4-1-2013

Improving Resource Management in Virtualized Data Centers using Application Performance Models

Sajib Kundu
Florida International University, skund001@fiu.edu

DOI: 10.25148/etd.FI13042904

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

IMPROVING RESOURCE MANAGEMENT IN VIRTUALIZED DATA CENTERS USING APPLICATION PERFORMANCE MODELS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Sajib Kundu

2013

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Sajib Kundu, and entitled Improving Resource Management in Virtualized Data Centers using Application Performance Models, 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.

Ming Zhao

Tao Li

Kaushik Dutta

Ajay Gulati

Raju Rangaswami, Major Professor

Date of Defense: April 1, 2013

The dissertation of Sajib Kundu is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Dean Lakshmi N. Reddi
University Graduate School

Florida International University, 2013

DEDICATION

To my loving, compassionate, supportive, and assuring mother, father, and brother.

ACKNOWLEDGMENTS

Journey as a PhD student is a rewarding yet an arduous one. Even with the utmost sincerity, perseverance, and dedication from my part, this long endeavor would not have been successful without the blessings and cooperations from my family members, advisor, professors, coworkers, friends, and the sponsors. Graduate research is a symbiotic ecosystem where individual accomplishment is only a part of a collective glory. The thesis contributions belong as much to me as to these people surrounding me.

Without any iota of exaggeration, this work is only possible because of the active and constructive guidance from my academic advisor, Raju. From my inception into the academic program, his supportive, compassionate, prudent, and sometimes admonishing mentorship has been a constant catalyst in stewarding my career. There have been brief moments of glitch between his expectations and my levels of commitment; but, overall, I feel these interactions have transformed me into a better researcher, and a better PhD candidate. It has been a cherishing experience to work with you, Raju and I sincerely thank you for your advice in my professional as well as in my personal life.

I want to hold special regards for my parents, Chandi Charan Kundu and Dipali Kundu, and for my brother, Rajib Kundu. My pursuit of graduate career started with their unconditional love and support for me. I am grateful to you for keeping faith and hope in me.

I would like to thank all of my thesis committee members, especially Kaushik, Ajay, and Ming. Your constant feedback and involvement in forming novel ideas, outlining directions of my research, publications, and writing, helped putting a definitive shape to this thesis. I am much obliged to you for your prompt and unparalleled stimulus to my work.

Also, I would like to thank my co-workers and friends in SyLab, specially Jorge, Luis, and Ricardo. I am so privileged to have you as my friends and lab mates. I would like to specifically mention their expert instructions on setting up lab machines and troubleshooting. I am also grateful to the department staffs, professors, and other graduate and undergraduate students I had interacted with. I also admire NSF and other federal agencies who have financially supported my research throughout the last six years.

Last, but not the least, I reserve special laudatory notes for my dearest friends, Aritra and Roberto. In happiness or in gloominess, if I had needed any sorts of help, I exactly knew whom to consult. I will adore you to the last breath of my life.

May I conclude on a different note by uttering 'NO' to the following: racism, mindless wars, wanton violence against women, and poverty. Let the free thinking, equality, love, peace, and shared prosperity be our guiding stars.

ABSTRACT OF THE DISSERTATION

IMPROVING RESOURCE MANAGEMENT IN VIRTUALIZED DATA CENTERS USING APPLICATION PERFORMANCE MODELS

by

Sajib Kundu

Florida International University, 2013

Miami, Florida

Professor Raju Rangaswami, Major Professor

The rapid growth of virtualized data centers and cloud hosting services is making the management of physical resources such as CPU, memory, and I/O bandwidth in data center servers increasingly important. Server management now involves dealing with multiple dissimilar applications with varying Service-Level-Agreements (SLAs) and multiple resource dimensions. The multiplicity and diversity of resources and applications are rendering administrative tasks more complex and challenging. This thesis aimed to develop a framework and techniques that would help substantially reduce data center management complexity.

We specifically addressed two crucial data center operations. First, we precisely estimated capacity requirements of client virtual machines (VMs) while renting server space in cloud environment. Second, we proposed a systematic process to efficiently allocate physical resources to hosted VMs in a data center. To realize these dual objectives, accurately capturing the effects of resource allocations on application performance is vital. The benefits of accurate application performance modeling are multifold. Cloud users can size their VMs appropriately and pay only for the resources that they need; service providers can also offer a new charging model based on the VMs performance instead of their configured sizes. As a result, clients will pay exactly for the performance they are actually experiencing; on the other hand, administrators will be able to maximize their total revenue by utilizing application performance models and SLAs.

This thesis made the following contributions. First, we identified resource control parameters crucial for distributing physical resources and characterizing contention for virtualized applications in a shared hosting environment. Second, we explored several modeling techniques and confirmed the suitability of two machine learning tools, Artificial Neural Network and Support Vector Machine, to accurately model the performance of virtualized applications. Moreover, we suggested and evaluated modeling optimizations necessary to improve prediction accuracy when using these modeling tools. Third, we presented an approach to optimal VM sizing by employing the performance models we created. Finally, we proposed a revenue-driven resource allocation algorithm which maximizes the SLA-generated revenue for a data center.

TABLE OF CONTENTS

CHAPTER PAGE

1. Introduction 1

2. Thesis Statement, Contributions, and Impact 6
2.1 Thesis Statement 6
2.2 Thesis Contributions 6
2.3 Thesis Impact 8

3. Related Work 9
3.1 Modeling 9
3.1.1 Past Modeling Approaches 10
3.1.2 Building Models 13
3.2 VM Sizing 14
3.3 Dynamic Resource Allocation 15

4. Resource Parameters Selection 19
4.1 CPU 22
4.1.1 Xen-specific Parameters 23
4.1.2 ESX-specific Parameters 23
4.1.3 Impact of CPU Allocation 24
4.2 Memory 25
4.2.1 Xen-specific Parameters 25
4.2.2 ESX-specific Parameters 25
4.2.3 Impact of Memory Allocation 26
4.3 Storage 27
4.3.1 CDIOPS 28
4.3.2 VM I/O Latency 29
4.3.3 Xen-specific Parameters 30
4.3.4 ESX-specific Parameters 31
4.4 Summary 31

5. Application Performance Modeling 33
5.1 Architectural Overview 34
5.2 Model Training 35
5.3 Evaluating Alternative Modeling Techniques 36
5.3.1 Regression Models 38
5.3.2 Artificial Neural Network Models 39
5.4 Optimizing Machine Learning Models 39
5.4.1 Limitations of a Single Global Model 40
5.4.2 Creating Multiple-Models with Sub-Modeling 41
5.5 Evaluation 45

5.5.1 Measure of Confidence 47
5.5.2 Robustness to Noise 48
5.5.3 Modeling Overhead 49
5.6 Summary 50

6. VM Sizing 52
6.1 VM Sizing Problem Definition 53
6.2 Model-based VM Sizing 54
6.3 Summary 56

7. Dynamic Resource Management 57
7.1 Modeling Resource Allocation 59
7.1.1 Problem Formulation 60
7.1.2 How Expensive is Exhaustive Search? 62
7.1.3 Other Heuristic Solutions 63
7.2 A Heuristic Solution 64
7.3 Evaluation 69
7.3.1 Experimental Setup 69
7.3.2 Alternate Solutions 71
7.3.3 Quantitative Evaluation 73
7.4 Summary 76

8. Conclusions 78

9. Future Work 81
9.1 VM Sizing 81
9.2 Performance Modeling 82
9.2.1 Online Updating of Performance Models 83
9.2.2 Cross-platform Performance Modeling 84
9.2.3 Modeling Cache Contention 84
9.3 Dynamic Resource Allocation 85

BIBLIOGRAPHY 89

VITA 96

LIST OF TABLES

TABLE PAGE

3.1 A compendium of related work on application performance modeling . . 10

4.1 Comparison between CDIOPS vs. VM I/O latency for modeling I/O contention. 28

5.1 Prediction error statistics for the regression and ANN techniques 38

5.2 Training and Testing data set sizes. 40

5.3 Prediction errors when using a single ANN Model. 40

5.4 Number of clusters and the average % overlap between consecutive cluster pairs, measured based on the Jaccard coefficient. 43

5.5 % Error statistics (average, median, and 90^{th} percentile) for different modeling techniques by using global models and sub models. 46

7.1 Description of symbols used in resource allocation problem formulation . 60

7.2 Description of symbols used in the algorithm 64

LIST OF FIGURES

FIGURE PAGE

4.1 Impact of *CPU limit*. 24

4.2 Impact of *memory limit*. 26

4.3 Impact of *VM I/O latency*. 29

5.1 Overview of approach. 34

5.2 % Error in prediction for points sorted based on obtained performance for the RUBiS bidding mix benchmark. 41

5.3 Actual performance and predictions using ANN and SVM based sub-models. The x-axis enumerates data points sorted by increasing performance values. The y-axes represent performance (requests/sec for RUBiS and operations/sec for Filebench respectively). 45

5.4 Error Distribution of RUBiS Bidding Mix when using sub models for prediction. The X axis represents CPU Limit (MHz); the Y axis represents Memory Limit (MB). Each box is divided into three columns - representing low, medium, and high (from left to right) VM I/O latency. Error value 0-3% = white, 3-9% = light grey, 9-27% = dark grey, 27% and more = black. 47

5.5 Change in median error when noise is introduced in the training data set. 48

5.6 Training time. Benchmarks appear in increasing training data set size from left to right 49

6.1 Application performance relative to performance target for model-based VM sizing. Each point represents a specific sizing query. 54

6.2 Optimality of VM Sizing under a target performance and a given VM I/O latency. Each box is divided into two triangles - lower triangle represents optimality of CPU, upper triangle stands for memory. The degree of optimality is determined by the color code (shown above) of each triangle. 55

7.1 Architectural Overview for Revenue Maximization. 58

7.2 Illustration of sub-optimal allocation with unit δ increments. 67

7.3 Experimental framework for revenue driven resource allocation. 70

7.4 Change in revenue when started with equal resource allocations 73

7.5 Change in revenue when the initial resource allocations to VMs are assigned proportionally to respective SLA weights 75

7.6 Trend of actual CPU consumption and suggested CPU allocation for a VM across multiple iterations of the resource allocation algorithm. The chosen VM has the lowest SLA priority. Suggested allocations are applied to the VMs using either *shares* or *limits*. 76

CHAPTER 1

INTRODUCTION

Data centers are being increasingly virtualized. The proliferation of virtualization technologies and cloud service providers have also made it easy to create or buy virtual machines (VMs) to host applications. As a result, hundreds of virtual appliances with diverse characteristics can be consolidated in one physical server. Consolidation optimizes the utilization of server resources but leaves administrative tasks in a quagmire. While still attractive relative to traditional non-virtualized hosting, VM sprawl and over-sized VMs present problems in terms of capital and operational expenditure at these data centers. If not mitigated, these problems can potentially forestall the adoption of virtualization techniques; thereby regressing to over-provisioned, dedicated systems with higher costs of resources such as CPU, memory, storage, network, and power.

