for changing composition workload

allocation. Figure 3-19 shows that the average query response time can meet the desired QoS target most of the time (Only 11 QoS violation periods occurred throughout the entire experiment).

Figure 3-21 and Figure 3-22 show the I/O allocation and query performance of the database VM when running the changing composition workload (the CPU allocation results

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

3.4.4. Modeling Sensitivity and Overhead

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

In the last experiment, we study the sensitivity to this threshold of our proposed approach study using a workload with changing composition created by switching between four mixes, each producing a constant percentage of write queries, 0%, 4%, 8%, and 20% respectively, to the database tier. Each mix lasts 300 seconds and then transits immediately to the next mix. The number of concurrent client sessions is kept at 200. We run this workload on RUBiS and use the fuzzy-modeling-based resource allocation with different

threshold values. The result shows that the same level of average throughput (21 query/s) can be achieved when the threshold value varies from 1, 2, to 4, but the total number of uses of the backup policy drops from 12, 2, to 1, respectively (the figures are omitted due to limited space). It confirms that if the threshold is set lower, the backup policy is invoked more often while it is set higher, longer QoS violations are experienced during the transitions. The result verifies that the threshold value of two is a good choice but in general this tradeoff can be determined by considering both the QoS requirement and resource cost.

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

3.5. Summary

Virtualization can greatly facilitate the deployment of applications and substantially improve the resource utilization of the hosting system. To fulfill this potential, resource management is the key, which should be able to automatically allocate resources to VMs based on their QoS targets. This chapter presents an autonomic resource management system that can achieve this goal through a fuzzy modeling based approach, which models a VM's resource usage behaviors based on observed data and predicts its resource needs for

its current workload demand. This process is done periodically (in terms of seconds) online to guide dynamic resource allocation and adapt to changes in the system. Experiments based on typical database benchmarks show that our system can accurately estimate a database VM's resource needs for dynamic and complex query workloads, meet the desired query QoS, and save substantial resources compared to peak-load based static allocation.

4. FUZZY MODEL PREDICTIVE CONTROL BASED RESOURCE MANAGEMENT

In the previous chapter, our fuzzy-modeling-based approach relies on a predefined backup policy to deal with situations where the VM's resource demand is misestimated due to dynamic changes in the VM's resource usage behaviors. However, empirical knowledge is needed to decide factors such as how many consecutive QoS violations should be observed before invoking the backup policy, and how much resources needs to be added on the current allocation when resource is under provision. In this chapter, we study a new fuzzy-model-predictive-control (FMPC) approach which better addresses this limitation by automatically adjusting the allocations based on performance error instead of manually increasing fixed amount of resources. Then it is further incorporated in a two-level cloud resource management framework to manage VMs across multiple hosts based on system-level objectives.

4.1. Background

4.1.1. Adaptive Virtual Resource Management

Emerging virtualized systems such as utility datacenters and clouds promise to be important new computing platforms where applications could be executed efficiently and resources could be utilized efficiently. A key challenge to fulfilling this promise is to correctly understand an application's VM's resource demand based on its QoS target and effectively optimize the resource allocation across VMs based on resource-provider

objectives. The major difficulty lies in the intrinsically dynamic and complex nature in the resource usage behaviors in such virtualized system.

First, the dynamics in an application's workload can lead to complex behaviors in its VM's resource usages as its intensity and composition change over time. For instance, a web workload's request rate varies depending on the time of day and the occurrence of events [48]; a database workload can also change in terms of its composition of a wide variety of queries with different levels of CPU and I/O demands as illustrated in Chapter 3. Second, interference among VMs hosted on the same physical machine can lead to complex nonlinear resource usage behaviors as they compete for various types of resources that cannot be strictly partitioned. For example, when co-hosted VMs compete for the shared last level cache or disk I/O bandwidth, the relationship between each VM's resource allocation and its application's performance is known to be nonlinear [17][49]. Finally, even if the application workloads stay relatively steady, their SLAs, which specify the QoS that they require and the cost that they are willing to pay, may change over time. Consequently, resources in the system need to be reallocated across different applications' VMs in order to sustain the system-level objective. As more applications become Internet-scale and resources become more consolidated, the above scenarios would also be increasingly common in a virtualized system.

In particular, machine learning techniques can be employed to automatically learn the relationship between a VM's resource allocation and its application's performance; Control-theory techniques can be used to build a feedback loop into the resource management which can automatically adjust resource allocations and quickly reach the

desired system objective. This chapter proposes a new resource management approach based on the combination of these two types of techniques that can effectively capture the nonlinearly in virtualized system behaviors and quickly adapt to the changes in such behaviors, which are discussed in details in the following subsections.

4.1.2. Model Predictive Control

Model predictive control (MPC) [50] is an advanced control technique in which the controller takes control actions by optimizing an objective function that defines the objective of controlling the system. To enable the predictive capabilities of the control system, an explicit model that characterizes the system behaviors is leveraged to make predictions of system output over a specific future prediction horizon. Such modeling and optimization typically involved in MPC can be performed iteratively in an online fashion, where real-time data are used to update the model in the modeling phase and new optimal action is computed based on the model to adjust the system control. In this way, the system can adapt to the changes in the system behavior in a timely fashion.

In contrast to an open-loop optimal control technique, the MPC system works in a closed-loop manner by feeding back the information on previous inputs and outputs to the controller at the end of each control period in order to keep track of prediction errors and control variations, so that on one hand the controller is able to make more informative control actions based on the feedbacks, while on the other hand the system is able to be driven back to the set-point target appropriately without large oscillations even in the presence of noise.

MPC has been used by related work on VM resource management [52][51], where most approaches as the traditional feedback control methods do adopt “black box” linear input-output models which are accurate enough to model nonlinear system behaviors within a limited region of control operation.

Our proposed FMPC approach combines the strengths of machine-learning and control-theory techniques in virtual resource management. Compared to other modeling based approaches, the FMPC approach can be effectively applied online and quickly adapt to changes in system behaviors. Typical model-based approaches require substantial data for training the model which is difficult to do online. Even if a model can be built offline, it is difficult to adapt it online when the system behavior changes. Compared to other MPC-based approaches, the FMPC approach can well capture nonlinear system behavior without much learning overhead. In a typical linear-model-based MPC approach, a linear model is assumed to approximate the nonlinear behavior within a limited region of an operation point while it can be updated adaptively as the system moves from one operating point to another. However, as demonstrated by our experiment results, the FMPC approach can more accurately capture the system behavior with a nonlinear fuzzy model and it can perform optimized control continuously over the entire operating space.

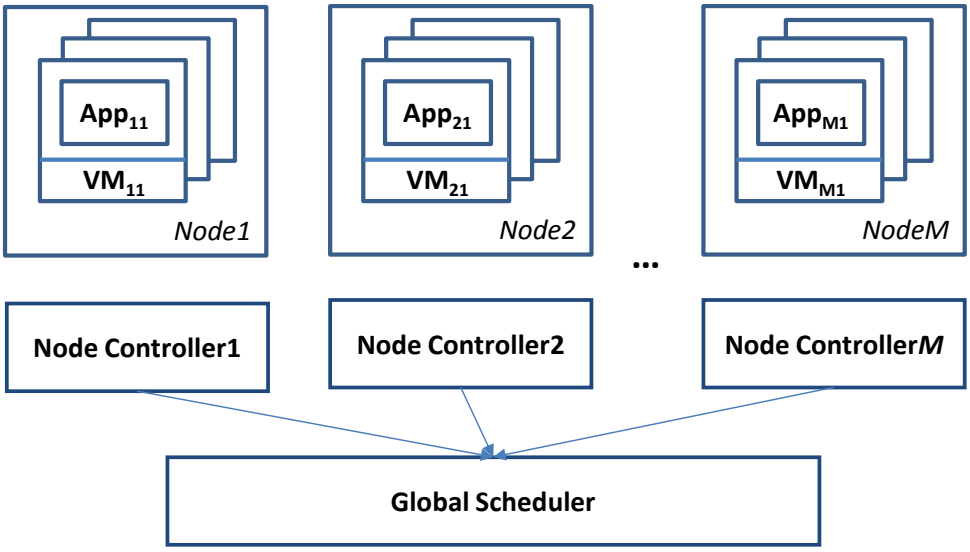

Figure 4-1 Two-level cloud resource management system

4.2. Two-level Resource Management Architecture

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

Figure 4-1 illustrates the architecture of the proposed two-level cloud resource management framework which includes a *Node Controller* on every VM host and a *Global Scheduler* for the entire cloud zone. Specifically, a node controller is responsible for

dynamically allocating resources to VMs and optimizing them using FMPC according to application QoS targets. The global scheduler dynamically adjusts VM placement through live migration in order to handle load variations on the VM hosts and to improve system-level performance. The node controllers and global scheduler cooperate with one another to complete the cloud resource management. When a node controller updates its predicted resource demands of its local VMs, it sends this information to the global scheduler for making VM migration decisions; when a global scheduler decides to migrate a VM, it coordinates with the node controllers on the source and destination hosts to update their performance models and adjust the resource allocations based on the new VM placement.

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

4.3. Host-level VM resource management

Figure 4-2 illustrates the architecture of our proposed system which consists of four key modules, *Application Sensors*, *Fuzzy Model Estimator*, *Optimizer*, and *Resource Allocator*. As the applications are running on their VMs, the *Application Sensors* monitor the performance $y_i(t)$ from each application *i* and then send them to *Fuzzy Model Estimator*. The estimator collects all necessary information including current and historical application performance and VM resource allocations to create the fuzzy model for performance prediction. Such a model which represents the relationship between the control input (resource allocations to the VMs) and the measured output (performance of the applications) is updated every control period. Based on the model, the *Optimizer* produces a resource allocation scheme for the next time interval that optimizes the system according to a predefined objective function. Then the *Resource Allocator* adjusts the VM's resource allocations accordingly. Together, these modules form a continuous feedback loop for the virtual resource management.

4.3.1. Fuzzy Model Estimator

The proposed FMPC is a fuzzy-model-based predictive control approach [50]. The major difference between FMPC and traditional MPC approaches lies in the modeling part. In FMPC, the fuzzy model estimator is responsible for building models that can describe complex system behaviors using fuzzy logic based method. The strength of this approach includes the following aspects: 1) it simplifies the learning of the complex models by describing nonlinearity using a set of linear sub models captured by the fuzzy rules; 2) it can perform optimized control over the entire operating space; 3) it inherits the benefits of

traditional predictive control that can guarantee dynamic performance in a closed-loop system and achieve desired target in a stable manner.

Consider a resource provider that hosts multiple applications by multiplexing multiple types of resources among them via VMs, a general *MIMO* model in MPC described by the following equation is used to build the time-varying relationship between resource allocations and application performance,

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

where the input vector $\mathbf{u}(t) = [u_1(t), u_2(t), \ldots, u_N(t)]^T$ represents the allocation of p types of controllable resources to the q applications' VMs at time step t ($N = pq$), and the output vector $\mathbf{y}(t) = [y_1(t), y_2(t), \ldots y_q(t)]^T$ is referred to as the predicted performance of q applications at time step t. For example, if there are two applications whose performance relies on two types of resources, i.e. CPU and disk I/O, then $\mathbf{u}(t)$ is a 4-dimensional vector, $[u_{CPU1}(t), u_{CPU2}(t), u_{IO1}(t), u_{IO2}(t)]^T$.

In traditional MPC approaches, linear models are applied to approximate the nonlinear behaviors around the current operating point, while m and n reflecting the impact of the previous inputs and outputs to current prediction are usually set to small values in order to reduce the complexity of the model, e.g., with $m = 0$, $n = 1$, $\boldsymbol{y}(t) = \Phi(\boldsymbol{u}(t), \boldsymbol{y}(t\text{-}1)) = \boldsymbol{a}\mathbf{u}(t) + \mathbf{b}\boldsymbol{y}(t\text{-}1)$.

In our proposed FMPC, the general Φ function from the control inputs to the system outputs is instantiated by a fuzzy model composed of a collection of Takagi-Sugeno fuzzy rules [31]

$\boldsymbol{R}^i$: $If\ \boldsymbol{u}(t)\ is\ \boldsymbol{A}^i\ and\ \boldsymbol{y}(t-1) is\ \boldsymbol{B}^i$,

$$then\ \boldsymbol{y}^i(t) = \boldsymbol{a_i u}(t) + \boldsymbol{b_i y}(t-1) \quad (1)$$

In the premise $\boldsymbol{A^i}$ and $\boldsymbol{B^i}$ are fuzzy sets associated with the fuzzy rule $\boldsymbol{R^i}$**.** Their corresponding Gaussian membership functions $\mu_{Ai}(\boldsymbol{u}) = e^{-\frac{(\boldsymbol{u}-c)^2}{2\sigma^2}}$ and $\mu_{Bi}(\boldsymbol{y}) = e^{-\frac{(\boldsymbol{y}-c)^2}{2\sigma^2}}$ determine the membership grades of the control input vectors $\boldsymbol{u}(t)$ and $\boldsymbol{y}(t\text{-}1)$, respectively, which indicate the degree that they belong to the fuzzy sets. In the consequence, the output $\boldsymbol{y}(t)$ is a linear function of the current control input and the previous output with trainable parameter matrices $\boldsymbol{a_i}$ and $\boldsymbol{b_i}$.

The *Estimator* adopts an efficient one-pass clustering algorithm, subtractive clustering, to build a concise rule base with a small number of fuzzy rules that can effectively represent the VMs' behaviors. Each cluster exemplifies a representative characteristic of the system behaviors and can be used to create a fuzzy rule accordingly. In this way, both the system structure and parameters are learned and adapted in real time from online data streams. The system model gradually evolves as opposed to having a fixed structure model, and the learning process is incremental and automatic. Owing to the speed of subtractive clustering and fuzzy modeling, this whole model updating process can be completed quickly within a fine-grained control interval.

The *Estimator* is invoked by the *Optimizer* discussed below in every control step *t* to predict the performance for specific input values and assist it to search for the optimal allocation solution across the input space. The *Estimator* applies *fuzzy inference* to predict the output $\boldsymbol{y}(t)$ for a given control input < $\boldsymbol{u}(t)$, $\boldsymbol{y}(t\text{-}1)$ > based on a trained fuzzy rule base with *S* fuzzy rules. It entails the following steps: 1) Evaluation of antecedents: the input variables are *fuzzified* to the degree, $\boldsymbol{\delta^i}$, to which they belong to each of the fuzzy sets via

the corresponding membership functions for each fuzzy rule $\boldsymbol{R}^i$;2) Implication to consequents: implication is performed on each fuzzy rule by computing $\boldsymbol{y}^i(t)$ based on the equation in the consequent of the rule; 3) Aggregation of consequents: the final prediction is performed as $\boldsymbol{y}(t) = \sum_{i=1}^{S} \boldsymbol{\delta}^i \boldsymbol{y}^i(t)$, where the outputs $\boldsymbol{y}^i(t)$ of all the fuzzy rules are aggregated into a single numeric value based on their corresponding membership grades $\boldsymbol{\delta}^i$.

4.3.2. Optimizer

Generally, the objective function in MPC can be formulated as

$$J(t) = \sum_{i=1}^{P} \|\Delta \boldsymbol{y}(t+i|t)\|^2 \, Q(i) + \sum_{i=0}^{M-1} \|\Delta \boldsymbol{u}(t+i|t)\|^2 \, R(i) \quad (2)$$

where P and M indicate the prediction and control horizon. $\boldsymbol{\Delta y}$ is the *predictive error* between $\boldsymbol{y}(t+i)$, the output of the next ith step predicted from the current time step t (using the fuzzy model produced by the Estimator), and the reference output $\boldsymbol{y}_{\boldsymbol{ref}}(t+i)$ of the next ith step. $\boldsymbol{\Delta u}$ indicates the *control effort*. The importance of tracking accuracy in performance targeting and maintaining stability in control operation can be determined by tuning the $Q(i)$ and $R(i)$ factors for the two components of the equation. Larger Q factor will make the controller react aggressively to tracking errors in performance. Larger R factor will guarantee the stability of the system by preventing from large oscillation in the resulting resource allocation, but lead to slower response to the tracking error.

To reduce the complexity of the problem, we choose an objective function with $M = P = 1$. In addition, in Equation 2, the performance of the q different applications, represented in $\boldsymbol{y} = [y_1(t), y_2(t), \ldots y_q(t)]^T$, are treated with equal importance. In practice, applications

concurrently hosted in a virtualized datacenter or cloud are often given different preferences, because they have different priorities or they generate different amounts of revenue to the system. Without loss of generality, we use a weight vector $\boldsymbol{w} = [w_1(t), w_2(t), \ldots w_q(t)]^T$ to represent the preferences given to the applications. The following objective function formulated as a constrained minimization problem considers not only tracking QoS targets for individual applications but also optimizing resource allocations for maximizing the system-level benefit especially when resources are contested.

$$J(t) = Q\left\|\boldsymbol{w}\cdot\left(\boldsymbol{y}(t+1) - \boldsymbol{y_{ref}}\right)\right\|^2 + R\left\|\left(\boldsymbol{u}(t+1) - \boldsymbol{u}(t)\right)\right\|^2$$

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

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

The fuzzy performance model in FMPC is rule-based and not differentiable; a minimization problem involving such models cannot be solved by any classical, derivative-based optimization algorithm. A genetic algorithm (GA) method is applied to solve this complex optimization problem [47]. This algorithm is well-known for tackling more general

optimization problems in which the objective function is non-differentiable, discontinuous or highly non-linear, that are not well suited for standard optimization algorithm, e.g., quadratic or linear programming. In light of the natural selection process in biological evolution, the GA algorithm encodes a solution in the optimization search space as a gene in biological reproduction. By mimicking the gene combinations in biological reproduction, it iteratively operates on a population of candidate solutions as a parent generation to produce its children generation by selecting the good parent candidates and performing randomly genetic operations (mutation and crossover) on them to produce the children for the next generation. The goodness of each candidate solution is computed by a predefined fitness function which is usually related to the objective function in optimization. Finally, the population "evolves" toward a globally optimal solution over successive generations.