Since virtualization technology facilitates several heterogenous applications to run in a shared environment, careful attention needs to be directed towards the resource consumption characteristics of individual workloads. As the consolidated application VMs are quite diverse, they exhibit varying degrees of resource demands. Without a thorough understanding of the effects of resource allocation on application performance, VM resource provisioning may be sub-optimal. Moreover, consolidation creates another challenging problem. An application running inside one VM can interfere with the performance of another application running inside another VM that share physical resources. This performance interference is often significant and applications cannot be modeled correctly without accounting for the interference. Unfortunately, the contemporary server management systems do not explicitly address this contention. To minimize the potential for interference affecting performance, administrators either resort to over-provisioning or require clients to pay more for additional resources. Without faithfully capturing the relationship between resource allocation and application performance and accurately understanding the effect of interference, configuration of cloud VMs and distribution of server resources in a data center are likely to be sub-optimal.

Cloud users today pay for a statically configured VM size irrespective of the actual resources consumed by the application (e.g. Amazon EC2 [ec2]). Thus, it is highly desirable for cloud users to size their VMs based on the actual performance needed by their applications and no more. At the same time, cloud service providers can benefit from a performance based charging model built around an application service-level agreement (SLA). Thereby, the service providers can supply and charge for a certain service level to an application, eliminating the need for guess work or over-provisioning by their customers [NKG10]. Doing so could increase customer willingness to pay a higher price for better service compared to paying a flat fee based on the size of their VMs. Moreover, cloud service providers would now have the flexibility to dynamically optimize the resources allocated to VMs based on actual demand.

The complexity of meeting application-level SLAs and doing performance troubleshooting in virtual environments has been mentioned in many recent studies [Dre08, Kot11, Vog08]. As pointed out before, the primary source of this complexity lies in registering accurate relationships between resource allocation and desired performance targets. Modeling is extremely challenging due to multiple resource types being involved and sometimes with inter-dependence among these (e.g. memory and disk I/O). Moreover, while some types of resources (e.g. CPU time, memory capacity) are easy to partition, other types (e.g. storage and network bandwidth) are not, making it hard to find a parameter that can characterize contention in a shared environment. The contention on these hard-to-partition resources can have a significant impact on a VM's performance and this needs to be captured well by the performance model. Virtualization magnifies this impact due to the inherent, underlying sharing and contention [Kot11].

Some previous solutions [PSZ$^+$07, PHZ$^+$09] have partially addressed the resource management problem by applying control-theory based models to proportionally allocate available CPU, storage to running VMs. But, key parts of those solutions were not explained adequately and the chosen SLA metrics were simple priority values (detailed in section 3.3). Other selected research acknowledged modeling the effects of only one type of resource (e.g. CPU [NKG10]) or modeling specific types of applications (e.g. database [SMA$^+$08, SLG$^+$09]). Moreover, storage I/O contention had not been handled explicitly. Further, server resources were distributed in proportion to single or dual priority value(s) based on the respective SLAs. To the best of our knowledge, no work has specifically addressed the resource allocation problem in data centers with the objective of maximizing SLA-generated revenues. In this thesis, we address the above gaps by making the following contributions:

- **Control Parameters:** We identify and study the impact of key VM resource control knobs and contention parameters that affect the performance of virtualized applications. In doing so, we find that the CPU and memory allocation levels are central predictors of application performance. We also find that the I/O latency observed by the virtual machine is an excellent indicator of I/O contention in a shared storage environment. We present a detailed account of the process of parameter selection in Chapter 4. The suggested parameters are application agnostic and are widely available in any virtualization platform.

- **Modeling Techniques:** We apply and thoroughly evaluate two machine learning techniques, Artificial Neural Network (ANN) and Support Vector Machine (SVM), to predict application performance based on the control parameters. We develop *sub-modeling*, a clustering-based approach that overcomes key limitations when directly applying these machine learning tools. Evaluation using a diverse set of benchmarks confirms that these optimization techniques substantially improve prediction accuracy. In Chapter 5, we explain the details of the modeling process, the limitations of standard regression based approaches, and appropriate application of ANN and SVM modeling techniques to VM performance.

- **VM Sizing:** We present a simple and effective approach to sizing the resource (CPU, memory) requirements for VMs in the presence of storage I/O contention based on the performance models that we develop. Evaluation indicates that modeling based VM sizing approach is able to deliver performance guarantees and that the estimated sizes are also optimal or nearly optimal. We present details in Chapter 6.

- **Dynamic Resource Allocation:** Data centers are revenue-driven and the design goal is that the applications experience levels of performances proportional to their shares of revenue as per their SLAs. We develop a greedy algorithm which dynamically repartitions physical resources among VM-hosted applications running in a data center server with the goal of maximizing SLA-generated revenues in US dollars. Evaluation captures substantial increase in revenue when our framework is deployed in comparison to traditional proportional allocation. We explore this contribution vividly in Chapter 7.

We expect that cloud service providers and users, data center administrators and clients will immensely benefit from our proposed solutions. The improvements will propel the growth of virtualization by improving client experience and confidence and significantly reducing data center costs. In the following chapter we formally present our thesis statement, articulate its contributions further, and enunciate its impact.

CHAPTER 2

THESIS STATEMENT, CONTRIBUTIONS, AND IMPACT

In this chapter, we categorize each of our contributions in depth, and extend the scopes of our solutions beyond what we have emphasized in the previous chapter. First, we declare a formal thesis statement. Second, we analyze each of the contributions. Lastly, we discuss how these contributions make significant impacts on constructing powerful resource management tools for virtualized data centers and cloud service providers.

2.1 Thesis Statement

In this thesis, I improve the management of virtualized data center servers by

(i) exploring several machine learning techniques for accurate and robust application performance modeling of the virtualized workloads,

(ii) developing new techniques for sizing of virtual machines with the goals of meeting target performance as well as reducing over-provisioning in terms of CPU and memory allocations, and

(iii) developing new techniques for allocating physical resources dynamically with the objective of maximizing the SLA-based revenue generated for the data centers.

2.2 Thesis Contributions

Management of data centers lacks automation and is often erroneous or sub-optimal due to imprecise and limited understanding of the effects of resource availability and contention on the applications behavior. This results either in over-provisioning of resources or in performance violations of critical client applications. This thesis develops a framework and techniques to automatically and dynamically allocate resources for virtualized data centers in a holistic way.

To address this objective, the first part of this thesis describes building accurate performance models for virtualized applications. Since VM-hosted applications exhibit complex non-linear relationship to the level of allocation of various resources, simple regression models become ineffective [KRDZ10]. Similarly, non-machine learning techniques based on queuing theory [DCA+03] and control theory [PHZ+09] are also incapable of modeling those complex behaviors because they often make idealistic assumptions about the system and use simple linear or quadratic models (elaborated in section 3.1). We found that machine learning techniques e.g. ANN and SVM, are capable of modeling the non-linear resource consumption trends typical of most applications [KRG+12]. Improving model prediction accuracy involves investigating such machine learning tools for VM application performance modeling and more importantly, how to configure them and optimize their usage. Robust and low-error performance models serve as the backbone of the other thesis contributions.

The second contribution of the thesis is VM sizing which is the problem of accurately estimating the amount of CPU and memory required to host client applications in the cloud. The goal here is two fold. First, the calculated allocations must be sufficient to reach the target application performance level for all appliances. Second, the allocations have to be optimal to prevent unnecessary wastage of resources which otherwise can be used to host other applications. We use our application performance models to meet both objectives.

The third contribution empowers administrators to repartition primary physical resources (CPU, Memory and Storage I/O) among VM-hosted applications to maximize revenue for the data centers. Several challenges need to be overcome to achieve this goal. First, the effects of altering resource allocations and contention on individual VMs have to be captured. As in the case of VM sizing, the per-VM performance models are utilized to accomplish this task. Second, an efficient algorithm which is able to maximize SLA-generated revenues by allocating server resources among the hosted workloads by handling diverse performance models and SLA functions, is necessary.

2.3 Thesis Impact

The contributions in this thesis will help both the cloud service providers and clients by bringing ease, transparency, and efficiency in the administrative operations as well as in delivering performance guarantees. Service providers (e.g. Amazon [ec2], Rackspace [rac]) which typically offer Infrastructure-as-a-Service (IaaS), are likely to adopt this work to reduce over-provisioning and to enhance their revenues without relying heavily on human supervision or being over-intrusive in monitoring performance data. Since individual SLAs are respected, clients will also feel comfortable to host their sensitive applications in the shared environment without renting/buying extra resources. The revenue-driven approach will also encourage clients to pay additional rents for guaranteed performance levels. In essence, this thesis underscores the importance of an improved management infrastructure in today's data centers. The proposed solutions are expected to accelerate the move towards shared cloud services as opposed to dedicated servers by boosting client confidence and administrative efficiency. Last, but not the least, the proposed techniques are universal enough to be incorporated in commodity virtualized server management software irrespective of the specifics of the underlying hypervisors and storage systems being used.

CHAPTER 3

RELATED WORK

In this chapter, we extensively explore the existing literature pertaining to this thesis. In doing so, the discussions will follow the contributions listed in the thesis statement (Chapter 2.1). First, we will delve into the modeling techniques related to application performance prediction and troubleshooting. Second, the current status of the works on VM sizing will be discussed. Last, research regarding dynamic resource allocation in servers will be emphasized.

3.1 Modeling

Creating performance models for applications as a function of underlying system parameters is a well researched area. Many previous studies have focused on predicting an application's performance based on low level performance counters related to cache usage, allocation, and miss rates [SKZS08, DCA$^+$03]. Utilizing such models is difficult in virtualized environments because the support of hardware performance counters at the VM granularity is not widely available in production hypervisors. However, virtualized environments provide an unique opportunity to model an application's performance as a function of the size of the VM or underlying hardware resource allocation. The resources allocated to a VM are fungible and can be changed in an online manner. For example, VMware's vSphere utility allows changing the minimum reservation and maximum allocation or relative priority of CPU, memory, and I/O resources available to a VM at runtime. In this section, we examine the work related to application performance modeling as a function of one or more resources.

Modeling Technique	*Strengths*	*Weaknesses*	*Applications*
		Queuing & Control Theory based Techniques	
Queueing Theory	Usability, Speed	Restrictive assumptions	Predicting response times of internet services [DCA$^+$03]
Control Theory	Simplicity	Computational complexity	Linear MIMO models to manage resource for multi-tier applications [PHZ$^+$09]
		Machine Learning (ML) Techniques	
Regression Analysis	Usability, Transparency	Limited scope	Memory resource modeling [WSW08], Translating physical models to virtual ones [WCOS08], Fingerprinting problems [BGF$^+$10]
Bayesian Networks	Extensibility, Transparency	Binary decisions, Domain-based	Fingerpointing for SLA violations [CGK$^+$04], Signature construction of systems history [CZG$^+$05], Building workloads signatures and classifying based on the signature type [VNM$^+$12]
Fuzzy Logic	Extensibility	Usability, Stability	Predicting resource demand of virtualized web applications [XZF$^+$08]
Reinforcement Learning (RL)	Exploratory	Value predictions not supported	CPU/memory resource allocation for VMs [RBX$^+$09]
Kernel Canonical Correlation Analysis (KCCA)	Multivariate analysis	Sensitivity to outliers	Predict Hadoop job execution time [GCF$^+$10]
Artificial Neural Networks	Powerful	Opacity, Configuration, Computational complexity, Overfitting	Performance prediction for virtualized applications [KRDZ10]
Support Vector Machines	Powerful	Opacity, Configuration	Workload modeling in shared storage systems [UYA$^+$05], Estimating power consumption [MAC$^+$11]

Table 3.1: A compendium of related work on application performance modeling

3.1.1 Past Modeling Approaches

We classify these works into two broad classes: (1) Queuing & control theory based techniques and (2) Machine learning techniques. Table 3.1 provides a summary of the related work; we elaborate on each below.

Queuing and Control Theoretic Models. Doyle *et al.* [DCA$^+$03] derived analytical models using basic queuing theory to predict response times of Internet services under different load and resource allocation. Bennani *et al.* [BM05] considered using multi-class queuing networks to predict the response time and throughput for online and batch virtualized workloads. The effectiveness of these solutions is limited by their simplified assumptions about a virtualized system's internal operation based on closed-form mathematical models.

Another related class of solutions have applied control theory to adjust VM resource allocation and achieve the desired application performance. Such solutions often assume a linear performance model for the virtualized application. For example, first-order autoregressive models were used to manage CPU allocation for Web servers [LZSA05, WZS05]. A linear multi-input-multi-output (MIMO) model was used to manage the multi-type resources for multi-tier applications [PHZ$^+$09]. A similar MIMO model was also used to allocate CPU resource for compensating the interference between concurrent VMs [NKG10]. Such linear models are not sufficient to accurately capture the nonlinear behaviors of virtualized applications which are demonstrated and addressed in this thesis.

Machine Learning Approaches. Machine learning techniques have been extensively studied for performance analysis and troubleshooting. The CARVE project employs simple regression analysis to predict the performance impact of memory allocation to VMs [WSW08]. Wood *et al.* use regression to map a resource usage profile obtained from a physical system to one that can be used on a virtualized system [WCOS08]. However, the accuracy of regression analysis has been shown to be poor when used for modeling the performance of virtualized applications under different levels of resource contention [KRDZ10].

Cohen *et. al* [CGK$^+$04] introduced Tree-Augmented Bayesian Networks to identify system metrics attributable towards SLA violations. The models enable an administrator to forecast whether certain values for specific system parameters are indictors of application failures or performance violations. In subsequent work, the authors used Bayesian networks to construct signatures of performance problems based on performance statistics and clustering similar signatures to support searching for previously recorded instances of observed performance problems [CZG+05].

Bodik *et al.* [BGF+10] challenged the usefulness of Bayesian classifiers and instead used logistic regression with L1 regularization to compute the metrics relevant to fingerprint computation. This was shown to be effective for automatic performance crisis diagnosis and in turn facilitating remedial actions. The above techniques address bottleneck identification and forecasting whether certain resource usage and/or application metrics would lead to SLA violations. However, they do not address how much SLA violation would be incurred or how resources should be allocated to prevent future violation. In contrast, we specifically address performance prediction: given a set of controllable/observable parameters, what would the application's performance be? Such prediction can then be used within an optimized resource allocation or VM sizing framework.

Xu *et al.* consider the use of fuzzy logic to model and predict the resource demand of virtualized web applications [XZF+08]. The VCONF project has studied using reinforcement learning combined with ANN to automatically tune the CPU and memory configurations of a VM in order to achieve good performance for its hosted application [RBX+09]. These solutions are specifically targeted for the CPU resource. In addition to CPU, we address memory and I/O contention explicitly in this thesis.

To address "what-if" questions, we apply ANN models in [KRDZ10]. Though the initial application of ANN in [KRDZ10] showed promise, our own investigation later revealed several drawbacks limiting its applicability. First, we observed that the parameter to capture I/O contention in shared storage platform can lead to arbitrary inaccuracy in the model (as demonstrated in section 4.3.1). Second, we also observed that constructing a single model encompassing the entire parameter space in a multi-dimensional model was also severely deficient. In this thesis, we discuss our initial experience with ANN (Chapter 5.3.2) and subsequently propose new modeling techniques that overcome these limitations effectively and evaluate our new techniques on a wide set of real-world virtualized server benchmarks (Chapter 5.4). Further, we demonstrate that our models can also be used for accurate VM sizing (Chapter 6).

We explore the power of both ANN and SVM approaches to machine learning. Although SVMs are generally applied as a powerful classification technique, SVM-based regression (SVR) is gaining popularity in systems data modeling. In [UYA+05], SVR was used to build models to predict response time given a specified load for individual workloads co-hosted on shared storage system. SVR has also been used to model power consumption as a function of hardware and software performance counters [MAC+11]. To the best of our knowledge, prior to our work, SVR has not been used before for performance prediction of virtualized applications.

3.1.2 Building Models

In this thesis, we apply advanced machine learning methods to model the relationship between the resource allocation to a virtualized application and its performance using a limited amount of training data. Such a model is subsequently used to predict the resource needs of an application to meet its performance target. One of the questions in this approach is when and how the model is built. Since our approach requires collecting application performance data for a wide range of resource allocation choices, it is difficult to build the model quickly based only on observations from production runs. One option is to have a staging area where a customer can deploy the application and run a sample workload against various resource allocation configurations to facilitate modeling. We can also leverage recent work like *justrunit* [ZBJ+09] where authors provided a framework for collecting training data by running cloned VMs and applications in an identical physical environment. The modeling techniques that we develop in this thesis can complement and enhance a such a system which used simple linear interpolation to predict performance results for unavailable allocations.

3.2 VM Sizing

The VM sizing problem in cloud services has been explored before under the purview of resource provisioning and capacity management of virtualized servers. There has been research in both industry and academia towards addressing this problem [capa, capb, pla].

Gmach et al. [GRCK07] proposed a trace-driven approach to predict required resource capacity for a set of workloads. Specifically, it addressed whether a resource pool has sufficient resources to host a new workload, packing of workloads on a specific server, and future demand predictions. Wood et al. [WCOS08] estimated resource requirements when applications are transitioned from non-virtualized system to virtualized system. They did so by running a series of micro-benchmarks on the non-virtualized system and on the target virtualized system and subsequently devised a regression model to capture the virtualization overheard. A workload trace was used as the basis for estimating resource usage in the non-virtualized system. Very recently, Meng et al. [MIK+10] pointed out that capacity prediction by sizing individual VMs separately leads to wastage of physical resources. Instead, they proposed a joint VM sizing approach which statistically multiplexes the resource demands of individual clients. They also offered a VM selection mechanism whereby administrators have the ability to group VMs on a physical server based on individual resource requirements estimated by application-specified SLA models.

All of these previous approaches have dealt with VM sizing from the point of view of capacity planning of the data centers. However, none of them have offered any flexibility towards choosing an appropriate VM size based on the clients performance target. In our work, we aim to provide a framework that will provide clients the ability to choose the customized resource sizes that will guarantee their target performance and the selected sizes will be close to optimal. We make a formal statement of our sizing problem and show the benefits of performance modeling based VM sizing in Chapter 6.