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

As described above, the Estimator and Optimizer work together in an online closed-loop. The input-output data pair <u(t), y(t)> is measured and collected in every control period to

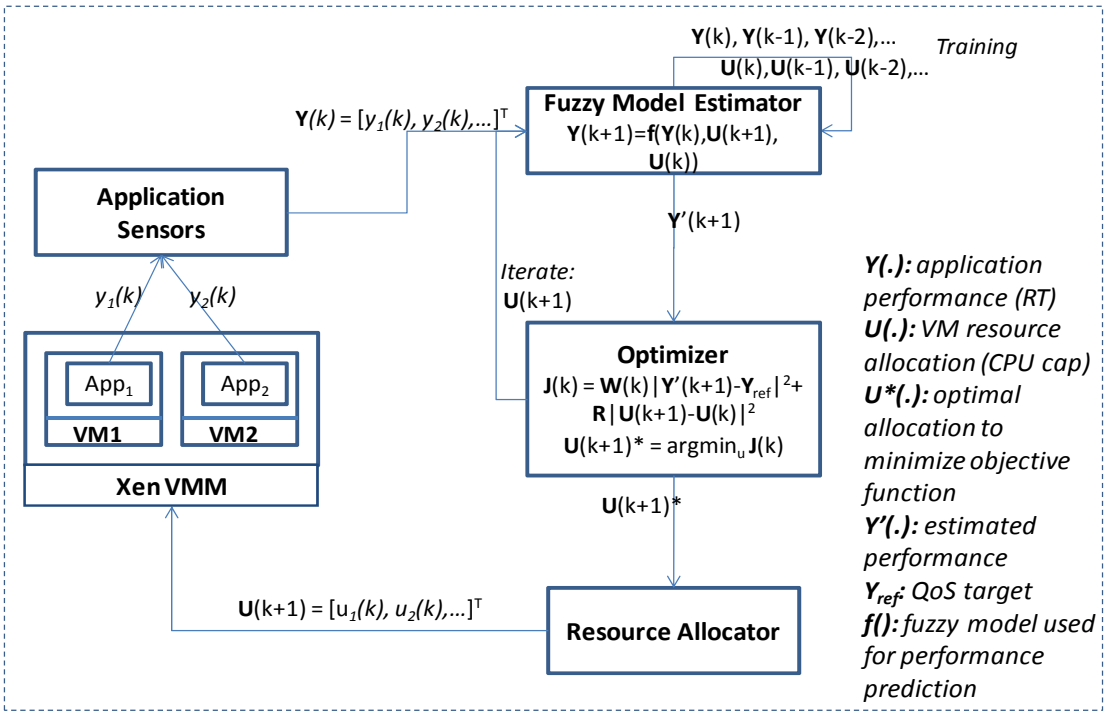

Figure 4-2 The architecture of the FMPC local controller system

train the fuzzy model. A MIMO fuzzy model can handle a coupled system with multi-input and multi-output to describe complex system behavior with implicitly contentions from system components. Once the model is established, it serves as a prediction tool for the controller to search for the optimal u(t+1) that promises the best y(t+1) which will be applied to the VM resource allocation in the next control period. As shown in the evaluation section, this control loop can be applied at fine time granularity (e.g., 20s) to meet QoS targets. It is capable of quickly recovering from model inaccuracy (during bootstrapping or dynamic changes in the system), as the observed performance for a given allocation is immediately used to update the model and reflect the current behaviors.

4.4. Cross-Host Cloud Resource Management

Within a host's resource constraints, the FMPC approach allows the node controller to effectively optimize the host-level performance objective by allocating the resources to its local VMs. However, local optimality achieved at individual host level does not guarantee the global optimality in the entire zone because resource utilization may be unbalanced across the hosts. The global scheduler in the proposed two-level cloud resource management architecture addresses this issue and optimizes the zone-level resource utilizations by live-migrating VMs across the hosts. There is a good amount of related work on the use of VM migration to optimize for a variety of performance, energy, and thermal objectives (e.g., [64][65]). The global scheduler in the proposed two-level cloud resource management architecture focuses on the use of VM migration for cross-host load balancing and its integration with the FMPC-based node controllers.

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

The global scheduler detects these conditions on its managed hosts based on the VM resource demands estimated by the FMPC controllers of their node managers. It then uses the information to carefully make migration decisions for the entire system. The global scheduler continuously updates two lists based on the resource demands periodically collected from the node controllers: *OutList*, the list of overloaded nodes which satisfy the migration condition and need to move out some of its hosted VMs; *InList*, the list of

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

When a VM is migrated, it needs to be removed from the source host's fuzzy MIMO performance model and added to the destination host's MIMO model. If the migrating VM's performance model has to be retrained from scratch, it would have a considerable adversarial impact on its performance as well as the performance of the other co-hosted VMs. To minimize this impact, the node controllers on the source and destination hosts work together and transfer the migrating VM's performance model from the source host and use it to bootstrap its model on the destination host. To facilitate this model transfer, the MIMO model is decomposed into a set of single-input-single-output (SISO) fuzzy models, so that the migrating VM's model can be extracted and transferred. Note that there will be inaccuracy when predicting the VM's performance using the transferred model because the other co-hosted VMs, which also affect the migrating VM's performance, change after the

migration. But this inaccuracy will be corrected by the Fuzzy Model Estimator which continuously updates the model online.

```
RunScheduler ( ) {
  Initialize ( );
  for (;;) {
    Update(D); Update(Avail);
    Update(Out_list);
    Update(In_list);
    for each node i in Out_list {
      for each VM_ij ϵ D_i {
        if (∃k ∈ In_list | Avail_k ≥ Res_ij){
          Migration_list ← {< VM_ij , i, k>};
          Avail_i += Res_ij; Avail_k -= Res_ij;
          if (Avail_k ≤ Threshold){
                In_list -= {k};
          }
          if (Avail_i ≥ 0){
                Out_list -= {i};
                break;
          }
        }
      }
    }
    DoMigration(Migration_list);
    Wait till next migration interval;
  }
}
```

```
Initialize ( ){
   for each node i {
      Avail_i = C_i - ∑_{VM_ij ϵ D_i} Res_ij
      if (Avail_i ≥ Threshold)
         In_list ← {i}
      else if (Avail_i ≤ 0 )
         Out_list ← {i}
   }
   Sort(In_list);
  Sort(Out_list);
}
```

Pseudo code for the migration scheduling

4.5. Evaluation

4.5.1. Setup

This section evaluates the proposed FMPC-based two-level cloud resource management using representative benchmarks in a typical virtualized environment. The testbed is a cluster of Dell PowerEdge 2970 servers, each equipped with two six-core 2.4GHz AMD Opteron CPUs, 32GB of RAM, and 1TB SAS storage. Xen 3.3.1 is installed to provide the VMs, and the guest operating system is Ubuntu Linux 8.10 with paravirtualized kernel 2.6.18.8.

To evaluate the FMPC approach's accuracy and adaptability for modeling the complex behaviors of such a multi-tiered application as a black box, the web and database tiers of a RUBiS instance are deployed on the same DomU VM using Apache Tomcat 4.1.40 and MySQL 5.0. The resource allocation to a RUBiS VM is dynamically controlled by the FMPC-based node controller. The client VMs, which generate workloads to the RUBiS VMs, are hosted on separate physical machines and they can launch up to 8000 emulated client sessions in total. To create high CPU contentions, another benchmark, FreeBench[55], which models computationally intensive jobs, was also used in the experiments.

Because these benchmarks cannot saturate the storage bandwidth, the evaluation focuses on the management of CPU resources. Nonetheless, the previous work studied the use of fuzzy modeling to estimate the demands of both CPU and IO resources and showed its significant advantage in accuracy over a linear modeling approach [60].

The control period of the node controllers is 20 seconds, during which a controller updates its local VMs performance model and optimizes the resource allocations to the VMs. The control period of the global scheduler is one minute, during which it gathers the resource demands from all the node controllers, decides the VM migrations, and coordinates the involved node controllers to execute the migrations.

The rest of this section presents the evaluation results. It first evaluates the FMPC approach's ability to correctly estimate an application resource demands and consistently meet its QoS target while servicing a dynamic workload. It then evaluates the node controller's ability to optimize the resource allocations to multiple VMs at the host level. Finally, it evaluates the global scheduler's ability to improve system-level performance by coordinating with the node controllers and dynamically migrating VMs across hosts.

4.5.2. Application-Level Target Tracking

The first group of experiment evaluates the ability of the FMPC-based controller in tracking fine-grained QoS target for a multi-tiered application (RUBiS) that services a dynamic workload.

The experiment compares the proposed FMPC approach to the adaptive linear MPC (LMPC) approach studied in the related work[12]. In the FMPC approach, the predicted performance is assumed to be dependent on only the current resource allocation, so Equation (1) is simplified as $\boldsymbol{R^i}$: $If\ \boldsymbol{u}(t)\ is\ \boldsymbol{A}_i,\ then \boldsymbol{y^i}(t) = \boldsymbol{a_i u}(t) + \boldsymbol{b_i}$. In Equation (3), both the input and output vectors $\boldsymbol{u}$ and $\boldsymbol{y}$ are normalized by their maximum values that the system can achieve; and the Q and R factor are both set to 1 to balance the importance between tracking accuracy and controlling stability. The baseline LMPC leverages a linear

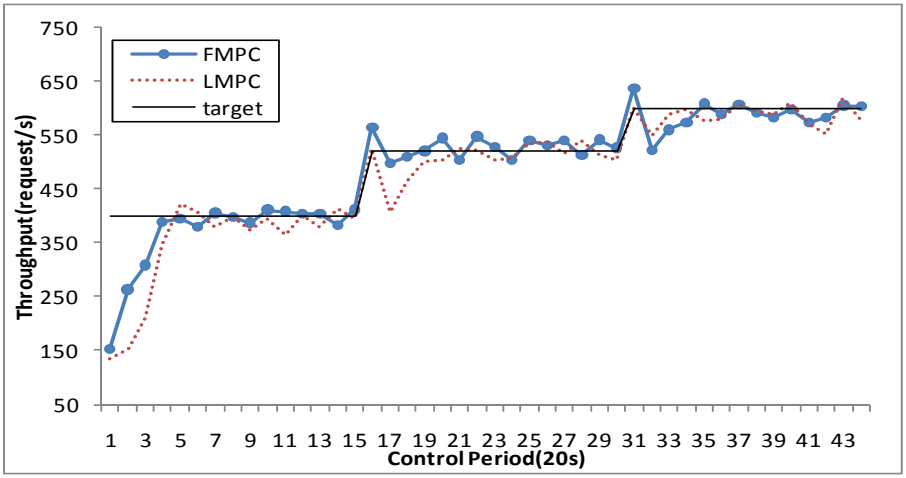

Figure 4-3 Performance for bursty RUBiS workload

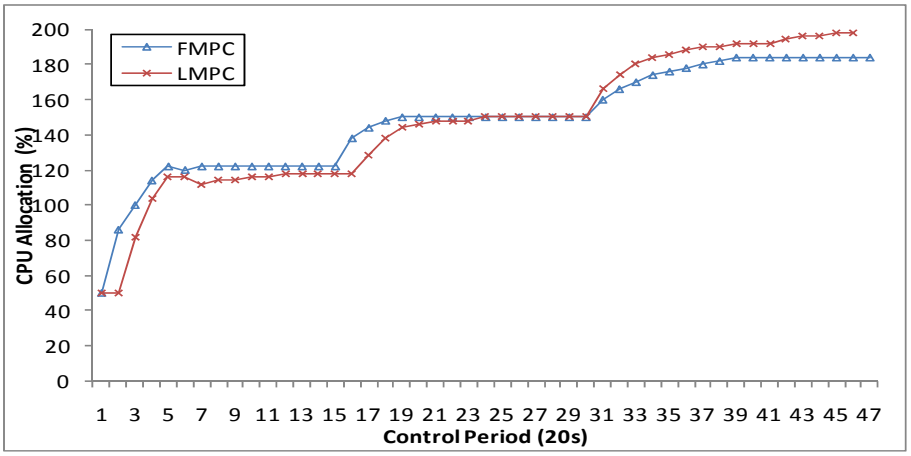

Figure 4-4 CPU allocation for bursty RUBiS workload

auto-regressive-moving-average (ARMA) model which automatically trains the linear VM performance model using the recursive least squares method [57] and is able to adapt the model based on the online training. For both approaches, once the workload is launched, the controller starts with an initial resource allocation that is much less than the actual demand. The model is created from scratch with the first few data points and afterwards it is updated every control interval.

a) Bursty Workload with Throughput-Based Target

First, we evaluate the robustness of our FMPC controller under a bursty RUBiS workload with abrupt fluctuations in the workload intensity within short period, i.e., the number of concurrent client sessions changes from 2400 to 3200 then to 4000. Each phase is kept for 15 control intervals (300s) before an immediate transition (within one control period) to the next one. The corresponding throughput targets for each phase are set to 400, 500 and 600 requests/s respectively. The fuzzy model adapts as those large stepped changes in workload: during the first phase, only 1 fuzzy rule is established in the rule base; by the end of the experiment, 2 rules are trained.

Figure 4-3 shows the performance (throughput in requests per second) of RUBiS measured every control interval, from using our proposed FMPC approach to manage the VM resources versus using the LMPC approach. As we can see both approaches are able to track the changes in the workload at periods 15 and 30 and meet the specified QoS targets pretty closely. However, FMPC outperforms LMPC in several important aspects. First, the FMPC based approach is more accurate in meeting the specified QoS target. The average *steady state error* throughout all three phases is 2.3% for FMPC and 2.9% for LMPC; particularly in the third phase, the steady state error is 1.7% for FMPC 3.3% for LMPC.

Second, the performance controlled by FMPC adapts faster than LMPC when a step change occurs in the workload intensity. The average *settling time* to within 5% of the steady state for all three phases is 3 control intervals in FMPC and 5 intervals in LMPC, where in each phase FMPC is 1 to 2 intervals faster than LMPC in settling time. This

advantage is because that FMPC's fuzzy modeling is more accurate than LMPC's linear modeling when transition happens. Owing to the flexibility of FMPC, it tunes its model more adaptively than LMPC does. For example, instead of being restricted by a fixed linear shape mode of LMPC, FMPC can immediately add a new rule as soon as new data comes which cannot be fit into current model. As a result, LMPC suffers from more than 20% tracking error ($1-y/y_{ref}$) when the first transition occurs, whereas in FMPC there is almost no tracking error. Overall, the average of the performance across all three phases using FMPC is about 5% higher than using LMPC approach.

To better analyze the results, Figure 4-4 shows the corresponding CPU allocations. With an initial CPU allocation of 50% the FMPC controller is able to detect resource under-provision as soon as the first target miss is observed and converge to an optimal allocation for meeting the target within a few control intervals. In comparison, the LMPC acts at least one interval slower than FMPC in the first phase and two intervals slower in the second phase. In the third phase, the LMPC approach also allocates 14% more CPU than the FMPC approach. Such over provisioning could lead to loss of performance for other co-hosted VMs and loss of revenue for the entire virtualized system.

b) Realistic Workload with Response-Time-Based Target

In the second experiment, we evaluate the capability of the FMPC controller in tracking the response-time-based QoS target which is more sensitive to the accuracy in resource allocation. 90^{th}-percentile response time is used as the performance metric since it is more reliable to reflect the Internet service quality [63]. However, it is also more challenging for solving a control problem due to its highly non-linear relation in the performance modeling.

To make the RUBiS workload more realistic, the number of concurrent client session is varied in a more random way by following a real daily trace collected from the production web server of CS department in FIU [56]. We collect the number of requests per hour in a daily trace and vertically scale the range of the request rate to the range that our RUBiS setup can handle (Figure 4-5). To speed up the replay of the trace, we keep it running for 200s to simulate one-hour duration in the real trace so that the duration of the workload is scaled from 24 hours to 2880 seconds. The experiment starts with an initial model pre-trained for the workload with 200 client sessions. As the workload varies, the model is adapted online every control period. The QoS target for this RUBiS workload is set to 20ms 90^{th}-percentile response time, which can be achieved under sufficient resource allocation.

Figure 4-6 and Figure 4-7 show the performance measurement and CPU allocations every control interval, from using our proposed FMPC approach to manage the VM resources versus using the LMPC approach. As we can see although both approaches are able to track the performance target eventually as workload changes, FMPC is able to meet the QoS target more closely and more responsive to the changes especially when the system is heavily loaded (from time 1600s to 1800s); while the LMPC suffers more fluctuations in performance than FMPC does during the same time period. This is mainly because FMPC can capture more accurately than LMPC the highly nonlinearity in a heavy-loaded system with respect to the percentile-based performance metric. The better accuracy in learning non-linear percentile-based performance model and its fast online learning algorithm allows FMPC to adapt more quickly under the highly dynamic workload and converge to steady state with less fluctuations in system.