3.3 Dynamic Resource Allocation

The related work on resource allocation in virtualized environments primarily fall into two broad categories: (a) application-specific solutions that employ domain knowledge as an integral part of the optimizer, and (b) dynamic reallocation of a specific resource type (in most cases CPU) for an arbitrary application class.

Aboulnaga et al. [SMA$^+$08] proposed automatic virtual machine configurations for database workloads. They implemented a virtualization design advisor that uses information about the virtualized database workloads to generate optimal configuration parameters for each VM hosting a database instance. At a high level, this approach addresses resource reallocation to minimize the cost associated with each database workload which is typically the execution time. While this work relies on expert knowledge about the database with the cost estimation model varying as the type of database changes, our approach is designed to be application-agnostic. ActiveSLA [XCZ$^+$11] suggested a framework for admission control of individual queries in cloud database systems where admission decisions are guided by SLAs and expected profits. In contrast, we formally model the application-independent version of the resource reallocation problem and compare it against known allocation-based optimization problems. We finally present an efficient algorithm that provides a widely applicable heuristic solution.

Recent work on adaptive control of virtualized resources in data centers [PSZ+07] describes an approach for handling multi-tier applications with the high-level goals of *(i)* guaranteed application-level QoS, *(ii)* high-resource utilization across all physical nodes, and *(iii)* QoS differentiation during resource contention. However, this work only considers CPU allocation and it is not clear how it would work for other types of resources. The follow-up work by the authors proposed an improved version [PHZ+09] which addresses among other improvements: *(i)* CPU as well as disk I/O resources, *(ii)* service level objectives within the contention differentiation metric, and *(iii)* the restriction of hosting a particular tier in a specific node. The optimization objective we consider in our work is different in the sense that we consider SLA to be more complex including not only the targeted performance metric and priority, but also differential revenue for achieving different levels of performance. Our problem formulation and solution are explicitly geared towards maximizing the revenue for the data center at any instant. Very recently, CloudScale [SSGW11] was proposed as an automatic elastic resource scaling system for multi-tenant cloud computing infrastructure to minimize SLO violations and meeting additional objectives of optimizing physical resource usage and energy savings. Again, the work is targeted towards the prevention of SLO violations and is only capable of predicting CPU load demands.

Some systems e.g. *Pesto* [GSA+11], *Cake* [WVA+12] dealt only with storage system provisioning in accordance to the SLOs. Xu's et al. work [XZF+08] on virtualized data center resource management also considered the profit-driven resource optimization problem, which was formalized as an instance of the continuous knapsack problem and solved using a greedy allocation algorithm. In comparison, this thesis recognizes the discrete nature of VM resource allocation and the need of incremental resource reallocation in order to ensure system stability and performance prediction accuracy.

Q-cloud [NKG10] aims to mitigate performance interference potentially caused by running several disparate VMs onto a single server. Under Q-cloud the client can expect to get the same performance as will be achieved by running the application in a dedicated system. Additionally, clients can provide multiple levels of Q-states and thereby cloud providers can distribute unused resources to the application VMs and hence earn more rent by increasing resource utilization. The implementation was directed towards CPU bound applications and the reason was attributed towards the unavailability of I/O capping mechanisms. However, their proposed models assumed a degree of linearity in resource consumption trends. In Chapter 4, we will show that linear trend is hardly the case for virtualized applications. Apart from our robust modeling technique and handling of CPU, memory and storage I/O, the most important difference lies in our goals. We are not just interested in performance interference removal, but we also envision a market or revenue driven approach where application VMs are rewarded or penalized based on their performance SLA curves given a certain amount of minimum reservations.

Recently, Bryant et al. proposed a prototype of a micro-elastic server called *Kaleidoscope* [BTI+11] to dynamically create small cloned worker VMs to satisfy increased demand on a target VM. They used a novel VM state coloring technique to glean useful semantic information of guest OS page tables and then clone VM states to instantly create replicas of those VMs which can satisfy additional user requests for parent VMs. Although the technique is promising, it did not address how the physical resources should be distributed according to their respective SLAs. Essentially, we view kaleidoscope as a complementary solution that satisfies instantaneous load spikes in user VMs. On the other hand, our revenue driven approach delivers a more effective resource partitioning when the loads on the VMs are stable and the resource allocation decision is guided by the SLAs.

The rest of the chapters are laid out as follows. Chapters 4 investigates the resource control parameters crucial for performance modeling as well as resource provisioning. Chapter 5 explores several modeling techniques and proposes new optimizations to existing machine-learning based approaches for accurate performance prediction. Chapter 6 discusses a performance model-based VM sizing approach. Chapter 7 emphasizes the potential of a novel revenue driven resource allocation algorithm that employs the performance models we create. Chapter 8 summarizes the thesis. Chapter 9 concludes by delineating several directions for future reserach.

CHAPTER 4

RESOURCE PARAMETERS SELECTION

The resources allocated to a virtual machine directly impact the hosted application's performance. Choosing appropriate control knobs to handle resource allocation for a VM is critical to ensure desirable performance and create a robust model. The purpose of this chapter is three folds. First, we discuss the principles we follow while selecting control parameters. Second, we identify the knobs in the form of VM resource allocation parameters that can be used to directly control application performance. We focus our discussions on control knobs available both in industry dominated VMware ESX hypervisor [VMw10a], and in university open-source Xen hypervisor [BDF$^+$03]. Third, we demonstrate that the relationship between these controls and the application performance is quite complex and hard to model. Several profiles of RUBiS [rub] and Filebench [fil] are chosen for this study; these applications consume a variety of physical resources (CPU, Memory and I/O) in a complex fashion. The brief descriptions of the workloads are presented next.

RUBiS Browsing. A Java servlet based RUBiS [rub] Browsing workload was used for the experiments. Client user requests were handled by a Tomcat web server and the underlying database was MySQL. The webserver, database and clients were run on the same VM to minimize network effects that we do not address in this work. 1000 clients were run simultaneously. The Browsing Mix consists of 100% read-only interactions. After each run, RUBiS reported average throughput as requests/sec, which was used as our application performance metric.

RUBiS Bidding. A similar set up was used for RUBiS Bidding Mix workload with 15% writes. 400 clients running simultaneously were used and the performance metric was average requests/sec as before. Both of the RUBiS profiles are CPU and memory intensive. They generate a small number of I/Os and hence are largely insensitive to the I/O contention.

Filebench-OLTP. Filebench [fil] is a widely used benchmark for creating realistic I/O intensive workloads such as OLTP, webserver, mail server, etc. We used the Linux based Filebench tool and ran the OLTP application profile which emulates a transaction processing workload. This profile tests for the performance of small random reads and writes, and is sensitive to the latency of moderate (128k+) synchronous writes to the log file. We configured the benchmark to create 32 reader threads and 4 writer threads; I/O size was set to 2KB with a 10GB dataset. We took the Operations per Second (Ops/Sec) reported by Filebench as the application performance metric for this workload.

Filebench-Webserver. We also used the webserver profile that performs a mix of open, read, close on multiple files in a directory tree, accompanied by a file append to simulate the web log. We created a fileset of total size 10GB and used 32 threads. Application performance was recorded in terms of operations per second (Ops/Sec).

Filebench-Fileserver. The Filebench-fileserver workload performs a sequence of creates, deletes, appends, reads, writes and attribute operations on the file system. A configurable hierarchical directory structure is used for the file set. Similar to the Webserver benchmark, we used a dataset of 10GB and 32 simultaneous threads.

For the experiments, we used an AMD-based Dell PowerEdge 2970 server with dual socket and six 2.4 GHz cores per socket. The server has 32 GB of physical memory and ran VMware ESX-4.1 hypervisor. All the VMs ran Ubuntu-Linux-10.04. VMs were restricted to use only four specific cores (0-3). Remaining cores were kept idle. All the virtual disks for VMs, and the ESX install were on a VMFS [CAVL09] (VMware's clustered file system) data-store on local 7200 RPM SAS drives from Seagate. We used a VMware vSphere client running on a separate physical machine for managing resources of the individual VMs. The statistics were collected using esxtop utility and all the data was transferred to a separate Dell PowerEdge T105 machine with quad-core AMD Opteron processor (1.15GHz×4), 8 GBs of physical memory, 7.2k rpm disk, running Ubuntu-Linux-10.10 for analysis. All modeling tasks were also done using this machine.

To simulate I/O contention, we used a separate Ubuntu-Linux VM with 1000 MHz of VCPU and 512 MB of memory which ran fio [fio] - a Linux-based I/O workload generation tool co-located with the virtualized application being modeled. The number of outstanding I/Os (OIO) and other workload parameters (e.g., sequentiality) were varied to create different levels of I/O contention on the shared VMFS data-store. Another VM on the same host was used to run Perl-based scripts for changing the allocation parameters for the VMs running the benchmarks.

We look for following requirements to hold true while identifying resource parameters. First, the parameters must either directly map to or indirectly reflect known resource usage behavior of processes, and they must be easy to control and/or observe. This will allow system administrators to intuitively use such parameters. Second, contentions in shared environment have to be accounted. For example, an I/O intensive application running in one VM may affect the I/O operations of an application running in another VM. We explicitly address storage I/O contention, local or networked. But, we do not handle (non-storage) network I/O contention in our work. While host-level NIC bandwidth is typically not a bottleneck, there is little control over in-flight packages once they leave the host. Datacenter level solutions are necessary to manage network I/O contentions. Third, we select the minimum set of model parameters that efficiently captures application performance with high-accuracy and are yet application-independent. We take a minimalistic approach to parameter selection by starting from known, high-level, system resources that can directly impact application performance including CPU, memory, and disk.

Finally, our chosen knobs are generic enough to be found in any hypervisor platform.

Although not considered in this thesis, it is conceivable that processor cache resources can be also incorporated as an additional parameter, either by controlling the cache allocation (if it is partitionable) across VMs or by taking into account the influence of cache contention between VMs via hardware performance counters. In this work, we have mostly focussed on macro-level resource management rather than micro-level characterization (e.g. cache misses). Next, we delve deeper into parameterizing each of the key resource dimensions - CPU, memory, and disk. A special attention has been given to model disk I/O contention.

4.1 CPU

The common practice in modeling CPU usage by an application is establishing a correlation between the average or peak CPU utilization of an application and its observed performance [DO00, SKZS08, SS05, WCOS08]. These models have been used for application placement to predict running times of applications [DO00], to predict CPU utilization at different application load levels for capacity planning purposes [SS05], for cross-platform performance prediction [SKZS08], and for mapping resource usage of an application running natively to that when the application runs within a VM [WCOS08].

Since a primary goal for performance modeling in our case is to provide tunable knobs to the system administrator for controlling performance, the commonly used CPU utilization, an observable (rather than controllable) parameter, is ill-suited. Moreover, forcing the application to specific CPU utilization levels is necessary to create a model that predicts performance based on CPU utilization and requires changing application load levels, thus requiring knowledge of application semantics. Instead, we choose the *CPU allocation* which merely imposes an upper limit on CPU utilization, and is a basic control parameter across all virtualization architectures and solutions. This parameter can be directly utilized by a data center system administrator to determine the expected application performance for a given CPU resource allocation.

4.1.1 Xen-specific Parameters

By default, Xen uses a *credit scheduler* for time-sharing CPU cycles across the VMs, including the dom-0 VM. We instantiate the *CPU allocation* generic parameter as the Xen-specific *CAP* parameter which places a upper bound on a VM's CPU usage and can be changed dynamically from within dom-0 (the controller VM in Xen hypervisor) at run-time [KRDZ10].

4.1.2 ESX-specific Parameters

ESX provides three control knobs for CPU allocation to individual VMs: *reservation, limit,* and *shares* [VMw10b]. Reservation guarantees a certain minimum CPU allocation expressed in MHz. Limit (in MHz) provides an upper bound on the CPU allocation. Share provides a mechanism for proportional allocation during time periods when the sum of the CPU demands of the currently running VMs exceeds the capacity of the physical host. We chose *reservation* and *limit*, both set to the same value as our control knob, to enforce the physical segregation of CPU resources across multiple VMs running on a single physical machine and to ensure that the VM will never get any allocation more than the set value as well. This approach is similar to the implementation in many public clouds such as Amazon EC2. In multiple SLA-level environments, reservation and limit can be set at different values for guaranteeing minimum performance and higher performance respectively.

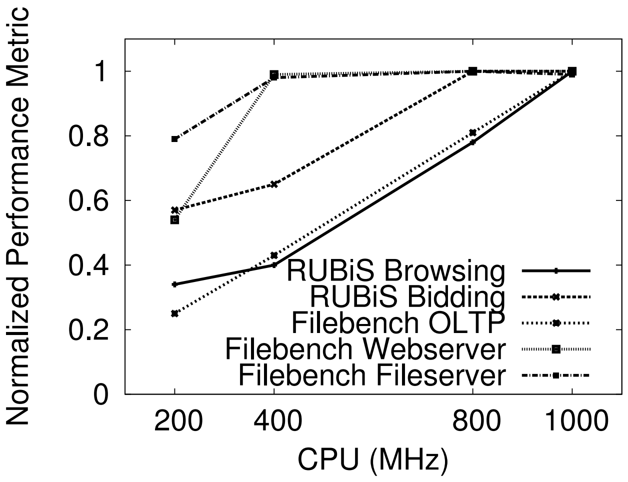


Figure 4.1: Impact of *CPU limit*.

4.1.3 Impact of CPU Allocation

Even with such physical segregation for the CPU, the typical relationship between application performance and the CPU allocation is complex. We exemplify this complexity in ESX environment by setting CPU limit at different levels. We measure the performance of the five virtualized applications while varying the VM's CPU limit from 200 MHz to 1 GHz. The memory allocations were kept high enough to ensure that memory is not the bottleneck. We used a VMFS data store on a local disk on the ESX host to store the virtual disks of the VMs. Figure 4.1 shows the normalized performance of these applications. As seen from the graph, both the RUBiS workloads behave non-linearly; the performance slope is different at various CPU allocation ranges. The three personalities of Filebench: OLTP, webserver, and fileserver behave quite differently. While the webserver and fileserver performances saturate quickly at 400MHz, OLTP performance, on the other hand, varies almost linearly with CPU allocation. Overall, this data reveals that virtualized workloads can have quite different performance curves with respect to CPU allocation [KRG+12].

4.2 Memory

The use of memory utilization for modeling application performance has been explored before [SS05]. In a virtualized environment, besides sharing the same drawbacks as CPU utilization when used as a control knob, the memory utilization metric also incorrectly characterizes an unused portions of the file system page cache as part of the memory utilization of a VM which gets attributed to the resident application; application performance and its memory utilization can change substantially while the VM memory utilization remains constant, and vice-versa.

Virtualization allows the VMs sharing the host physical memory to have their own isolated memory allocation. Following the rationale for the CPU resource parameter, we choose the the VM *memory allocation* for the VM as our model parameter. This parameter provides a control knob that is available across all virtualization solutions.

4.2.1 Xen-specific Parameters

Changing memory allocation for a VM is very straightforward in Xen. These memory allocation limits are strictly enforced by the virtual machine monitor. The *xm mem-set* command can be issued from dom-0 to change the memory allocation of a VM dynamically, allowing full control over dynamic memory repartitioning across VMs as needed. Thus, we instantiate the *memory allocation* generic parameter as the Xen-specific *mem-set alloc (MEM)* parameter [KRDZ10].

4.2.2 ESX-specific Parameters

Similar to the knobs for CPU, the ESX hypervisor provides three controls for memory allocation: *reservation, limit*, and *shares.* The semantics of these knobs are sim-

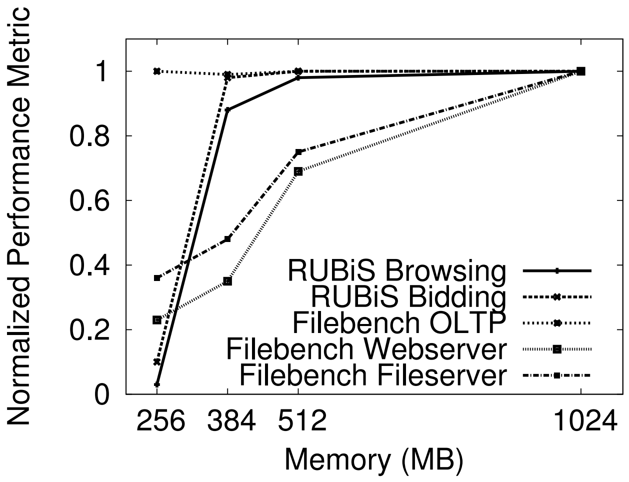


Figure 4.2: Impact of *memory limit*.

ilar to the ones for CPU. Once again, we used limit (specified in MBs) as the control parameter which guarantees a certain memory allocation and no more. Reservation can also be used in conjunction to preserve minimum performance of the workload [KRG$^+$12].

4.2.3 Impact of Memory Allocation

As in CPU, we tested the effects of memory allocations on the same five workloads on ESX. Figure 4.2 shows the normalized performance of these applications as we vary the memory limit from 256 MB to 1 GB. The CPU allocation was kept at a sufficient level to avoid saturation, and there was no I/O contention at the storage. In case of the RUBiS browsing mix workload, performance improves sharply between 256 MB and 512 MB, and remains almost flat afterwards. This behavior can be attributed to the fact that working set size of the workload fits into the memory after a certain allocation. A similar observation can also be made in case of the RUBiS bidding mix workload, where the working set fits within 384 MB of memory. The Filebench-OLTP workload shows almost no memory dependency and the performance remains flat when the VM's memory allocation ranges from 256 MB to 1 GB. The performance of the Filebench webserver and fileserver rises gradually as more of the working set fits in memory.

Overall, these workloads show varied behavior. Some are insensitive to VM memory allocation, while others show either a sudden or gradual increase in performance as the entire or an increasing fraction the working set fits in memory.

4.3 Storage

Strict performance isolation and guaranteed I/O allocation in virtualized environments is challenging because storage arrays are accessed in a distributed manner and the allocation is not under direct control of the hypervisor [GAW09, GMV10]. Currently there are no widely available mechanisms for strictly partitioning I/O bandwidth across multiple VMs. Most virtualization solutions provide a mechanism to prioritize I/O requests from different VMs at the level of I/O scheduler which directly impacts the I/O performance. However, relative prioritization alone is insufficient to model the influence of disk I/O resource on application performance which ultimately depends on the disk I/O bandwidth actually made available to the application VM. The contending I/O volume due to concurrently running VMs on the same host has a direct influence on the resource availability, especially in case of shared storage. So, a parameter to incorporate contention is a must to address the effects of storage I/O on application performance. I/O contention can be controlled or modeled in several ways: measuring or controlling the number of I/Os/sec, MB/sec or I/O latency. We specifically tried with two candidates - *Competing Disk I/O Operations Per Second (CDIOPS)* as well as *VM I/O latency.* In the following two sections, we elaborate on the details of each metric and explain why we pick up VM I/O latency as a winner over the other choices.

I/O Type	Postmark TPS	CDIOPS	I/O latency [ms]
Seq	49	6016	48.08
Seq	52	8483	48.43
Seq	46	8303	46.14
Rand	13	154	77.14
Rand	13	155	70.06
Rand	11	165	81.9

Table 4.1: Comparison between CDIOPS vs. VM I/O latency for modeling I/O contention.