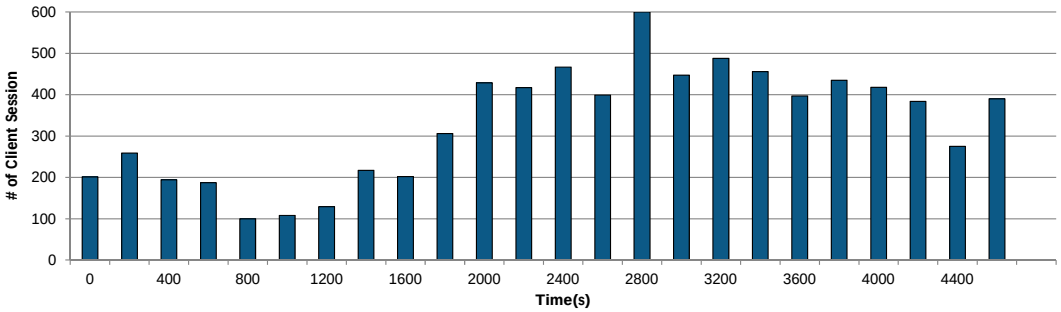

Figure 4-5 A real trace replayed in RUBiS browsing mix

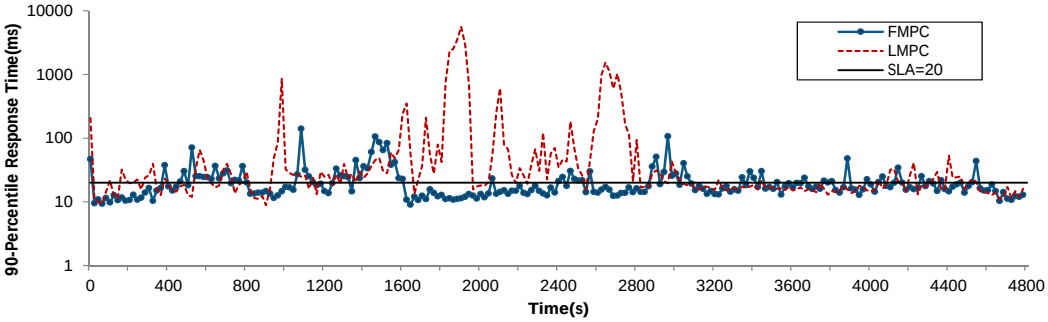

Figure 4-6 Performance for realistic RUBiS browsing mix

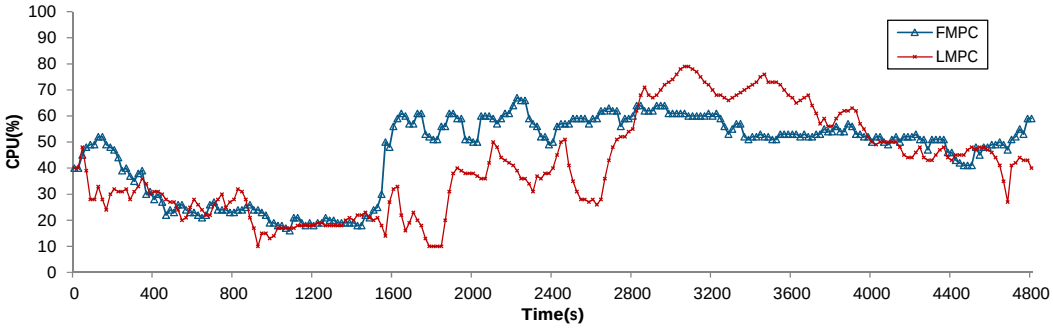

Figure 4-7 CPU allocations for realistic RUBiS browsing mix

In summary, the proposed FMPC controller can automatically track the reference QoS for an application by allocating the proper amount of resources to its VM. It also outperforms LMPC in terms of the adaptively and accuracy.

4.5.3. Host-Level Resource Management

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

a) Fixed Workloads with Changing Weights

In the first experiment, we evaluate whether the proposed FMPC approach can always achieve optimal total revenue where application SLAs change over time and how quickly it adapts to such dynamic changes by hosting two RUBiS VMs on the same pair of physical cores and varying their priorities during the execution.

To make it more interesting, we create scenario where interference exists between the two VMs. By experimenting with the RUBiS workload, we notice that having 2400 concurrent users for one VM-hosted RUBiS application would create a total CPU demand of 100% on the single dual-virtual-CPU VM which hosts both the web and database tiers of RUBiS. However, if we run two independent RUBiS VMs concurrently and host both VMs on the same pair of physical cores (using CPU affinity), then neither VM can achieve the same level performance when serving the same workload even though each of them can still get 100% of CPU. This observation confirms the existence of performance interference across VMs which commonly exists on a highly consolidated virtualized system.

To capture the behaviors for the entire system, including the individual VM performances as well as the coupling relation among them, we use a two-input-two-output

FMPC to control the resource allocations to the two VMs. The input variables are the CPU allocations to the two VMs and the outputs are the measured performance of the two RUBiS applications. As discussed in Section 4.3.2, we assign different weights w_1 and w_2, to the two VMs ($w_1 + w_2 = 1$), which represent the different priorities or impacts to revenue as determined by the application SLAs. So the objective function is:

$$J(t) = \left[w_1(t) * \left(y_{ref1} - y_1 \right)^2 + w_2(t) * \left(y_{ref2} - y_2 \right)^2 \right] + \left| \Delta \bar{u}(t) \right|^2$$

where $\boldsymbol{u} = [u_1, u_2]^T$ denotes the CPU caps set to the two VMs. Since they share the same two physical cores, the total available CPU is 200%. The workload intensity for each VM is fixed to 2400 client sessions. The QoS target y_{refi} is set to 400 request/s for both RUBiS instances, which is the performance that it can achieve with 100% CPU and no interference.

Figure 4-8 shows the CPU allocations to both application VMs made by our FMPC controller in the experiment. Initially, both VMs have equal CPU shares. In the first phase, *VM1* got more CPU resource (around 140%) than *VM2* (around 60%) because the former has a higher weight. Starting from the interval 16, as the weights change to 1:1, u_1 decreases and u_2 increases, both quickly converging to 100% of CPU as expected. During the third phase, *VM1* is assigned less CPU (around 60%) than *VM2* (around 140%) because VM2 now has a higher weight. Interesting, when one VM's weight is set to three times of the other one, it does not get three times of resource allocation, because of the nonlinear relationship between VM resource allocation and application QoS.

To demonstrate the effectiveness of the FMPC-based resource management, we compare it with the LMPC-based approach and another weight-based scheme which

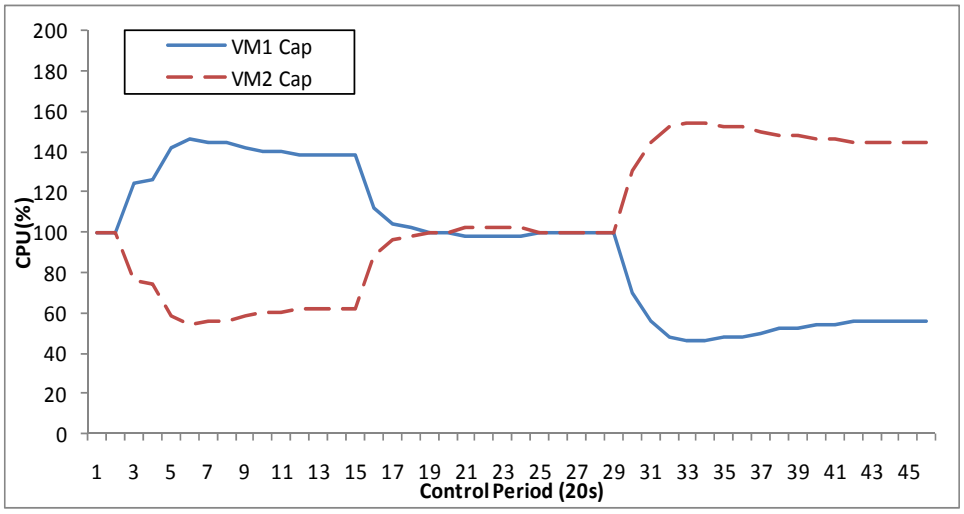

Figure 4-8 CPU allocations for interfering VMs

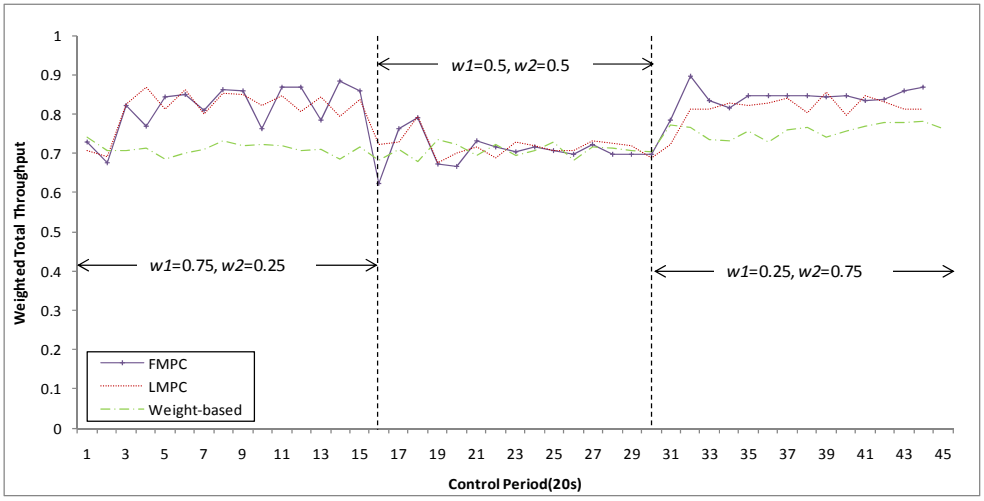

Figure 4-9 Weighted total throughput of interfering VMs

intuitively partitions the total resource to VMs based on their assigned weights (i.e., the CPU caps are set to 3:1, 1:1 and 1:3 for VM1:VM2 across the three phases.). The *weighted total throughput* that is aggregated by the weighted throughputs from all applications in the system is used as the performance metric for host-level objective in this experiment. The results in Figure 4-9 illustrate that the allocation decisions made by the FMPC controller substantially outperform the weight-based scheme across all three phases.

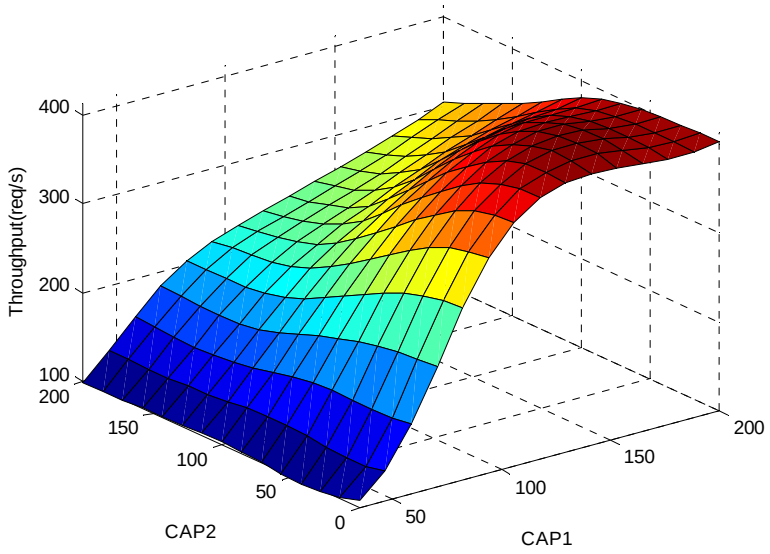

Figure 4-10 The 3-D fuzzy model for *VM1*

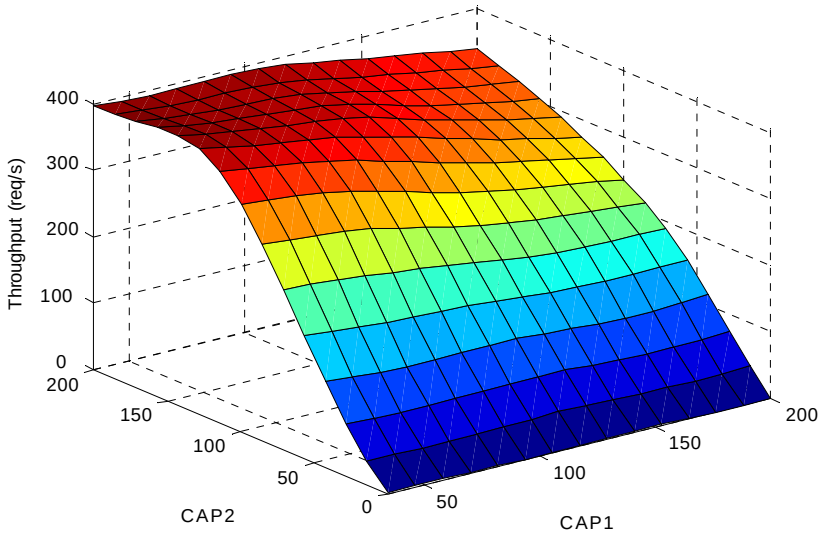

Figure 4-11 The 3-D fuzzy model for *VM2*

During the first two phases, LMPC works as well as FMPC. However, in the third phase, FMPC generates about 4.7% more throughput in average than LMPC does. From the results, we can see that FMPC can achieve higher weighted total throughput, particularly in the first and third phases. Nonetheless, the FMPC approach can correctly capture these nonlinear behaviors and produce much better resource allocations.

To further understand the impact of interference on VM performance, we use fuzzy modeling to build a global two-input two-output non-linear model given the entire input space for the two competing RUBiS VMs, where the two control inputs are the CPU allocations to the VMs and the two control outputs are the measured performance for the individual RUBiS instances hosted on the VMs. The model is created in the following way: while keeping the workloads concurrently running against the two VMs, the CPU cap set to each VM is varied from 0% to 200%. The model is trained offline based on a total of 350 data points collected from a set of evenly distributed cap values in this range. Each data point is 4-element tuple $< cap_1, cap_2, y_1, y_2 >$. The fitting error is 7.4%.

For better illustration, we split this model into two 3-D models and illustrate them separately in Figure 4-10 and Figure 4-11 each representing the behavior of one VM under the interference from the other. From the models, we can see that for each application, the performance is not only dependent on the CPU allocation to its hosting VM but also affected by the CPU cap set to the other VM. With the same value of cap set to one VM, its application's performance will drop as the cap value of the other VM increases. Nonetheless, the fuzzy logic based modeling technique is able to capture more complex relationship between resource allocation and performance with the presence of interference resulted from co-hosted VMs.

b) Changing Workloads with Changing Weights

In the second experiment, we evaluate our FMPC controller on larger-scale virtualized system which hosts a mix types of application workloads, in which both the workloads and the applications' weights change dynamically.

Two different benchmarks are used in this experiment which are RUBiS and Freebench[55]. A total of 12 VMs are pinned on the same 3 pairs of physical cores, each configured with 1 virtual CPU and 1G RAM serving different types of application workloads. 8 of them are deployed with the multi-tier RUBiS setup consisting of web and database tiers and the other 4 VMs are deployed with Freebench. The entire experiment lasts for 1200s, all the RUBiS VMs is performed with the same browsing mix trace with varied intensity as illustrated in Figure 4-12 while all Freebench VMs are kept busy serving continuous requests as long as the RUBiS workloads last. We assign different weights for different applications, denoted as w_R and w_F for RUBiS and Freebench respectively. Those weights is varied as well as the workload as showed in Figure 4-12. The VMs that host the same application are treated equally. 90^{th}-percentile response time and average response time are used as performance metrics for RUBiS and Freebench. The QoS target is set to 20ms for the former and 0.8s for the latter. To make the performance of different applications comparable, the real-time performance measurement is normalized into the same magnitude by dividing its target value.