4.3.1 CDIOPS

CDIOPS is the sum of all contending disk I/Os (at a given moment) from other virtual machines (excepting the target VM itself) sharing the same storage. Although, we found it to be quite effective in our initial work [KRDZ10], on subsequent analysis two majors drawbacks were found. First, obtaining the true CDIOPS value when using shared networked storage requires explicit communication either with other hosts or the storage device; this may not be feasible or if so, would incur substantial overhead to keep the information up-to-date. Second, when there is high variance in I/O sizes from competing workloads, the CDIOPS metric can be substantially inaccurate in capturing the actual I/O contention. Large I/O requests would keep the CDIOPS low while causing high device latencies for all VMs. Finally, even the sequentiality characteristics of competing I/O can lead to inaccuracies when using a single CDIOPS value for modeling sequential versus random I/Os which have different costs.

To illustrate this limitation of using IOPS for modeling I/O contention, we ran the Postmark [Kat97] benchmark in a VM running on a ESX host and generated I/O contention using *fio* [fio] on a different VM. We fixed the I/O size at 4KB and and issued 4 outstanding I/Os at a time. Keeping the CPU and memory allocation levels constant, we configured the fio VM to issue either random or sequential I/O. We record the data for three different instances for each type. Table 4.1 reports the VM

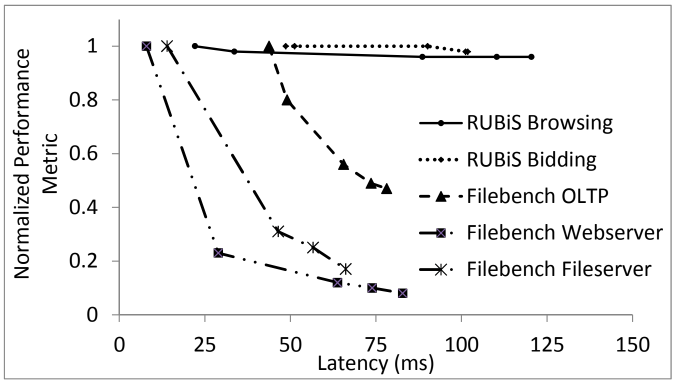


Figure 4.3: Impact of *VM I/O latency*.

storage latency, CDIOPS, and the resulting transactions-per-second (TPS) of the Postmark VM. When the competing I/O is sequential, in spite of the higher CDIOPS values, the application performance is better than the case when the competing I/O is random and the CDIOPS values are lower. This simple experiment clearly indicates the inadequacy of the CDIOPS as a measure of I/O contention. Similar example can be constructed for bandwidth (in MB/sec) based modeling by using small and large sized I/Os.

4.3.2 VM I/O Latency

VM I/O latency is the storage I/O latency observed by the target VM. We refer to VM storage I/O latency or storage I/O latency as VM I/O latency henceforth. VM I/O latency has broader acceptability as it directly reflects the impact of VM I/O contention irrespective of its complexity due to different I/O sizes, sequentiality etc. Moreover, latency can be easily measured from the hypervisor hosting the target VM, no matter whether the storage is networked or local. Table 4.1 demonstrates that using the VM I/O latency more accurately reflects the performance impact of I/O contention on the Postmark workload.

To understand how VMs behave as I/O contention (as captured by I/O latency) varies, we ran each of the five applications in one of the VMs (*appVM*) and ran the competing I/O fio workload in another VM (called *fioVM*) both sharing the same storage. The load is varied from the fioVM to create different levels of I/O contention and cause different I/O latencies perceived by the appVM. Figure 4.3 shows normalized performance as the I/O latency is varied from 20 ms to 120 ms. Most of the applications suffer significant performance degradation when the average I/O latency seen by the appVM increases. The RUBiS Bidding and Browsing workloads generate very few number of I/Os (because their working set is small and fits well in the memory) which made them largely insensitive to the I/O contention [KRG$^+$12].

4.3.3 Xen-specific Parameters

In Xen, all disk I/O for a particular VM is attributed to its corresponding *blkback* process running inside dom-0. *Ionice* values can be assigned to blkback processes to adjust their relative priorities in disk I/O scheduling. Thus, the *disk I/O priority* generic parameter is instantiated as the Xen-specific *IONICE* parameter for each driver domain blkback process. I/O operations from all VMs can be conveniently measured using the *xentop* tool, which provides cumulative statistics of total number of read and write requests separately from each domain. We used *ionice* in conjunction with *CDIOPS* for our initial publication [KRDZ10]. However, no storage latency metrics were found in the xentop tool, at the time we were working with xen.

4.3.4 ESX-specific Parameters

I/O latency can be measured in ESX hypervisor with the use of *esxtop*. Moreover, I/O shares provide control over relative I/O prioritization in case the LUN queue is saturated. Recently, some techniques have been proposed to control I/O latency seen by a VM in ESX environment. Techniques like PARDA [GAW09] and mClock [GMV10] have been proposed to offer better I/O scheduling inside the hypervisor. With PARDA, each virtual disk can be assigned an I/O share value which determines the relative weight of the I/Os from this virtual disk as compared to others. mClock provided additional controls of reservation and limit to control VM latency. Given the lack of access to the source code of ESX and these technologies, we used the degree of contending workloads to control I/O latency for our experiments. This gives us the same model although with more effort. In future, we plan to explore some of these soft controls (reservations, shares, limit) to vary I/O latency for VMs in our experiments.

4.4 Summary

In this chapter, we identified the key allocation parameters (controllable/observable) that effectively characterize application performance. On next chapter, we experimentally corroborate that these chosen set of relatively few number of parameters indeed ensure accurate performance modeling. Moreover, the micro-analysis in this chapter shows that the performance of various applications varies in a non-linear and complex manner as the allocation of resources for the VM is changed. We also noticed that the relationship with respect to one resource is dependent on the availability of other resource as well. For an example, RUBiS bidding workload is seen to be almost neutral to I/O latency in Figure 4.3. While this observation holds true for most of the memory levels, under low memory allocations (256 MB) the performance is impacted by the changes in VM I/O latency. When operating on 256 MB, the performance changed from 1.5 requests/sec under high latency (80-100 ms) to 6 requests/sec under low latency (25-35 ms); an increase of 300%. On the other hand, the performance remains stuck at 59-60 requests/sec when operating under 1024 MB, no matter what the VM I/O latency level is. This sort of behavior clearly emphasizes the level of complexities exhibited by VM-hosted workloads.

CHAPTER 5

APPLICATION PERFORMANCE MODELING

Accurately modeling behavior of the virtualized applications is a key to enable automated VM sizing or revenue driven resource allocation in cloud data centers. The task of modeling is non-trivial due to non-linear dependence of performance on resource levels and the complex influence of contention; as demonstrated in Chapter 4. Moreover, data centers typically host dissimilar applications with widely-varying characteristics on a single physical node. Modeling with respect to a specific type of workloads may not be suitable due to this wide-scale heterogeneity. It is important to find modeling tools which are application agnostic and can characterize applications behavior without collecting too many informations from the applications itself. In this chapter, we discuss the following things. First, we depict the overview of the architecture we are targeting. Second, we show how the the models are trained. Third, we experimentally verify why simple regression models do not work in our environment whereas advanced machine learning tools e.g. Artificial Neural Network (ANN) and Support Vector Machine (SVM) turn out be the best candidates. Fourth, we show how even sophisticated tools e.g. ANN and SVM can fail to deliver higher prediction accuracy for certain complex workloads. We analyze the root cause of why direct applications of ANN and SVM are still insufficient and propose improved use of those tools to substantially increase modeling prowess. Lastly, we thoroughly evaluate our modeling optimization including its prediction accuracy, robustness, and overhead.

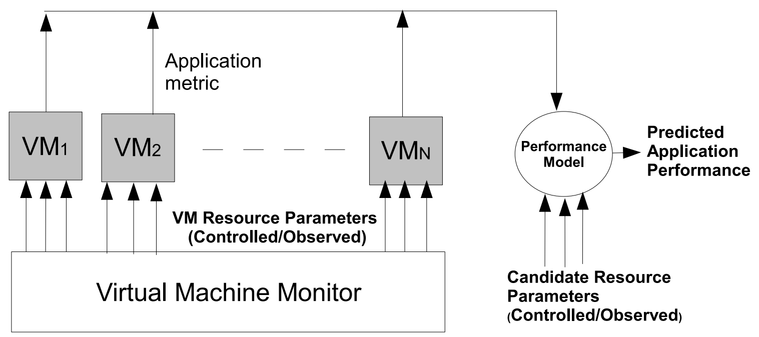

Application metric
VM1
VM2
VMN
VM Resource Parameters (Controlled/Observed)
Virtual Machine Monitor
Performance Model
Predicted Application Performance
Candidate Resource Parameters (Controlled/Observed)

Figure 5.1: Overview of approach.

5.1 Architectural Overview

We portray how modeling can be used for a virtualized host in Figure 5.1. The virtual machine monitor is responsible for allocating basic resources such as CPU cycles, memory capacity, and disk bandwidth. At a high level, allocating a specific share of physical resources to a VM results in a specific performance that is measurable using application-specific performance metrics such as response time and/or throughput. The *Performance Model* for any VM is built by recording application performance metrics under certain combinations of those parameters. This procedure is called *Model Training.* A detailed description is presented in section 5.2. After the training, the model is queried to forecast performance that will be achieved under certain values of candidate parameters. As discussed in Chapter 4, the input set of parameters can either be *observed* or *controlled* easily by system administrator and can be used to achieve a target performance for the virtualized applications in a dynamic environment where resource consumption characteristics or target SLA deliverables of any application are subject to change.