The experiment can be divided into three phases according to the weight values, i.e., $< w_R, w_F > = <0.25, 0.75>, <0.5, 0.5>, <0.75, 0.25>$, for each phase the workload intensity of RUBiS VMs increases from 300 to 400 client sessions. The total capacity in the system

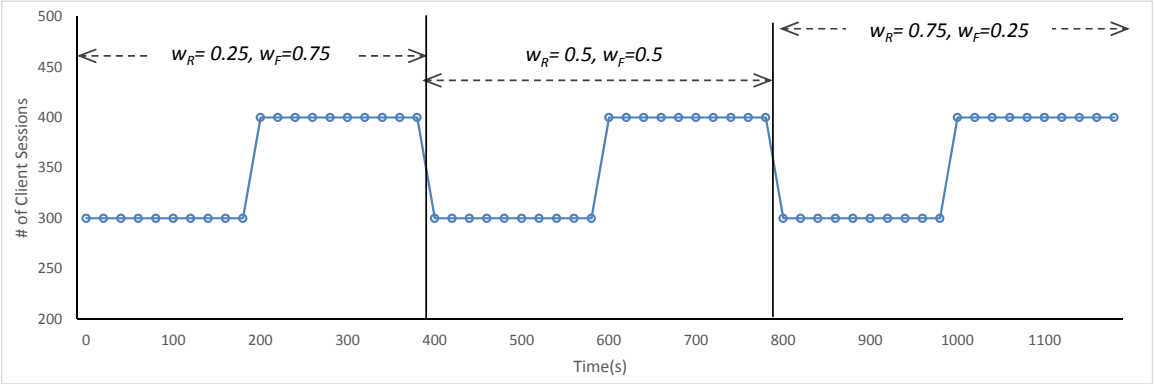

Figure 4-12 Changing workload for RUBiS VMs with changing weights

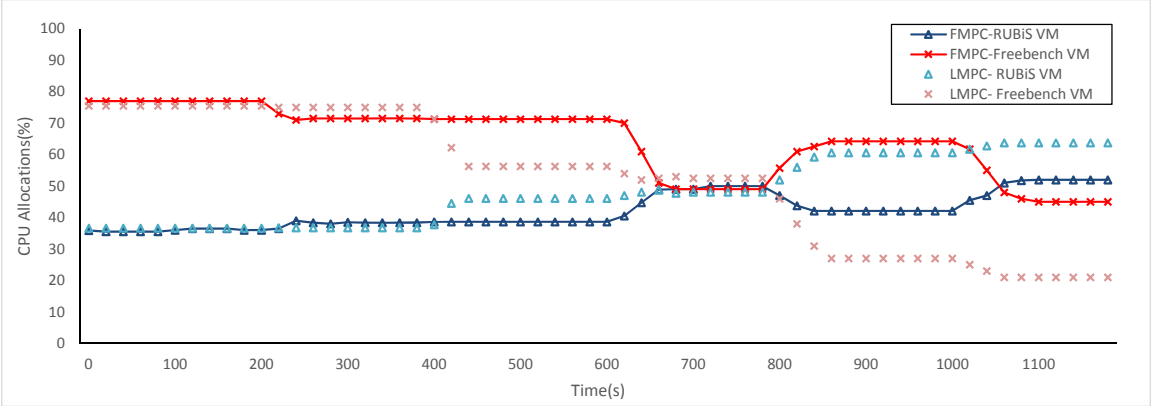

Figure 4-13 Average CPU allocations for each group of VMs

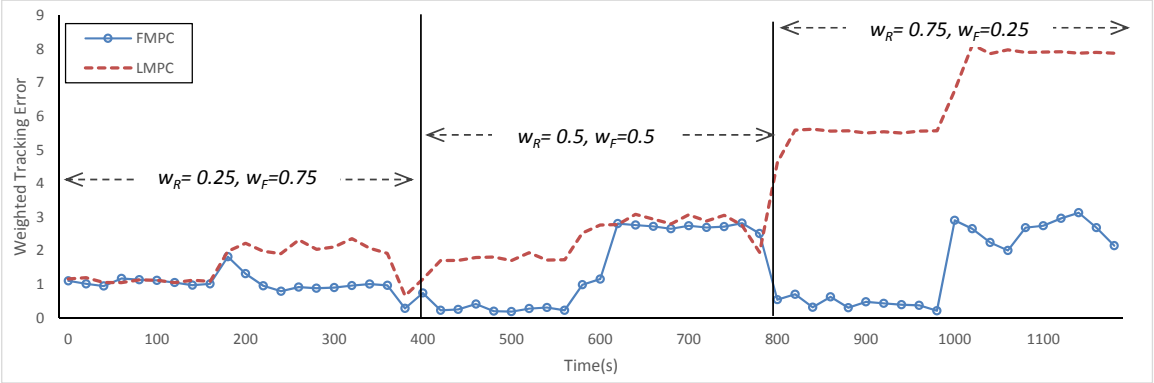

Figure 4-14 Weighted performance error for all VMs

is limited to 6*100% CPU. Both FMPC and LMPC approaches are compared in managing all 12 VMs at the same time to optimizing the overall system performance. The experiment can be then divided into three phases according to the weight values, and for

each phase the workload intensity of RUBiS VMs increases from 300 to 400 client sessions. The QoS target is set to 20ms response time for RUBiS and 0.8s loop time for FreeBench. To make the performance of different applications comparable, the actual performance measurement is normalized into the same range.

Figure 4-13 compares the online resource allocations made by FMPC vs. LMPC. For simplicity, the average value of CPU allocations to VMs that run the same application is shown for every control interval. Figure 4-14 compares the weighted sum of the normalized performance errors, $\|\boldsymbol{w}(\boldsymbol{y}(t+1) - \boldsymbol{y_{ref}})\|$, achieved by FMPC and LMPC. This metric reflects the total performance discrepancy from the QoS target vector $\boldsymbol{y_{ref}}$, which should be minimized by the controller in a steady state.

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

because their higher weight. As the workload intensifies for the RUBiS VMs, FMPC increases their allocations which eventually exceed the FreeBench VMs, whereas LMPC continues to favor the FreeBench VMs. This opposite decision causes LMPC to perform substantially worse (up to 11 times higher performance errors) than FMPC.

c) Realistic Workloads

The third experiment evaluates both the scalability and stability of the proposed FMPC approach in managing more VMs under realistic workloads with more dynamic changes. In this experiment, eight VMs share four physical CPU cores, and they all run RUBiS using the same real-world web trace described in Section 4.5.2.c). To make the experiment more interesting, the VMs are divided into four groups, and each group starts the replay from a different offset of the trace, as shown in Figure 4-15. As a result, the four groups reach their peaks and values at different times in the experiment, and the total load of the VMs also varies over time. In this experiment, equal weight and QoS target (15ms) are set for all the VMs. Note that when the system is saturated, none of the VMs can meet its QoS target under equal resource allocations. However, this experiment focuses on how to optimize the overall performance by minimizing the distance to the VMs' QoS targets.

Figure 4-16 and Figure 4-17 compare the CPU allocations made by FMPC and LMPC. For better clarity, the figures show the average allocations to each group of VMs. All VMs start with equal resource allocations. The difference between FMPC and LMPC appears from the 600th second when FMPC allocates an average of 9.8% more CPU than LMPC to the VMs in Group 2 as the intensity of their workloads dominates over the other three. When the system's total load is around its peak (1400-2200s), FMPC favors the VMs in

Groups 1 and 3 even more than LMPC because their higher demands than the other two groups. Then, when the workloads of all the other groups are decreasing (2400-2800s), FMPC allocates more CPU to Group 4, which is still at its peak, than LMPC.

Figure 4-18 compares the overall performance achieved by FMPC vs. LMPC using the average 90th-percentile response time as the metric because all the VMs have the same QoS target and weight. At the beginning and the end of the experiment, the overall system load is low and as a result there is not much difference in performance between FMPC and LMPC. But when the system is more loaded, FMPC outperforms LMPC significantly. For example, from 1000s to 1600s, while LMPC achieves an average response time of 29.2ms and causes serious QoS violations (w.r.t. the 15ms target), FMPC still maintains a good performance (17ms average response time). From 1800s to 2600s, when the system is saturated, FMPC delivers a 15.6% better overall performance in average response time than LMPC. From 2400-2800s, as all the workloads decrease, FMPC allocates a higher CPU allocation (64.4%) to *Group* 4 than the remaining ones due to its larger ratio in the total workload; LMPC only allocates an average of 55.2% CPU to the same group.

To demonstrate the effectiveness of the controller, the *weighted 90th-percentile response time*, the mean of the 90th-percentile response time measurements from all RUBiS VMs in the system is used as the host-level performance metric. Note that mean value is used due to equal weights to all VMs. The performance comparison in Figure 4-18 shows that the FMPC controller outperform the LMPC in coordinating multiple VMs to achieve better host-level performance. For the most of time especially when the total workload in the system is not so resource-intensive, e.g., for the first and the last 600s, LMPC works as

well as FMPC. However, for some time periods, more significant performance degradation observed in LMPC than in FMPC; from time 1000s to 1600s, while LMPC suffers from serious overall QoS violations, an average of 29.2ms in weighted response time, due to the rising intensity in total workload, while FMPC can still maintain the weighted response time as good as 17ms; from time 1800 to 2600s, while none of the VMs can obtain sufficient resource in both approaches since the system is highly overloaded and the total CPU amount needed is far from the host capacity, FMPC delivers a 15.6% better overall performance in weighted response time compared to LMPC. According to the above observation, FMPC is proven to be able to provide better allocation solutions to optimize system performance in a more realistic scenario where multiple long workloads competing for the limited amount of resources.

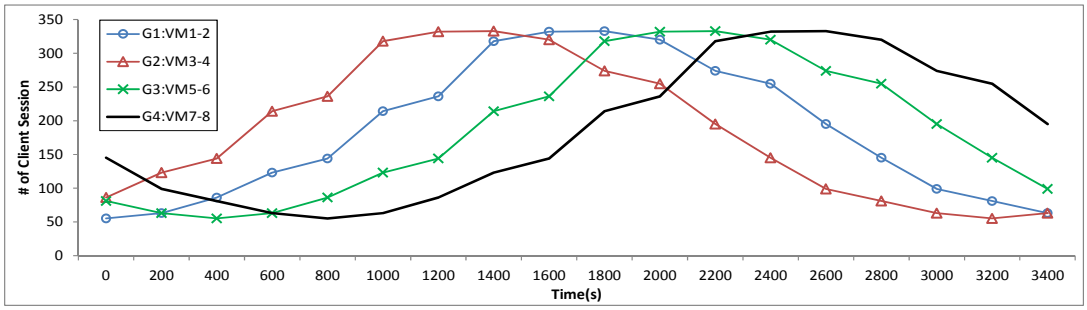

Figure 4-15 The workload trace for all 8 VMs

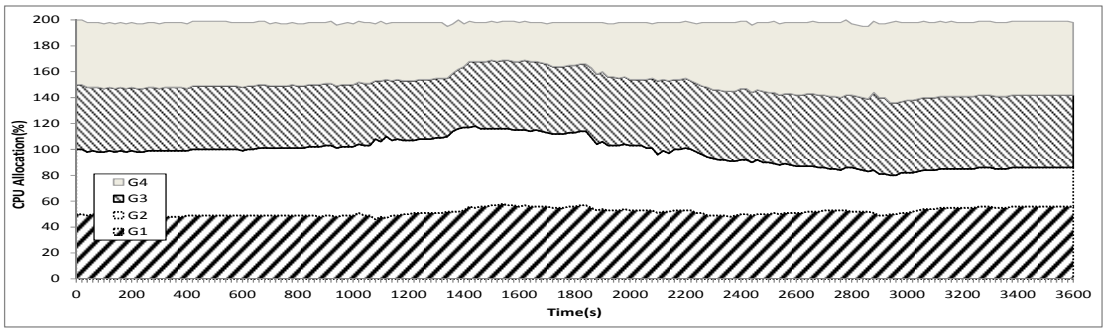

Figure 4-16 CPU allocation in LMPC

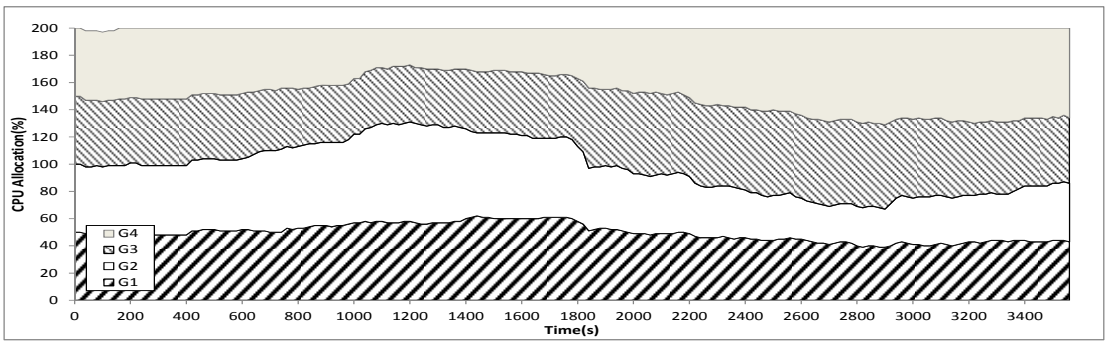

Figure 4-17 CPU allocation in FMPC

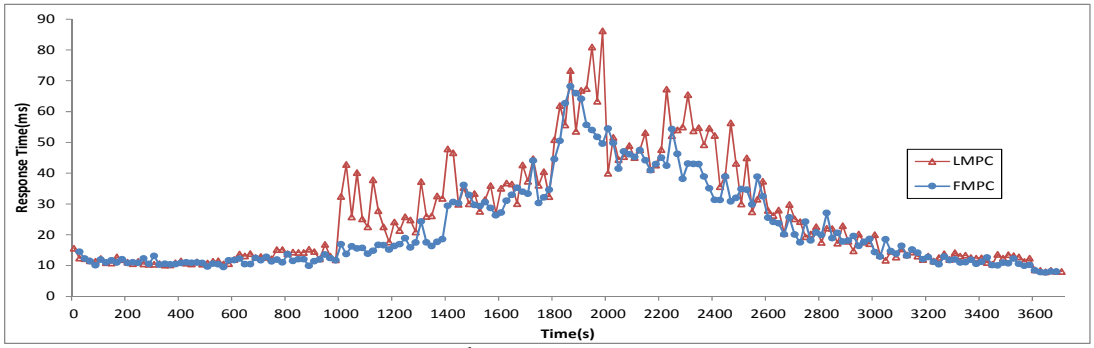

Figure 4-18 Weighted 90th-percentile response time for all VMs

4.5.4. System-level Resource Management

The last group of experiments evaluates the scalability of the proposed two-level cloud resource management framework using a larger testbed. The setup described in Section 4.5.3.c) is extended from single host to six hosts, each initially running eight RUBiS and nine FreeBench VMs. There are a total of 102 VMs under the management of a global scheduler and six node controllers. Each scheduler/controller runs on a dedicated CPU core to prevent interference from the benchmark VMs. An additional six client VMs are used to generate the workloads for the RUBiS VMs. The traces for the RUBiS VMs are created similarly to Section4.5.3.c), where all the RUBiS VMs are divided into six groups and each group starts the replay from different offset of the trace.

This experiment is designed to evaluate the ability of the two-level resource management to use dynamic VM migrations to optimize the overall performance across hosts. The baseline uses only the FMPC-based node controllers but without VM migrations. Figure 4-19 shows the level of QoS violations—the weighted sum of the normalized performance errors—occurred on every host over time using heat map. The *x*-axis shows the time in seconds and the *y*-axis shows the host ID. The gray shades represent different levels of QoS violations (the darker the worse), whereas the white color indicates when all the VMs' QoS targets are met. The results show that the use of VM migration substantially improves the performance of the VMs across the entire system. Overall, the average performance across all the VMs in the system is improved by 23.7% compared when migration is not used. This improvement is made possible by the global scheduler which decides VM migration based on the resource demands estimated using FMPC, and

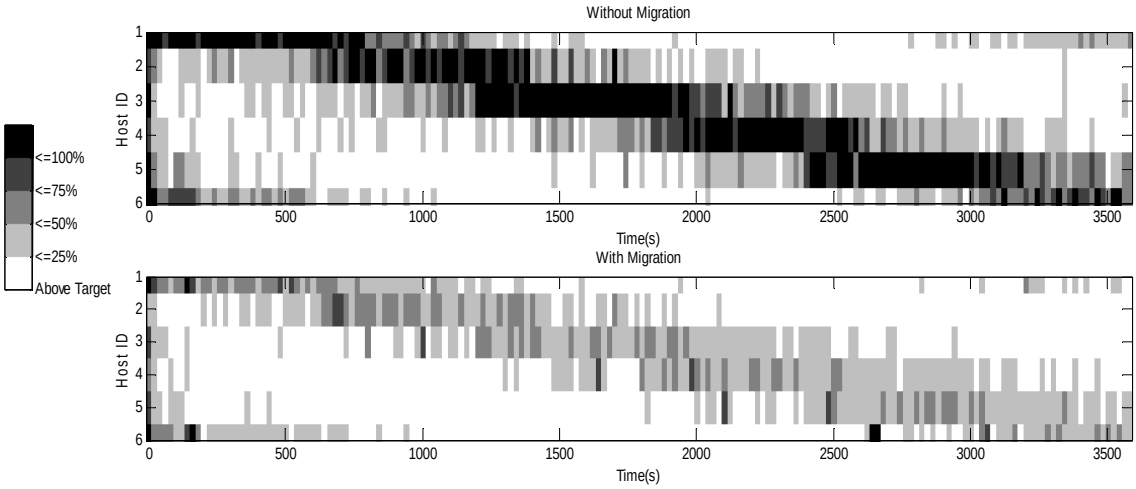

Figure 4-19 Level of QoS violation (weighted sum of the normalized performance errors) across hosts

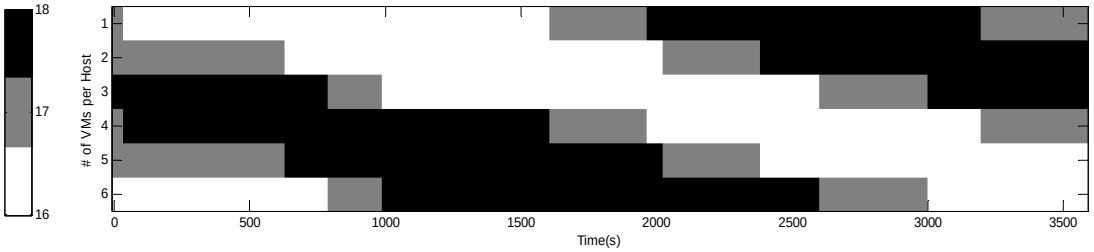

Figure 4-20 Placement of VMs across hosts

by the node controllers which cooperate to migrate the VMs and their performance models. Figure 4-20 also uses a heat map to illustrate the distribution of the VMs over time when migration is employed to balance the load across hosts. The gray shades in the legend represents the number of VMs on a host.

4.6. Summary

This chapter first presents a new fuzzy modeling based predictive control (FMPC) approach which improves the adaptability in the previous chapter's fuzzy-modeling-based

solution by adjusting resource allocation based on observed performance for host-level objective. Then a two-level cloud resource management framework is extended to achieve cross-host management. The node controllers work on the VM host level to estimate VM resource demands and optimize each host's resource allocations. The global scheduler works at the cloud zone level to optimize resource utilization across hosts through dynamic VM migrations.

Extensive experimental evaluation based on a multi-tiered applications and real-world traces prove the effectiveness of the proposed approach. It shows that FMPC can accurately estimate the resource allocation for a VM hosting dynamic workload and achieve the desired QoS. It also shows that FMPC can capture the complex behaviors of competing VMs and optimize the resource allocations under dynamic workload and policy changes in the system. Finally, the experiment with over 100 VMs shows that the proposed two-level resource management can well manage a large number concurrent VMs running on distributed hosts and optimize the performance across the entire system. Compared to traditional LMPC, FMPC is shown to be better in terms of the obtained application performance and the speed and accuracy in achieving the application- or system-level QoS target.

5. APPLICATION-AWARE CROSS-LAYER OPTIMIZATION

Existing resource management solutions in datacenters and cloud systems typically treat VMs as black boxes when making resource allocation decisions, which presents a hurdle to achieving efficient resource allocation for complex workloads and good application performance under dynamic resource availability. In this chapter, we propose a cross-layer optimization based on the fuzzy modeling approaches studied in the previous two chapters and advocate the cooperation between VM host- and guest-layer schedulers for optimizing the resource utilization and application performance. 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 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, complex resource demands and sophisticated configurations.

5.1. Motivating Examples

In this section, we first 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 and then discuss the related work in the literature.

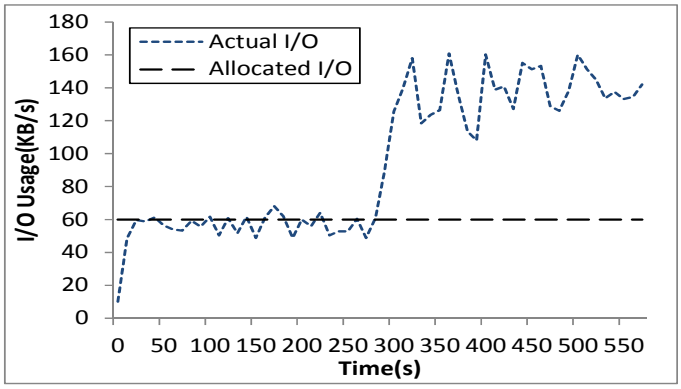

Figure 5-1 I/O Allocation for a changing mix in RUBiS

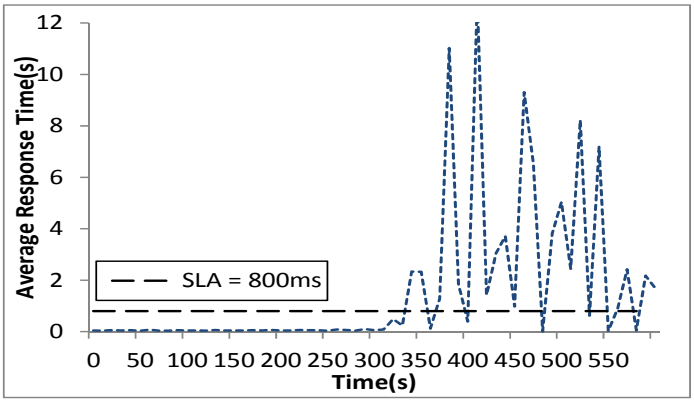

Figure 5-2 Performance for a changing mix in RUBiS

5.1.1. 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 to demonstrate this limitation (Figure 5-1 and Figure 5-2).

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 5-1). 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 5-2). To address this problem, this chapter proposes to exploit application-specific knowledge of workload characteristics in terms of different types of requests in order to make more accurate allocation decisions.

5.1.2. Host-to-Guest Application Adaptation

Different virtualized applications are used as examples here to show the advantages of feeding back the host-layer's resource allocation information to the guest-layer.

In the first two examples, we use examples from virtualized database to show the advantage of feeding back the information of resource availability from host- to guest-layer. We run a workload consisting of a single copy of TPC-H [39] query Q8 against a

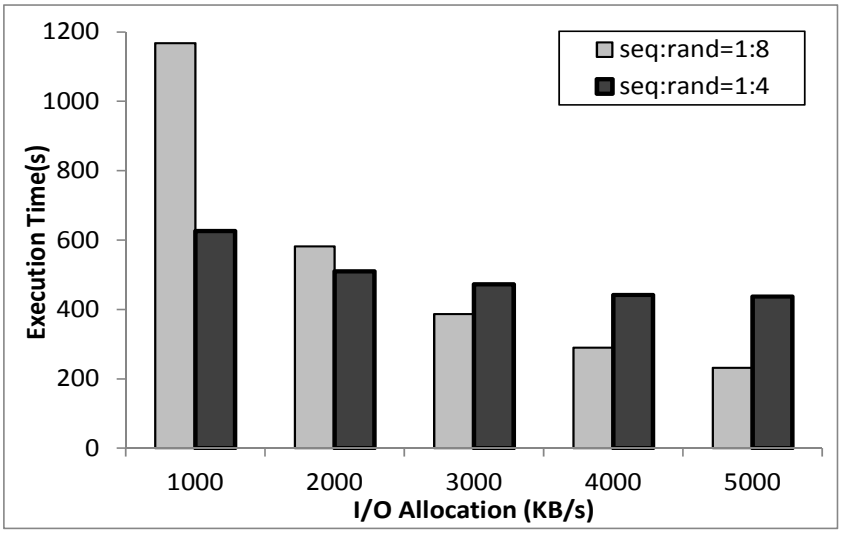

Figure 5-3 Execution time of Q8 with varied I/O allocations

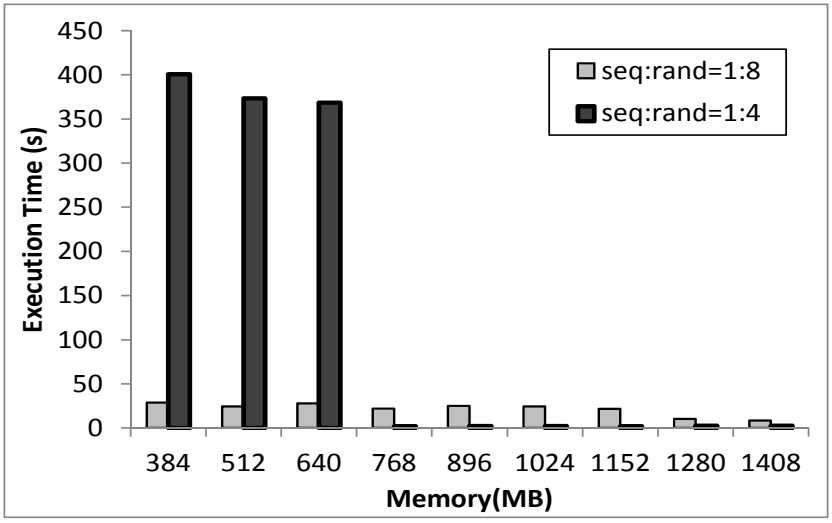

Figure 5-4 Execution time of Q8 with varied memory

3GB database VM. Figure 5-3 and Figure 5-4 compare the query performance using two representative settings of the cost model parameters in database, *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 performance indirectly by influencing the database's 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; while when the ratio is high, the database favors execution plans that use more random scans.

Figure 5-3 shows the performance of, Q8 on a database VM when its memory cache is cold. As the I/O band-width allocated to the VM is reduced 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, the latter's performance is much less affected and becomes faster than the former (by 1.9 times at 1000 KB/s).

Figure 5-4 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. As the memory availability increases 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).

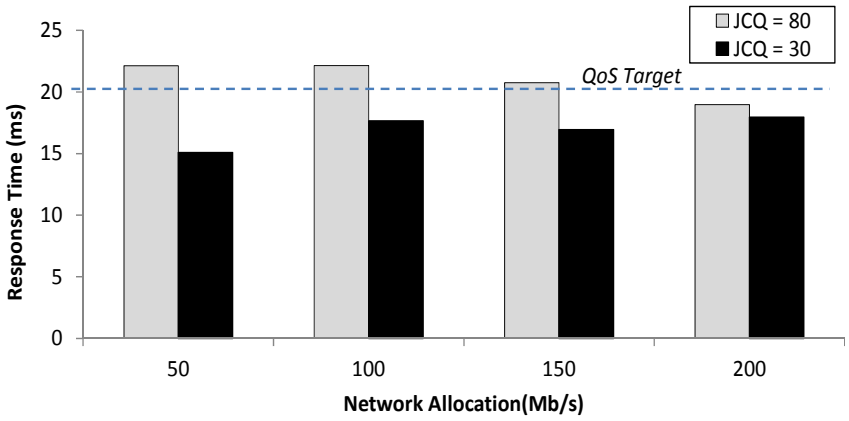

Figure 5-5 Response time of TerraFly workload with varying network allocation

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 5-5, 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 returned map imagery. When the available network bandwidth is sufficient, both configurations can meet the response time target, but the one with higher JCQ is desirable because of its higher image quality. But as the available network bandwidth is reduced, 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.

5.2. General Approach 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 to collaboratively improve the performance of a virtualized application and satisfy its QoS requirement. In traditional resource management solutions, VMs are usually considered 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 in order to meet 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 decision. 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.

5.2.1. The Framework of Cross-Layer VM Resource Management

The cross-layer optimization can be integrated onto the two existing resources management approaches discussed in Chapter 3 and 4. Here for simplicity, we consider to instantiate it on the first solution. In the fuzzy-modeling-based resource management system, since it directly employs a workload-resource model for allocations, it can better

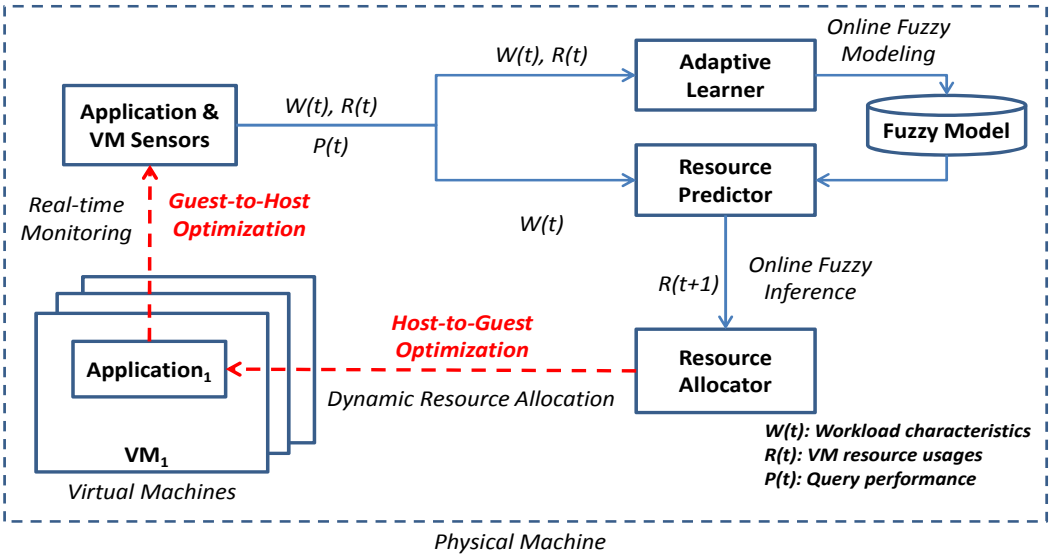

Figure 5-6 Architecture of cross-layer optimization on fuzzy-modeling-based resource management system

demonstrate the effectiveness of our cross-layer optimization in improving the modeling accuracy.

The main challenges to VM resource management are how to efficiently allocate resources to VMs and how to do so automatically and continuously, which have been already addressed in our previous chapter by employing a fuzzy-modeling approach 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 5-6 illustrates the architecture of our fuzzy-modeling-based resource management system integrated with our proposed cross-layer optimization. The four key modules work in a more efficient way with the help of cross-layer communication. As a workload executes on the VM, the

Application Sensor abstracts the workload $W(t)$ more accurately based on the knowledge from application layer, and the *VM Sensor* monitors the corresponding performance $P(t)$ and the VM's resource usage $R(t)$. With a better understanding of the application workload, the *Adaptive Learner* is able to learn a fuzzy model that reflects the relationship between an actual workload and its VM's resource needs. With this model and the precise knowledge of current workload $W(t)$, the *Resource Predictor* can estimate the accurate resource needs for time $t+1$. As the *Resource Allocator* adjusts the allocation accordingly, the allocation decision is feedback to the guest application as well. The internal self-optimization process of the application is then invoked and the corresponding application-level parameters are tuned according to the changes in resource availability for better performance. 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. 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 Chapter 3.

5.2.2. 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.

Specifically, 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. It is often strongly correlated with the VM's resource demands and can be easily obtained without requiring much knowledge of the application's internals. However, this characteristic alone is not sufficient for a real-world workload that consists of requests with diverse use of resources. For a simple 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 request rates are exactly the same (the former consumes mainly CPU while the latter requires also substantial I/O bandwidth). Therefore, 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 VM 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 can

be a 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) databases, 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 5.3.

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 the requests in the workload characterization. A concise representation is needed to effectively compress all the request information, which is critical to ensure low overhead and high robustness of the characterization. To this end, 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, … r_m), the entire workload's composition can then be characterized by the request rates of all these groups (W_{r1}, …, 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 of the online adaptive request clustering is to perform one-pass, non-iterative

clustering of a stream of requests. 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 [32]). Such a clustering approach has the ability to gradually adapt to the changing data patterns. It allows flexible clustering with an evolving shape so that it can better match the current data distribution. The computation complexity of this non-iterative approach is also lower compared to other iterative algorithms.

The above proposed workload characterization process will be performed online periodically, 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.

5.2.3. 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. Another example application is a simulator that tunes the modeling resolution based on its host's resource availability and the performance requirement.

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 applications 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.

In order 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 cross-layer 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 performance 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 (R_i, W_i, C_i) is optimized.

On a physical machine, this process needs to be done only once when an application is first deployed, because the total amount of resource is fixed. We only need to find out the appropriate C_i that leads to the best performance. However when the application i is virtualized, the optimization needs to be done dynamically as the VM's resource availability R_i changes over time. The configuration C_i of the application need to be adjusted accordingly as the given resource allocation to the VM changes. In order to enable such adaption, we need to have a means of mapping the given recourse allocation R_i to a specific configuration C_i by finding the optimal parameter settings for the current environment. 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 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 calibration process. Iterate a variety different value c_{ik} and measure its performance. Collect the parameter value c_{ik_opt} with the best performance.
3) Repeat Step 2 under multiple different candidate resource allocation.
4) Collect the data pairs $<c_{ik\_opt}, R_i>$ for each allocation, 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 into the application to enable its 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 allocations 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 performance optimization. The resulting autonomic resource management system can not only automatically allocate resources to VMs based on their dynamic workload demands but also adaptively improve application performance even when the system is overloaded and the VMs cannot get their requested resources.

5.2.4. Integration with Fuzzy-modeling-based VM Resource Management

The aforementioned two aspects of cross-layer optimization are integrated with the fuzzy-modeling-based VM resource management introduced in Section 5.2.1. For guest-to-host optimization, the workload is characterized by *Application Sensor* based on *application-specific* knowledge obtained from the guest. Specifically, *Application Sensor* can be implemented as a proxy which is deployed on the host of the application. It intercepts all the requests to the application and uses application-specific knowledge to characterize the requests before forwarding them to the application. The workload characterization is used by the *Adaptive Learner* for better modeling and prediction of the VM's resource demands. 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 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 decision 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 its configuration. The next section presents two concrete case studies using two different and representative applications, databases and web-based map services, to demonstrate the cross-layer optimization approach.

5.3. 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 and stream data management, 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. Virtualization allows a database to transparently share

the consolidated resources with other applications, with strong isolation between their dedicated VMs. In a virtualized system, a database's resource usage can elastically grow and shrink based on the dynamic demand of its workload. In addition, it allows efficient database distribution and replication for performance and reliability improvements.

5.3.1. Virtualized Database

a) Guest-to-Host Workload Characterization

Databases are challenging applications 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's 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 [24], and second, it does not capture the entire VM's resource needs.

Specifically, the PostgreSQL database system can be used as an example to demonstrate the guest-to-host workload characterization. PostgreSQL's internal cost model is defined as a function of a set of database cost parameters, denoted as $CostD(C)$

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 database. For example, *seq* and *rand* represent the overhead of a single sequential and random 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 <*CostCPU*, *CostI/O* >.