5.2 Model Training

During the training process, a machine learning model gradually tunes its internal network by utilizing the *training data set*. The accuracy of any model is contingent upon selection of a proper training data set and is evaluated using a separate, non-identical *testing data set*. Briefly, the training starts with a boot-strapping phase which requires system administrators to identify the best-case and worst-case resource allocation considered feasible across each resource dimension (CPU limit, memory limit, and virtual disk I/O latency/CDIOPS) for the workload on the target hardware. The input parameter set is then chosen by first including these boundary allocation values and selecting additional allocation values obtained by equally dividing the ranges between the lowest and highest values across each resource dimension. The input parameter set and the corresponding output parameter set (obtained by running the workload on the target system) are chosen as the initial training data set. Additional allocation values (chosen at random) and corresponding output values are collected for populating the testing data set.

After this initial training, the modeling accuracy with the initial training data set is measured by predicting for the testing data set. If satisfactory accuracy (defined an administrator chosen bound on prediction error) is achieved, the training process concludes. Otherwise, additional allocation values are then computed by preferentially varying highly correlated input parameters (based on the correlation coefficient calculated using any statistical tool) by further subdividing the allocation range with the goal of populating the training set with allocation values that represent the output parameter range more uniformly. Additional constraints allow removing values from the training data set to address over-fitting.

5.3 Evaluating Alternative Modeling Techniques

Given the range of behaviors of virtualized applications, identifying techniques that can adequately model them is a formidable exercise. We examine the suitability of several regression techniques that have been used to model application behavior in a non-virtualized systems. Observing the impotency, we probe employing advanced machine learning techniques e.g. ANN. For evaluating model accuracy, we uniformly use the *percentage prediction error* when the model is applied for predicting application performance. Following benchmarks were used for experiments related to this section.

Sysbench-CPU. It's a CPU intensive benchmark from the SysBench [sys] package which consists of configurable number of events that compute prime numbers from 1 to N (user-specified). The benchmark reports the average event handling time which we used as the performance metric. As expected, this benchmark is sensitive to CPU allocation, but insensitive to memory allocation and I/O contention.

Memory-intensive benchmark. We created a micro-benchmark that allocates a large array in memory and continuously writes to random elements of that array. The application performs a fixed number of operations (user-specified) and reports memory-operations-per-second (MOPS). A 1 GB sized file in dom-0 (for Xen platform) was configured as the SWAP virtual block device of the benchmark's VM. This simple workload shows complex non-linear behavior with respect to CPU, Memory allocations as well as to I/O contentions and I/O shares.

Postmark. PostMark [Kat97] is a disk I/O intensive benchmark which models e-mail systems, electronic news, and e-commerce systems. It creates a number of files and performs append, create, delete, and truncate operations on the pool of files. The benchmark reports Transactions Per Second (TPS) as the performance metric. We configured the benchmark to create a data set of size 1.5 GB and perform 10000 transactions. The configures workload showed linear dependence on CPU and memory allocation, and I/O contentions.

Sybench-OLTP. We used the online transaction processing (OLTP) benchmark of SysBench [sys] suite with a MySQL-based database setup. This benchmark utilizes all the three types of resources intensively. We created a table size of 2GB and configured the benchmark to perform 10000 database transactions. The benchmark reports the transactions-per-second (TPS) as the performance metric. This benchmark also demonstrates quite complex behavior to all the three resource types.

The experiments were conducted in xen hypervisor; we used a Dell Optiplex 755 dual core Intel Pentium 4 machine with 2 GB of physical memory running Xen-3.2.0 and and Linux VMs. All VMs including dom-0 ran Linux Kernel-2.6.18.8-xen. At any instant, dom-0 could use one or more cores that were available. Guest machines were restricted to use a single core with the choice of the specific core made at runtime by the VMM, a default Xen option. The VMs used physical partition backed virtual block devices for storage on the same 7.2K RPM SATA disk drive.

Initial experiments indicated that inadequate CPU allocation in dom-0 can adversely impact application performance. In our setup, we ensure at least 25% CPU allocation for dom-0 and did not impose any upper bound on the CPU usage for dom-0. In addition, dom-0 memory allocation (512 MB) was kept constant for all the experiments. To emulate disk I/O contention, we created an additional VM with 256 MB of memory and 20% CPU CAP, which ran an application issuing random reads to large files at varied IOPS values. To obtain training and testing data, each benchmark was run thrice for each input parameter configuration, and an average value of the performance metric was chosen as the output parameter value.

Benchmark	**Sysbench-CPU**			**Memory**			**Postmark**			**Sysbench-OLTP**		
Modeling	*% prediction error*			*% prediction error*			*% prediction error*			*% prediction error*		
Technique	**avg.**	**med.**	**90p.**	**avg.**	**med.**	**90p.**	**avg.**	**med.**	**90p.**	**avg.**	**med.**	**90p.**
Regression-L	24.90	20.12	54.88	19.87	20.24	34.87	6.04	4.73	11.42	23.95	17.91	50.54
Regression-Q	21.69	17.81	48.88	8.66	6.47	19.36	6.27	5.09	11.19	73.51	53.12	195.49
Regression-LI	21.89	19.35	49.31	19.80	16.71	37.19	6.58	5.71	12.60	71.36	46.31	213.53
ANN	11.50	6.65	29.60	2.50	1.16	6.10	7.31	3.34	16.24	8.48	4.24	21.95

Table 5.1: Prediction error statistics for the regression and ANN techniques

5.3.1 Regression Models

We investigate several regression types:

Regression linear(L) is the simplest of the regression family which attempts to establish a relationship between the output and the input parameters by considering only first degree terms of the input variables whereas **regression quadratic(Q)** allows both first order and second order terms for the input variables. To capture if certain inputs have any combined influence on the application output (e.g. Memory and I/O Latency), we applied the **regression linear interactive(LI)** technique which combines first degree of inputs with pairwise interactive terms. We used the *R* statistical package [R] to do the regression analysis.

Table 5.1 summarizes the median, average, and 90th percentile for prediction errors across three regression models that we examined. Identical set of workloads and training set and test set were used for all the experiments.

Almost all the cases, the prediction errors are quite high to be considered acceptable. This experience with regression models leds us to hypothesize that it might be extremely difficult, to create a conventional mathematical model which can predict a virtualized application's performance with acceptable accuracy. These models primarily employ curve-smoothing techniques to fit the training data which may not help in capturing behaviors when output changes non-smoothly in different ranges of the input parameters. A technique is needed which is adaptive and efficient in modeling complex non-linear relationships between outputs and inputs. Evolutionary approaches such as artificial neural networks and support vector machines provide such an ability.

5.3.2 Artificial Neural Network Models

Artificial neural networks (ANN) [Sar94] are advanced non-linear statistical modeling tools based on biological neural networks. The input and output variables of the ANN can be separated by multiple layers each of which has a configurable number of hidden neurons. The number of hidden layers and hidden neurons depends on the number of input and output variables and the complexity of their inter-relationships. There are other internal parameters which need to be tuned as well - activation function, neuron weights etc. We tuned each of these internals for getting the best out of ANN for our environment.

After careful tuning, we apply ANN on our chosen set of workloads. The error values for ANN in Table 5.1 indicate that it is able to adequately model the performance of all the benchmarks providing median error in the range 1.16%-6.65%. These set of initial results clearly indicate the suitability of ANN in VM-hosted application modeling. However, we will see in next section that even a cautiously tuned ANN model which applied directly delivers poor prediction accuracy when applied to more complex workloads.

5.4 Optimizing Machine Learning Models

In this section, we demonstrate that simple application of ANN-based modeling can produce large modeling errors when applied directly for realistic data center appli-

Benchmark	Training points	Testing points
RUBiS Browsing	160	79
RUBiS Bidding	198	99
Filebench OLTP	135	75
Filebench Webserver	160	80
Filebench Fileserver	68	68

Table 5.2: Training and Testing data set sizes.

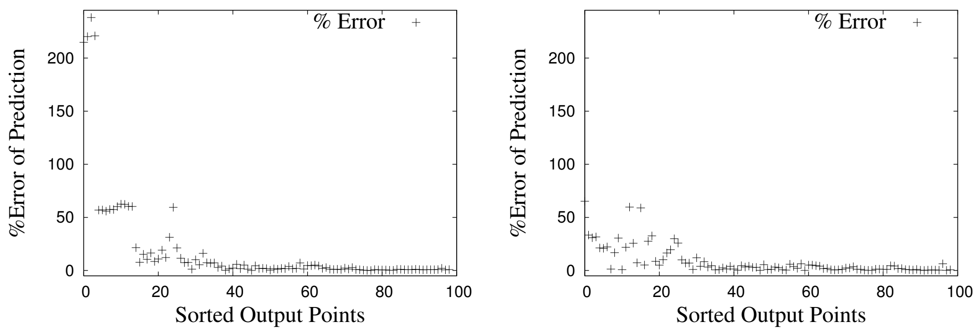
Benchmark	% Avg.	% Med.	Stdev.	90p.
RUBiS Browsing	68.57	5.23	119.73	340.00
RUBiS Bidding	19.30	2.29	45.86	60.18
Filebench OLTP	11.59	8.82	12.63	21.08
Filebench Webserver	19.85	12.88	30.36	38.60
Filebench Fileserver	12.89	6.80	18.64	28.78

Table 5.3: Prediction errors when using a single ANN Model.

cations under a wider span of resource allocations. We introduce the use of another powerful machine learning model, Support Vector Machine (SVM) which has gained more popularity recently. We encounter that it has similar limitations as ANN when used directly for modeling. We probe the root cause of this limitation and propose improved use of those tools to substantially enhance modeling accuracy. For the following evaluation, RUBiS and Filebench workloads have been used and the all the pertaining experiments have been carried out in an ESX testbed (Chapter 4). The number of training and testing data points for each workload is shown in Table 5.2.