To characterize a workload, the *Application Sensor* first extracts the cost vector for all unique queries in a database workload and then performs subtractive clustering [9] on the set of collected query cost vectors. This algorithm initially treats each query vector as a potential cluster and selects cluster centers based on the density measures. 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. As soon as a query vector arrives, the *Application Sensor* computes the distance to each existing cluster center and classifies it into the most similar cluster. If it is not within the radius of any cluster, then a new cluster with this new query vector will be added.

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 work-load can be abstracted as a vector of arrival rates of these clusters < C_1, C_2, …, C_K >. Then the above arrival rate vector that reflects the current characteristics of the workload is fed to the *Adaptive Learner* as an input for modeling the VM's current usage behaviors. At the same time, the workload characterization of current time t is also used as the input

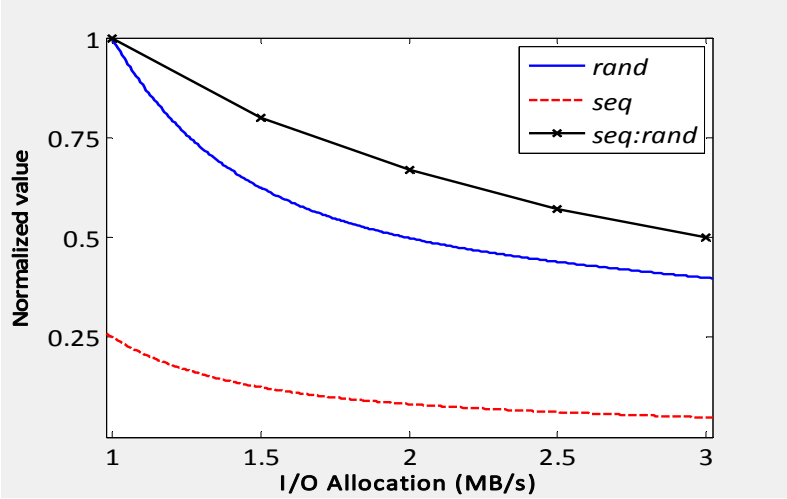

Figure 5-7 The mapping between database cost parameters and VM I/O bandwidth allocation

for the *Resource Predictor* to estimate the resource demands of the next time step t+1 based on the assumption that no abrupt change happens to the workload within one period of time.

b) Host-to-Guest Database Adaptation

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 the aforementioned 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 5.1.2, 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 allocation on the 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 mainly from the scanning operations, the performance of different plans (sequential scan vs. random scan) can be considered as the estimation 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

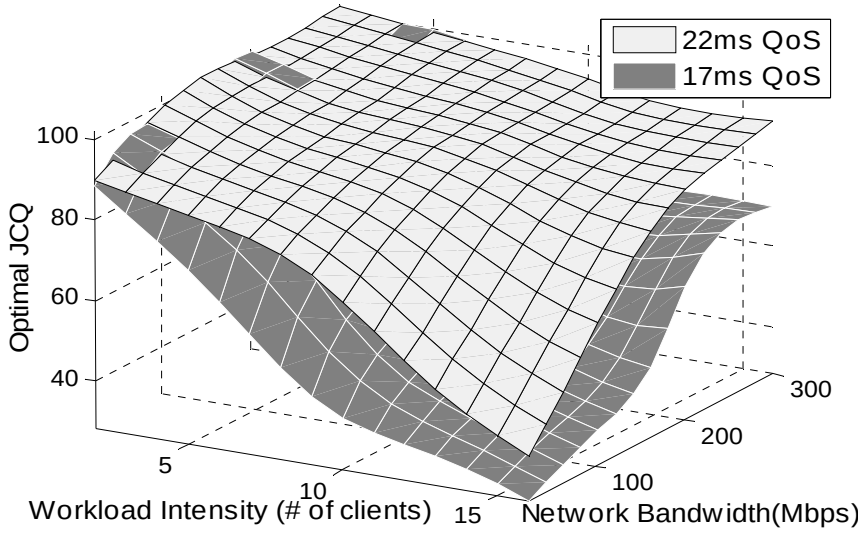

Figure 5-8 The mapping between map service JCQ and workload intensity and VM network bandwidth allocation

parameters (**Error! Reference source not found.**). 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 defines the database's own limit for certain type 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.

5.3.2. Virtualized Map Services

Another interesting case study of this chapter's cross-layer optimization is web-based map services [33][36]. 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. Because the requests to a map service are often well organized by the map tiles, their resource usages are relatively uniform, and a map service workload can be well characterized by using the workload intensity only (e.g., the number of requests per second, the number of concurrent users). Hence, this case study focuses on the second aspect of the cross-layer optimization, the host-to-guest adaptation of map services.

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 solution 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 5.1.2, 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 by varying the network band-width allocation and workload intensity under different JCQ settings. 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. **Error! Reference source not found.** illustrates two of such mappings for the response time targets of 22ms and 17ms. A total of 144 data points were collected to build a mapping in this figure and the fitting error is 2.95% on average. With these mappings, the optimal JCQ value can be then adjusted automatically as the network availability or the workload intensity changes.

5.4. Evaluation

5.4.1. Setup

This section evaluates cross-layer optimization approach using both databases and web map services discussed in the 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 database system, 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 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 us-age 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.

Two typical database benchmarks, TPC-H and RUBiS, are used in the experiments. Experiments performed on TPC-H benchmark are based on synthetic workloads with highly complicated queries in order to show the accuracy in modeling complex resource usage behaviors. For RUBiS, real-world workload is used to show the adaptiveness to dynamic changes in virtualized system.

To evaluate the web map service, Microsoft Hyper-V 6.2 [34] 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 [35] , 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.

5.4.2. Guest to Host Optimization

a) 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 demands and control the resource allocation. The evaluation of more realistic workloads with online training is discussed in Section 5.4.3. 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 that the database's query cost estimation cannot be used directly to infer the VM's resource needs.

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 a 2-dimention input vector for the fuzzy modeling. The result is a 3-D

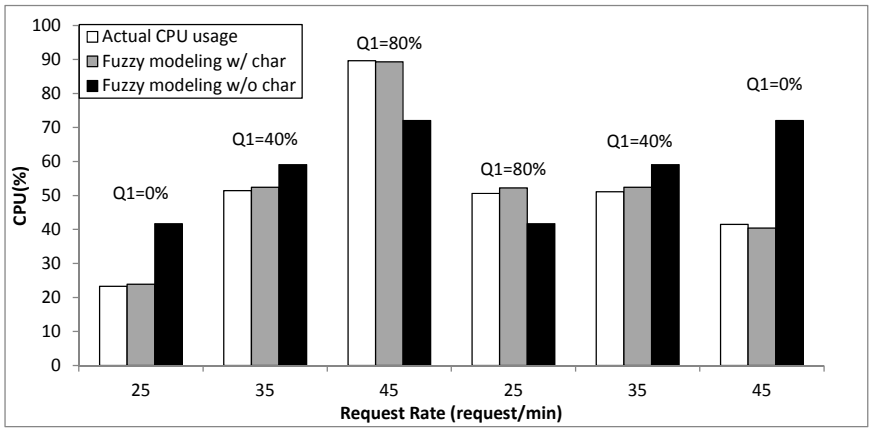

Figure 5-9 CPU allocations for a CPU-intensive TPC-H workload

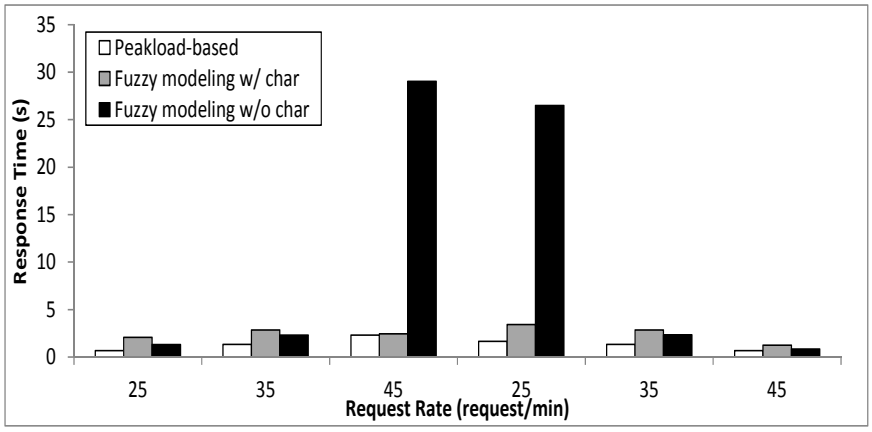

Figure 5-10 Performance for a CPU-intensive TPC-H workload

fuzzy model with 7 rules. 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. As a result, a 2-D fuzzy model with 4 rules is trained. 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 separately.

Error! Reference source not found. 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. **Error! Reference source not found.** 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 model created with workload characterization, 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.

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.

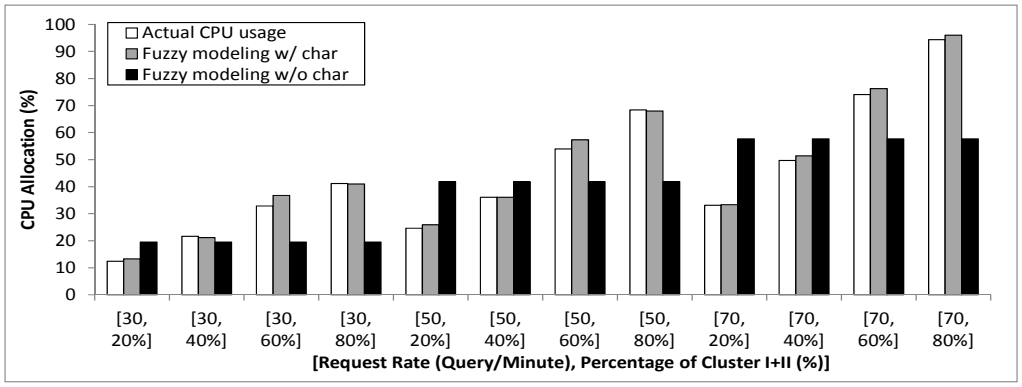

Figure 5-11 CPU allocations for a CPU/IO-intensive TPC-H workload

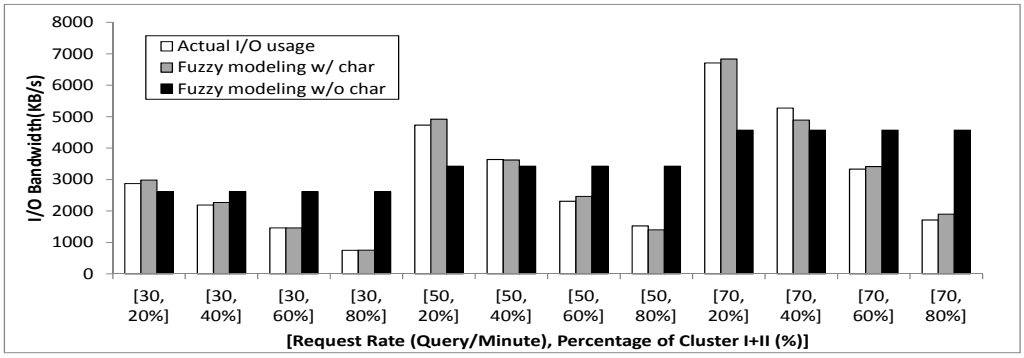

Figure 5-12 I/O allocations for a CPU/IO-intensive TPC-H workload

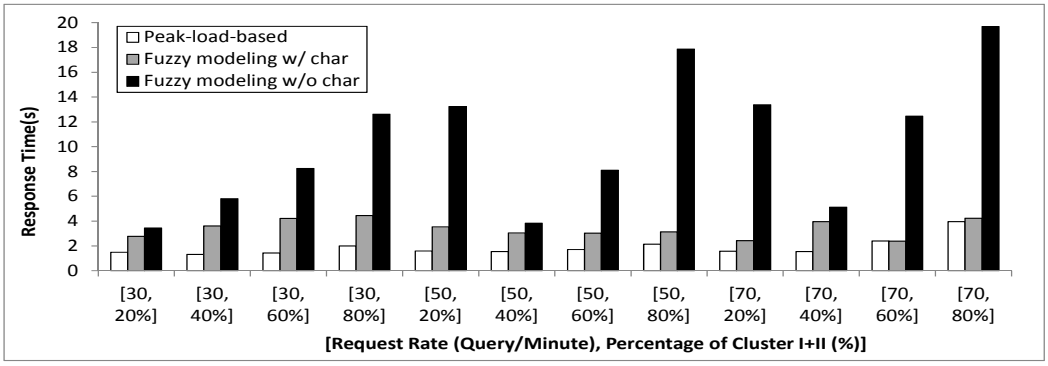

Figure 5-13 Performance for a CPU/IO-intensive TPC-H workload

Error! Reference source not found. and **Error! Reference source not found.** 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. **Error! Reference source not found.** 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 demands 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 can always achieve the same level of performance as in 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 in 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 demands: although 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.

b) RUBiS Experiments

For RUBiS experiment, the same setup as in Chapter 3.4.3 is deployed. The database tier is hosted on the dedicated VM to be controlled. Realistic workloads are simulated according to the real traces from the 1998 World Cup site. The workload with fixed intensity but changing ratio of browsing to bidding request (**Error! Reference source not found.**) is performed on the virtualized database.

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 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 will be used again for guiding the resource allocation. **Error! Reference source not found.** and **Error! Reference source not found.** show the I/O predictions and allocations using a fuzzy model created with or without workload characterization, respectively, for the changing composition RUBiS workload. **Error! Reference source not found.** compares the corresponding performance in both cases with the pre-set QoS target. For the fuzzy

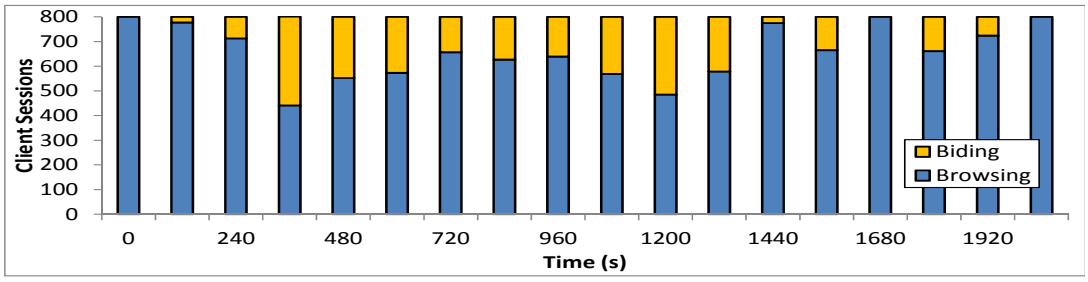

Figure 5-14 Trace for RUBiS with changing composition

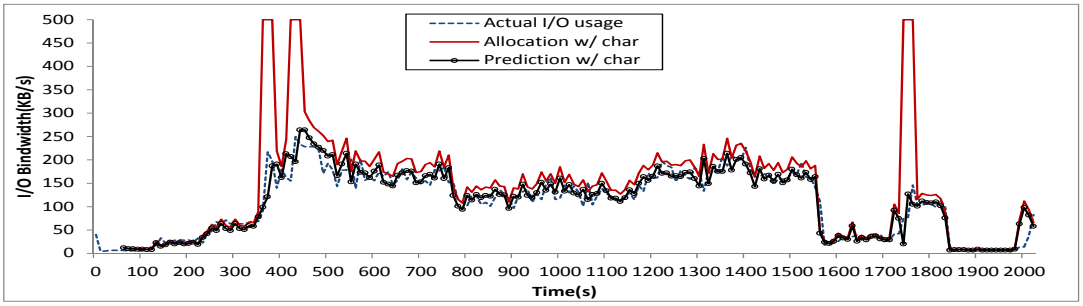

Figure 5-15 I/O allocation with workload characterization

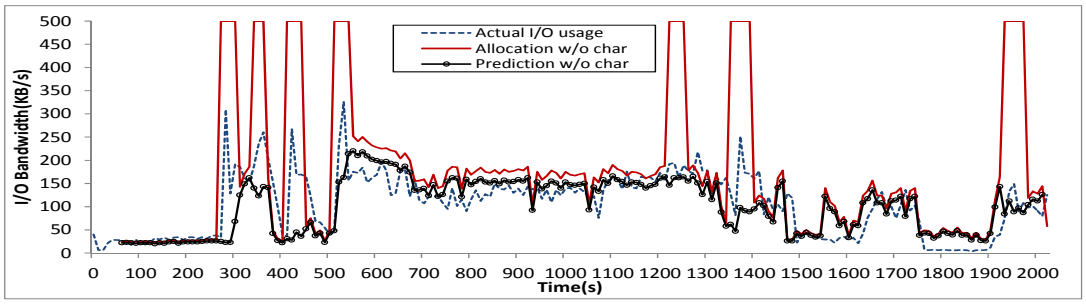

Figure 5-16 I/O allocation without workload characterization

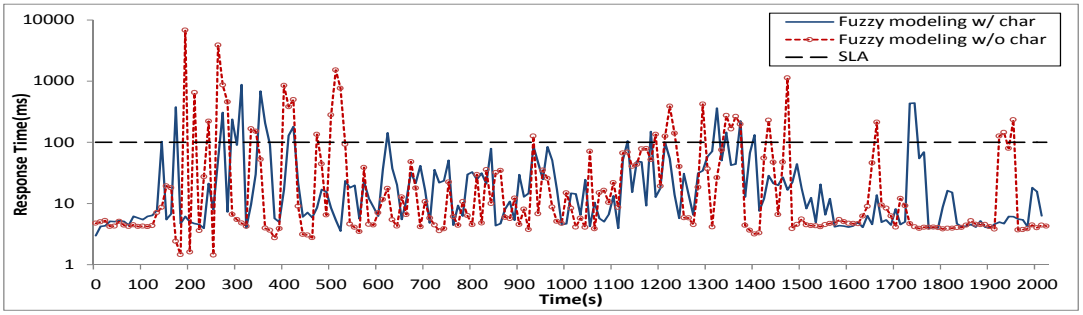

Figure 5-17 Performance comparisons for RUBiS workload

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

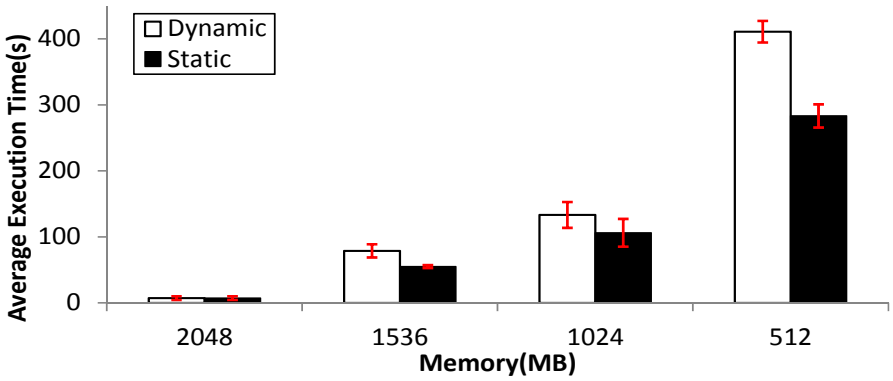

Figure 5-18 Performance of a TPC-H workload with 50 request/s

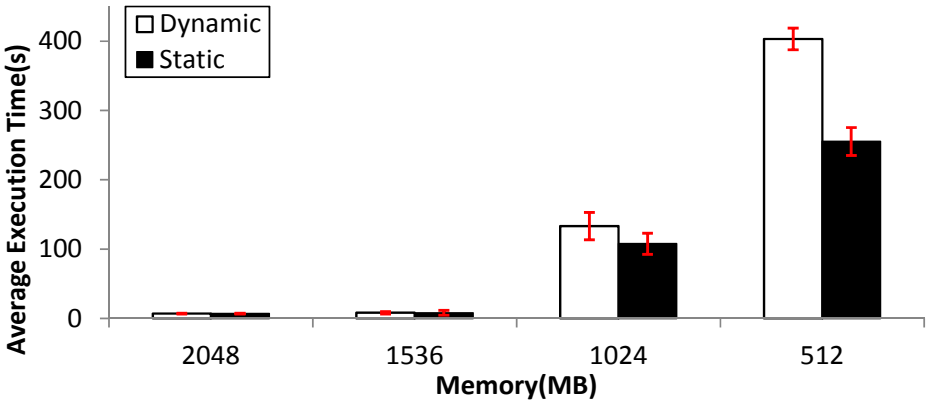

Figure 5-19 Performance of a TPC-H workload with 30 request/s

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.

5.4.3. Host-to-Guest Optimization

a) TPC-H Experiments

This experiment demonstrates the effectiveness of the host-to-guest optimization by automatically tuning a database system under varying memory availability. An I/O

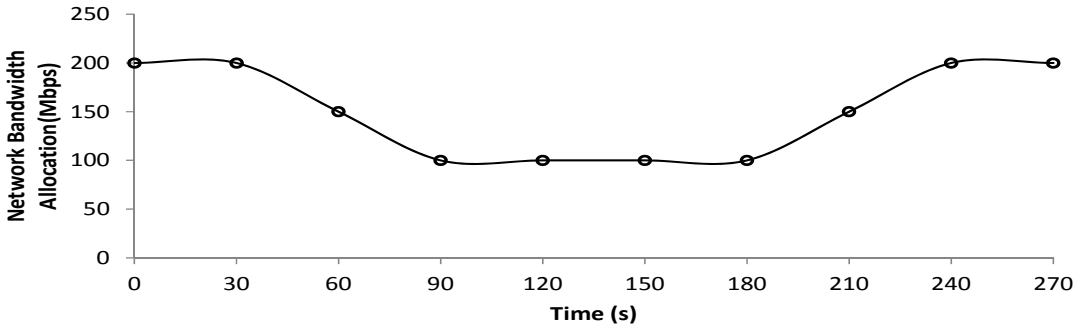

Figure 5-20 Network bandwidth allocations to TerraFly VM

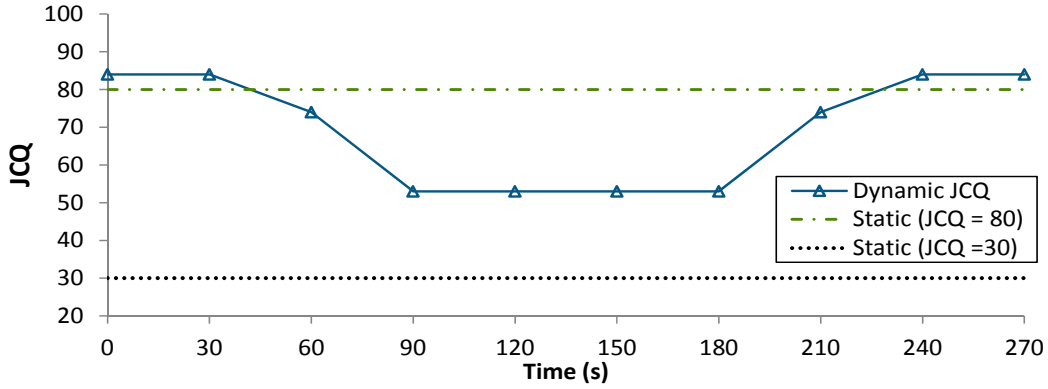

Figure 5-21 TerraFly's JCQ settings

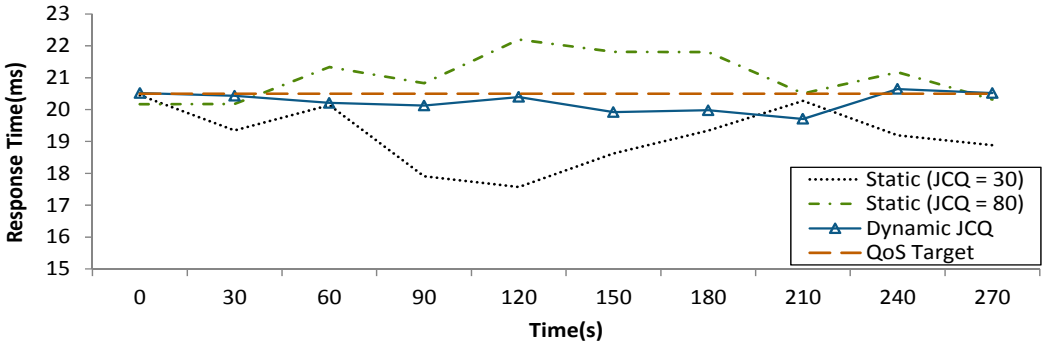

Figure 5-22 TerraFly's performance with different JCQ settings

intensive workload consisting of a mix of duplicated copies of Q4, Q6, Q8 and Q14 from TPC-H queries is run on a database with warm memory, where the query processing can be done mostly using 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.

Error! Reference source not found. and **Error! Reference source not found.** compare the performance of two TPC-H workloads with different intensities from the scheme that uses host-to-guest optimization (*Dynamic*) vs. with-out it (*Static*). The former dynamically adapts the ratio between *seq* and *rand* as the availability memory 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, about 41% speedup is observed; while for the workload with 30 request/s, the advantage of optimization becomes evident (27% speedup) only when the available memory is reduced to under 1GB.

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 switch from a random-scan-preferred configuration to a sequential-scan-preferred one by tuning its ratio of *seq* vs. *rand* from the default 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 bound where sequential scans are more efficient.

b) TerraFly Experiments

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. The trace in **Error! Reference source not found.** shows that the network bandwidth allocated to TerraFly 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.

Error! Reference source not found. compares the performance of TerraFly using three different JCQ settings shown in **Error! Reference source not found.**: 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.

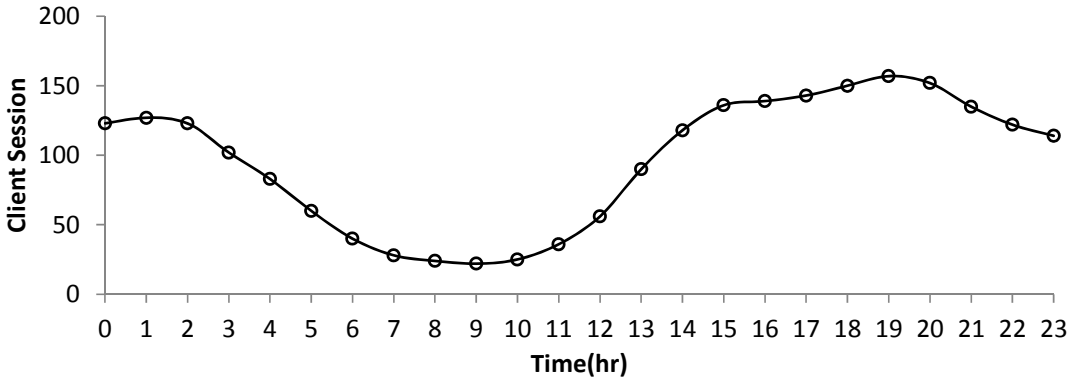

Figure 5-23 A real TerraFly workload with changing intensity

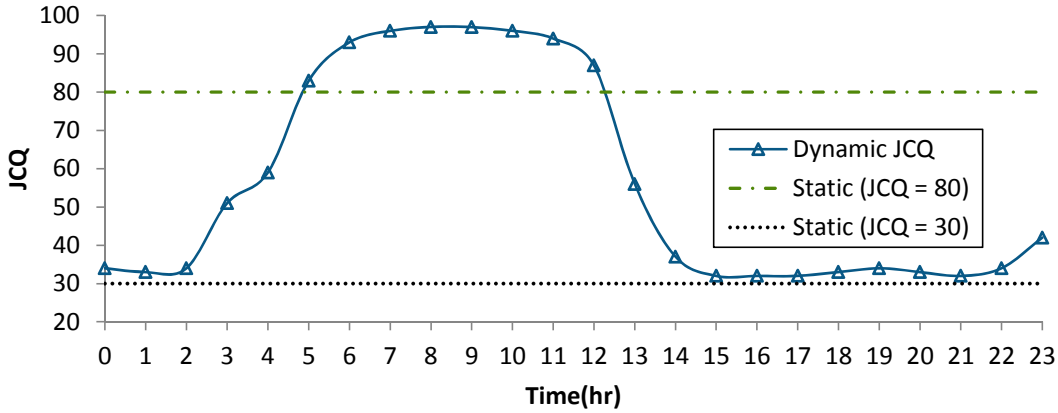

Figure 5-24 TerraFly's JCQ settings

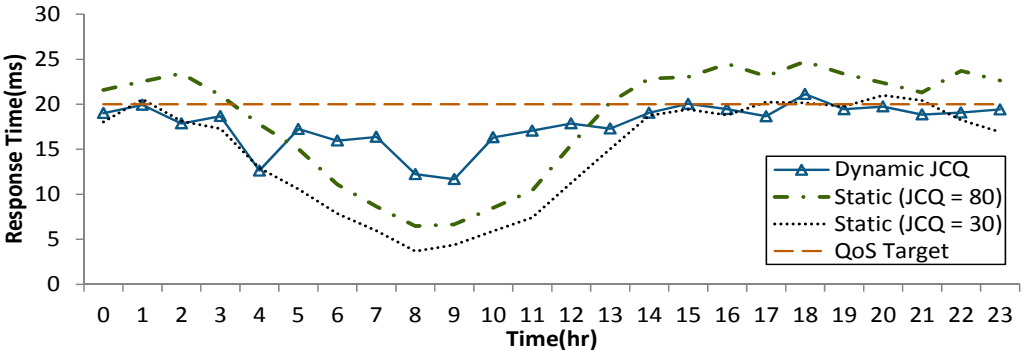

Figure 5-25 TerraFly's performance with different JCQ settings

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 is replayed with a 60-fold speedup (**Error! Reference source not found.**). 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.

Error! Reference source not found. compares the performance of TerraFly using three different JCQ settings shown in **Error! Reference source not found.**. 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 static high JCQ is not able to meet the response time target when the workload intensity be-comes 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 (vs. a static JCQ of 30).

5.4.4. Combining both Guest-to-Host and Host-to-Guest Optimizations

The last experiment further demonstrates the effective-ness of the cross-layer optimization by combining guest-to-host workload characterization and host-to-guest database tuning for an OLAP-like database work-load.

An interesting workload is constructed by mixing multiple copies of Q1, Q4, Q6, and Q14 from the TPC-H queries. To make these queries more diverse in resource usage

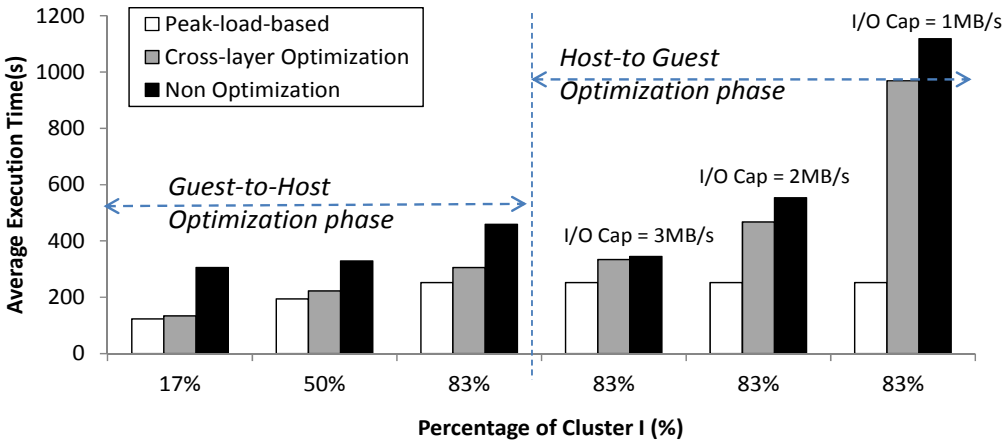

Figure 5-26 Performance of a TPC-H workload with both guest-to-host and host-to-guest optimizations

patterns, distinct query copies are derived from Q4, Q6, and Q14 by modifying the condition in the where cause of 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 is kept constant with 83% of Q1.

Using the cross-layer optimization, during Phase 1, the VM's resource demands are modeled using the workload characterization result, [*Request rate*, *Percentage of ClusterI*], as the input (Q1 is a CPU-intensive query classified to *Cluster I* while the others are I/O

intensive and classified to other clusters). When the experiment transits to Phase 2 and I/O contention is introduced into the system, the cross-layer optimization approach feeds the I/O pressure back to the guest layer by tuning the database parameters according to the resource availability. In comparison, the experiment is repeated with fuzzy-modeling-based resource allocation but without cross-layer optimization. In this case, during Phase 1, 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.

Error! Reference source not found. compares the database's performance under fuzzy-modeling-based resource management with cross-layer optimization (*Cross-layer Optimization*) and without it (*No Optimization*) versus the ideal performance under peak-load-based resource allocation (*Peak-load-based*). The result shows that in Phase 1, the performance from using cross-layer optimization closely follows the one under peak-load-based allocation. It is as much as seven times better than the scheme without cross-layer optimization. In Phase 2, both approaches suffer from the reduced I/O bandwidth. However, the cross-layer optimization still achieves about 17% performance improvement than the scheme without cross-layer optimization. The host-to-guest feedback enables the database query optimizer to switch from a sequential-scan-preferred plan to an random-scan preferred plan by tuning the ratio of *seq* vs. *rand* 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 performance significantly because the random-scan-preferred plan uses indexes which require much less I/O bandwidth than the sequential-scan-preferred one.

5.5. Summary

This chapter presents a new cross-host-guest optimization approach based on the existing fuzzy modeling based resource management by enabling the communication between VM host- and guest-layer schedulers to 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 case studies, the proposed approach is applied to virtualized databases and map services which have challenging dynamic, complex resource demands and sophisticated configurations. 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 VMs serving workloads with dynamically changing intensity and composition and improve the applications' performance when under resource pressure.

6. CONCLUSION AND FUTURE WORK

6.1. Conclusion

In this dissertation, a fuzzy-modeling-based autonomic resource management system is first proposed to automatically allocate resources to VMs based on their QoS targets. The experimental results demonstrate this approach can accurately estimate a VM's resource needs for dynamic and complex workloads based on its desired QoS while improving resource utilization.

However, this modeling-based approach relies on a predefined backup policy to deal with situations where the VM's resource demand is misestimated due to dynamic changes in the VM's resource usage behaviors. To eliminate the need for such a supplementary strategy, we proposed another new approach which combines fuzzy modeling with predictive resource control. This approach allows a VM's resource allocation to be directly adjusted based on the application's performance feedback and the QoS target. It employs multi-input-multi-output fuzzy modeling which can simultaneously model the resource usages of multiple VMs and at the same time capture the interference between them. It also uses live VM migration to further optimize resource usages across hosts. A prototype of the proposed approach is evaluated on a virtualized system using realistic workloads. The experimental results show that it is able to not only automatically track the single QoS target and but also optimize high-level service objective by quickly adapting to changes in the system. The results also show that the approach can effectively manage over one hundred concurrent VMs and optimize their performance across multiple hosts.

As an extension to the base framework in which four major modules work together to form a closed control loop, a cross-layer optimization is proposed to enable the communication between VM host- and guest-layer schedulers and allow them to collaboratively optimize the resource allocation and application performance. As challenging case studies, these proposed approaches are applied to the fuzzy-modeling-based resources management system for virtualized databases and map service. Experiments based on typical database benchmarks, TPC-H and RUBiS, and a map service application, TerraFly, show the cross-layer optimization approach can accurately allocate resource for dynamic and complex workloads and effectively adapt guest-layer's configurations according to its resource allocation, significantly outperforming the application-unaware approaches that treat VMs as black boxes.

6.2. Future Work

In this dissertation, the cross-layer optimization is integrated to one of our resource management frameworks, the fuzzy-modeling-based one. In our future work, we will consider applying it to the second solution, the FMPC-based management system to combine the benefits of host-guest collaboration with predictive control as illustrated in **Error! Reference source not found.**.

From one aspect, the existing model in the FMPC-based system is a performance model, where only the measurements of workload performance are served as the inputs. To enable the guest-to-host optimization, we will consider to involve the workload characteristics from the application-level knowledge in the model to improve the modeling

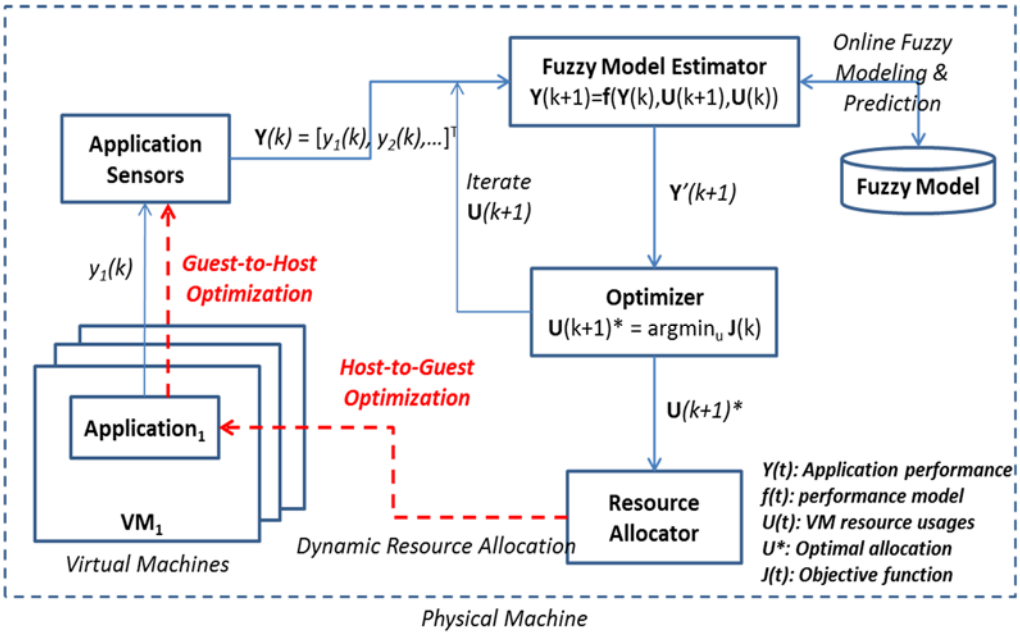

Figure 6-1 Architecture of cross-layer optimization on fuzzy-modeling-based resource management system

accuracy. Then a general *MIMO* model in the FMPC system would be rewritten into the following format:

$$\boldsymbol{y}(t) = \Phi(\boldsymbol{u}(t), \boldsymbol{w}(t), \boldsymbol{y}(t-1))$$

Where $\boldsymbol{w}$(t) represents the workload which is characterized based on guest-level knowledge as we described in Chapter 5.2.2. It will help host-level scheduler adapts to the dynamics in workloads proactively. However, it is also challenging as the dimension of the model increases, and the complexity of training and updating the model may increase considerably.

From another aspect, a host-to-guest layer optimization will be added based on the existing framework of FMPC. There are also new challenges that need to be well addressed. For example, how would the host-to-guest layer optimization affect the modeling part? Since the adaptation of application will further tune guest-level's application based on the

allocation, this may cause more discrepancy between the actual performance measurements and the prediction from the model, and it is likely that the actual performance would be better than the predicted one. Either the model needs to be retrained after performance improvement observed after each adaptation or the mapping between the tunable parameters and resources availability needs to be considered in the optimizer to produce the performance prediction to reflect actual performance after tuning. We also need to take care the system stability issue of adapting the application and its performance model.

The awareness between virtualization software and virtualized application 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 dissertation. 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 operating systems and applications.

REFERENCES

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

[2] P. Barham, B. Dragovic, K.Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, “Xen and the Art of Virtualization”, SOSP, 164-177, 2003

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

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

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

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

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

[8] S. White, J. Hanson, I. Whalley, D. Chess, and J. Kephart., “An Architectural Approach to Autonomic Computing”, In Proc. 1st International Conference on Autonomic Computing (ICAC), 2-9, 2004.

[9] R. Doyle, J. Chase, O. Asad, W. Jin and A. Vahdat, “Model-Based Resource Provisioning in a Web Service Utility”, USENIX, 4:5-5, 2003.

[10] M. Bennani and D. Menasce, “Resource Allocation for Autonomic Data Centers using Analytic Performance Models”, ICAC, 229 - 240, 2005.

[11] X. Liu, X. Zhu, P. Padala, Z. Wang and S. Singhal, “Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform”, CDC, 3792-3799, 2007.

[12] 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, 13-26, 2009.

[13] R.Nathuji and A. Kansal, “Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds”, Eurosys, 237-250, 2010.

[14] J. Wildstrom, P. Stone and E. Witchel, “CARVE: A Cognitive Agent for Resource Value Estimation”, ICAC, 182-191, 2008.

[15] T. Wood, L. Cherkasova, K. Ozonat and P. Shenoy, “Profiling and Modeling Resource Usage of Virtualized Applications”, Middleware, 366-387, 2008.

[16] J. Rao, X. Bu, C. Xu, L. Wang and G. Yin, "VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration", ICAC, 137-146, 2009.

[17] S. Kundu, R. Rangaswami, K. Dutta and M. Zhao, "Application Performance Modeling in a Virtualized Environment," HPCA, 1-10, 2010.

[18] Z. Gong and X. Gu, "PAC: Pattern-driven Application Consolidation for Efficient Cloud Computing", MASCOTS, 24-33, 2010.

[19] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "CloudScale: elastic resource scaling for multi-tenant cloud systems", SOCC, 1-14, 2011.

[20] A. Chen, P. Goes, A. Gupta and J. Marsden, "Heuristics for Selecting Robust Database Structures with Dynamic Query Patterns", EJOR, 168: 200-220, 2006.

[21] M. Wang, T. Madhyastha, N. Chan, S. Papadimitriou and C. Faloutsos, "Data Mining Meets Performance Evaluation: Fast Algorithms for Modeling Bursty Traffic", ICDE, 507-516, 2002.

[22] L. Zadeh, "Fuzzy Sets", Information and Control, 1965.

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

[24] U. Minhas, J. Yadav, A. Aboulnaga, K. Salem, "Database Systems on Virtual Machines: How Much do You Lose?", Intl. Workshop on Self-Managing Database Systems, 35-41, 2008.

[25] A. Soror, U. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis and S. Kamath, "Automatic Virtual Machine Configuration for Database Workloads", SIGMOD, 953-966, 2008

[26] G. Weikum, A. Moenkeberg, C. Hasse and P. Zabback, "Self-tuning Database Technology and Information Services: From Wishful Thinking to Viable Engineering", VLDB, 20-31, 2002.

[27] S. Chaudhuri and G. Weikum, "Foundations of Automated Database Tuning", ICDE, 104-, 2006.

[28] B. Schroeder, M. Harchol-Balter, A. Iyengar and E. Nahum, "Achieving Class-based QoS for Transactional Workloads", ICDE, 153-, 2006.

[29] P. Martin, S. Elnaffar and T. Wasserman, “Workload Models for Autonomic Database Management Systems”, ICAS, 10-, 2006.

[30] T. Wasserman, P. Martin and D. Skillicorn, “Developing a Characterization of Business Intelligence Workloads for Sizing New Database Systems”, DOLAP, 7-13, 2004.

[31] T. Takagi and M. Sugeno, “Fuzzy Identification of Systems and its Application to Modeling and Control” TSMC, 15(1):116-132, 1985.

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

[33] Google Maps, URL: https://maps.google.com/

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

[35] N. Rishe, S.C. Chen, N. Prabakar, M.A. Weiss, “TerraFly: A High-performance Web-based Digital Library System for Spatial Data Access”, International Conference on Data Engineering, 2001

[36] B. Craig. "Online Satellite and Aerial Images: Issues and Analysis" North Dakota Law Review 85 (2007): 547

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

[38] J. Rolia, L. Cherkasova and C. McCarthy, “Configuring Workload Manager Control Parameters for Resource Pools”, NOMS, 127-137, 2006.

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

[40] 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, 3-13, 2002.

[41] Y. Diao, J. Hellerstein and S. Parekh, “Optimizing Quality of Service Using Fuzzy Control”, DSOM, 42-53, 2002

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

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

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

[45] G. Jung, M. Hiltunen, K. Joshi, R. Schlichting and C. Pu, “Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures”, ICDCS, 62-73, 2010.

[46] R. Singh, U. Sharma, E. Cecchet and P. Shenoy, “Autonomic mix-aware provisioning for non-stationary data center workloads”, In Proceedings of the 7th international conference on Autonomic computing (ICAC '10)2010, 21-30.

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

[48] 1998 World Cup Web Site Access Logs, URL: http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[49] D. Gupta, L. Cherkasova, R. Gardner and A. Vahdat., “Enforcing Performance Isolation Across Virtual Machines in Xen,” Middleware, 342-362, 2006.

[50] J. Maciejowski, “Predictive Control with Constraints,” Prentice Hall, 1 edition, 2002.

[51] C. Lu, X. Wang and X. Koutsoukos, “Feedback Utilization Control in Distributed Real-Time Systems with End-To-End Tasks,” TPDS 16(6): 550-561, 2005.

[52] X. Wang, M. Chen and X. Fu, “MIMO Power Control for High-Density Servers in an Enclosure,” TPDS, 21(10):1412-1426, 2010.

[53] Y. Huang, H. Lou, J. Gong, T. Edgar, “Fuzzy Model Predictive Control,” IEEE Transactions on Fuzzy Systems, Vol. 8, No. 6, 2000.

[54] Credit-Based CPU Scheduler, URL: http://wiki.xensource.com/xenwiki/CreditScheduler.

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

[56] FIU-SCIS website: https://cs.fiu.edu

[57] K. Astrom and B. Wittenmark, “Adaptive Control,” 1995.

[58] Neuro-adaptive Learning, URL: http://www.mathworks.com/ help/toolbox/fuzzy/fp715dup12.html.

[59] L.Wang, J. Xu, M. Zhao and J. Fortes, “Adaptive Virtual Resource Management with Fuzzy Model Predictive Control” FeBID, 7 pages, 2011.

[60] L. Wang, J. Xu, M. Zhao, Y. Tu, and J. Fortes, “Fuzzy Modeling based Resource Management for Virtualized Database Systems,” Proceedings of the 19th Annual Meeting of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS2011), 2011, 32-42.

[61] M. Kallahalla, M. Uysal, R. Swaminathan, D. Lowell, M. Wray, T. Christian, N. Edwards, C. Dalton and F. Gittler, “SoftUDC: A Software-based Data Center for Utility Computing,” Computer, 37(11): 38-46, 2004.

[62] T. Abdelzaher, Y. Diao , J. Hellerstein , C. Lu and X. Zhu, “Introduction to Control Theory and its Application to Computing Systems, Performance Modeling and Engineering “, Springer, 185-215, 2008.

[63] B. J. Watson, M. Marwah, D. Gmach, Y. Chen, M. Arlitt, and Z. Wang. “Probabilistic performance modeling of virtualized resource allocation”, In Proc. IEEE Int’l Conf. on Autonomic computing (ICAC), 99-108, 2010.

[64] J. Xu and J. Fortes, “A Multi-objective Approach to Virtual Machine Management in Datacenters,” Proceedings of 8th International Conference on Autonomic Computing (ICAC 2011), 225-234.

[65] J. Xu and J. Fortes, “Multi-objective Virtual Machine Placement in Virtualized Data Center Environments,” Proceedings of 2010 IEEE/ACM International Conference on Green Computing and Communications (GreenCom2010), 179-188.

[66] R. Chiang, J. Hwang, H. Huang and T. Wood, “Matrix: Achieving Predictable Virtual Machine Performance in the Clouds”, ICAC, 45-56, 2014.

[67] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini, “DejaVu: accelerating resource allocation in virtualized environments”, SIGARCH Comput. Archit. News 40, 1, 423-436, March 2012.

[68] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini, “DeepDive: transparently identifying and managing performance interference in virtualized environments”, USENIX ATC'13, 219-230, 2013.

[69] A. Gandhi, P. Dube, A. Karve, A. Kochut and L. Zhang, “Adaptive, Model-driven Autoscaling for Cloud Applications”, ICAC, 57-64, 2014.

[70] Z. Xu, Y. Tu and X. Wang, “Dynamic Energy Estimation of Query Plans in Database Systems”, ICDCS 2013: 83-92.

[71] L. Chen, H. Shen and K. Sapra, “Distributed Autonomous Virtual Resource Management in Datacenters Using Finite-Markov Decision Process”, In Proceedings of the ACM Symposium on Cloud Computing (SOCC’14). Article 24, 13 pages, 2014.

[72] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao and K. Dutta, “Modeling virtualized applications using machine learning techniques”, In Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments (VEE’12) 2012: 3-14.

[73] A. Gulati, G. Shanmuganathan, I. Ahmad, C. Waldspurger, and M. Uysal, “Pesto: online storage performance management in virtualized datacenters”, In Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC’11) 2011, Article 19, 14 pages.

[74] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, “CloudScale: elastic resource scaling for multi-tenant cloud systems”, In Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC’11)2011, Article 5, 14 pages.

[75] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu and H. HacigümüŞ, “ActiveSLA: a profit-oriented admission control framework for database-as-a-service providers”, In Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC’11)2011, Article 15, 14 pages.

[76] T. Salomie, G. Alonso, T. Roscoe and K. Elphinstone, “Application level ballooning for efficient server consolidation”, In Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys’13)2013, 337-350.

VITA

LIXI WANG

2004	B.S., Computer Science Nanjing University of Aeronautics and Astronautics Nanjing, China
2006	M.S., Computer Science Nanjing University of Aeronautics and Astronautics Beijing, China
2007-2015	Doctoral Candidate in Computer Science Florida International University Miami, FL, USA

PUBLICATIONS AND PRESENTATIONS

L. Wang, J. Xu, M. Zhao, *Modeling VM Performance Interference with Fuzzy MIMO Model*, Proceedings of 7th International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID, co-held with ICAC'12), 6 pages, September 2012.

L.Wang, *Modeling VM Performance Interference with Fuzzy MIMO Model*, Paper presented at the International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks, San Jose, CA, September 2012

L. Wang, J. Xu, M. Zhao, *Application-aware Cross-layer Virtual Machine Resource Management*, Proceedings of 9th International Conference on Autonomic Computing (ICAC'12), 13-22, September 2012.

L. Wang, *Application-aware Cross-layer Virtual Machine Resource Management*, Paper presented at the International Conference on Autonomic Computing, San Jose, CA, September 2012.

L. Wang, J. Xu, M. Zhao, Y. Tu, J. Fortes, *Fuzzy Modeling based Resource Management for Virtualized Database Systems*, Proceedings of 19th Annual Meeting of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'11), 32-42, July 2011.

L. Wang, J. Xu, M. Zhao, J. Fortes, *Adaptive Virtual Resource Management with Fuzzy Model Predictive Control*, Proceedings of 6th International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID, co-held with ICAC'11), 7 pages, June 2011.

L. Wang, J. Xu, M. Zhao, J. Fortes, *Adaptive Virtual Resource Management with Fuzzy Model Predictive Control*, Proceedings of 8th International Conference on Autonomic Computing (ICAC'11), 191-192, June 2011.

Y. Xu, L. Wang, D. Arteaga, M. Zhao, Y. Liu, and R. Figueiredo, *Virtualization-based Storage Management for High-end Computing Systems*, Proceedings of 5th Petascale Data Storage Workshop (PDSW, co-held with SC'10), 1-5, November 2010.

J. Martinez, L. Wang, M. Zhao, and S. Sadjadi, *Experimental Study of Large-scale Computing on Virtualized Resources*, Proceedings of 3rd International Workshop on Virtualization Technologies in Distributed Computing (held in conjunction with ICAC-09) (VTDC'09), 35-42, June 2009.