5.4.1 Limitations of a Single Global Model

Table 5.3 summarizes error statistics when using ANNs for modeling a set of workloads. We note that prediction errors can be high in some cases, for instance, the RUBiS workloads. We registered identical observations by applying SVM modeling as well. Further analyzing the data revealed that large errors were mostly concentrated in a few sub-regions of the output value space, indicating a single



(a) Single global model (b) Sub Models

Figure 5.2: % Error in prediction for points sorted based on obtained performance for the RUBiS bidding mix benchmark.

model's inability to accurately characterize changes in application behavior as it moves across critical resource allocation boundaries. We demonstrate this behavior in Figure 5.2(a) where We plot the % error in performance prediction across different testing points of RUBiS bidding mix benchmark (specified by resource allocation and I/O latency levels) when sorted by actual obtained performance in *requests/sec*. Given the resources available in today's servers, the multi-dimensional input parameter space and the corresponding output space can both be large. Consequently, accurately characterizing an application's performance with a complex relationship to multiple resource parameters in different portions of the parameter space using a single model proves difficult.

5.4.2 Creating Multiple-Models with Sub-Modeling

To overcome the limitations posed by a single model, we explored the use of multiple models that target specific regions of the input parameter space. Our proposed *sub-modeling* technique divides the input parameter space into non-overlapping sub-regions and builds individual models for each sub-region. For making predictions, specific sub-model(s) is (are) chosen based on which sub-region(s) the parameters of the prediction request fall into. One approach to sub-modeling is sub-dividing the space into several equal-sized regions. However, this seemingly simple approach is inadequate. First, it is difficult to determine how many sub-models to use. If the partitioning of the input space is too coarse grained, the sub-models may not improve prediction accuracy; if it is too fine grained, it may lead to an unmanageably large number of sub-models, making it impractical to create sufficient training data points for each to ensure accurate sub-models. Second, since applications behave non-linearly and non-smoothly with respect to resource allocations, merely building sub-models for equally divided regions may not always be effective in isolating and capturing unique behaviors.

To create robust sub-models, we employ classical clustering techniques whereby the data points are separated into clusters based on a chosen indicator parameter. We used an improved version of K-means clustering technique (*pamk* function in the fpc [pam] package of R [R]) that automatically identifies the optimal number of clusters based on the observed values. We used *application output values* and *prediction error values* (from using the global model) as two choices for the clustering indicator parameter. To verify that the clustering results were useful, we checked whether the cluster boundaries can be clearly identified based on the input parameter values. In other words, a well-defined cluster should be defined by continuous ranges in the input parameter space. Next, we demonstrate that output-value based clustering indeed produces well-defined regions at the input space with negligible overlapping.

We first report the number of clusters and the degree of overlap in consecutive clusters for each benchmark in Table 5.4. If the total number of clusters is n, the number of consecutive pairs of clusters is *n-1*. Sub-modeling is only viable when there is less overlap in the input dimensions of the clusters formed. We consider two consecutive clusters overlap only if one or more points in both the clusters overlap in

Benchmark	**# clusters**	**% Overlap of consecutive clusters**
RUBiS Browsing	2	1.25
RUBiS Bidding	4	0.37
Filebench OLTP	8	1.09
Filebench Webserver	2	0.63
Filebench Fileserver	2	1.47

Table 5.4: Number of clusters and the average % overlap between consecutive cluster pairs, measured based on the Jaccard coefficient.

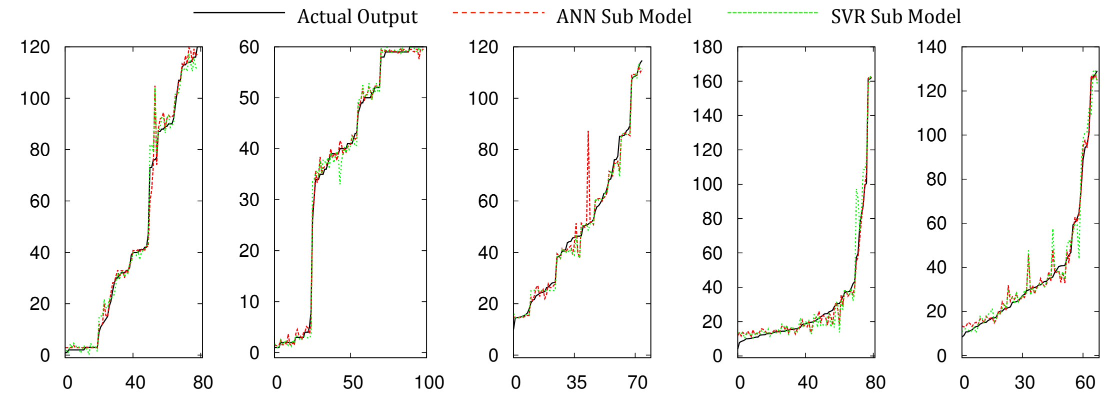
all three input dimensions (CPU, Memory and I/O latency). As we see, the output-value based clustering is able to produce well-defined clusters as the average overlaps for all the benchmarks is around 1%, as calculated using the Jaccard coefficient.

After the clustering stage, we segregate the training data points into buckets based on cluster boundaries and build separate sub-models corresponding to each cluster.

Predicting for a given resource assignment entails checking the input parameter values and determine which sub-model to use. When clusters do not overlap, for points in the boundary regions, we use ensemble averaging of the two consecutive clusters that these points straddle. If clusters overlap in the input parameters space; we use one of two methods to identify the model to use. The first method uses ensemble averaging of predictions using all the overlapping sub-models. The second method coalesces the overlapping clusters and builds a single sub-model for the merged cluster. In general, for applications with high prediction errors either distributed across the entire parameter space or simply concentrated in a single sub-region, the sub-modeling technique can help reduce prediction errors substantially in comparison to a single global model over both ANN and SVM techniques. We demonstrate the effectiveness of the output value based sub-modeling optimization in Section 5.5.

Apart from evaluating clustering based on *output values*, we also performed clustering based on *prediction error values* from the single global model. This choice was motivated by the skewing of high prediction error values towards low allocation values for several of the applications (e.g., in the case of RUBiS bidding mix). At low resource allocation, the sensitivity of the error computation with respect to predicted values is higher when using a single model for the entire range because the application performance drops significantly in this range for most workloads. For instance, requests/sec for RUBiS workloads drops to very low values when the memory assignment of the VM nears 256 MB. Figure 5.2(a) confirms that the large errors are concentrated in the lower output region of RUBiS bidding mix which corresponds to the application output with less than or equal to 256 MB. We made a similar observation for the RUBiS browsing workload. We created multiple models based on the clustering results on the % prediction error values obtained from the global model. Figure 5.2(b) demonstrates that error based clustering can substantially reduce the % errors in the lower output regions. In fact, the 90th percentile errors for RUBiS browsing mix dropped from 340% to 27.28%. For the bidding mix, the reduction is from 60.18% to 25.80%.

In general, if large errors are concentrated within a specific region of input parameters space, sub-models based on prediction error values from a single global model become valuable. However, this trend does not hold across all the workloads. Except the RUBiS workload mixes, error-based clustering did not lead to well-defined sub-regions and resulted in high overlap in the corresponding input parameter spaces, rendering the clusters practically unusable. On the other hand, sub-modeling based on output values produced robust models across all the workloads we examined. We evaluate output value based sub-modeling in more detail in the following section.



(a) RUBiS Browsing (b) RUBiS Bidding (c) Filebench OLTP (d) Filebench Webserver (e) Filebench Fileserver

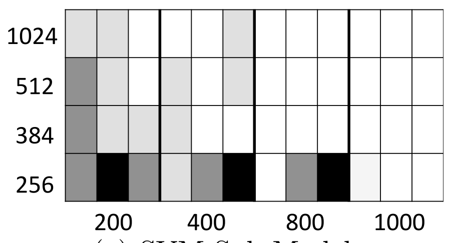
Figure 5.3: Actual performance and predictions using ANN and SVM based sub-models. The x-axis enumerates data points sorted by increasing performance values. The y-axes represent performance (requests/sec for RUBiS and operations/sec for Filebench respectively).

5.5 Evaluation

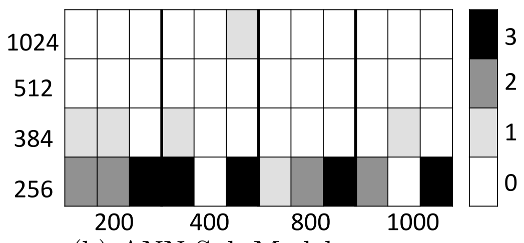
We present error statistics for sub-modeling based on output value clustering for each benchmark in Table 5.5. For all workloads and modeling techniques, sub-modeling successfully reduces the mean, median and 90^{th} percentile of errors when compared to using a single global model. Interestingly, even simple regression models achieved higher accuracy using sub-modeling for most of the benchmarks. This uniform trend clearly indicates that the complexity of these applications cannot be reduced to a single consistent representation as a global model would be forced to adhere to. As the sub models are confined to only a smaller portions of the entire application performance metric space, these models can offer constrained, but more accurate, representations of behavior. Although, using sub modeling with clustering increase the attractiveness of simpler regression based models, the power and utility of using the machine learning based models is evident when we consider error variance. We found, in particular, that the effectiveness of regression based modeling is tied to

	Modeling	**Regression-L**			**Regression-Q**			**Regression-LI**			**SVM**			**ANN**		
Bench		*% prediction error*			*% prediction error*			*% prediction error*			*% prediction error*			*% prediction error*		
mark		**avg.**	**med.**	**90p.**	**avg.**	**med.**	**90p.**	**avg.**	**med.**	**90p.**	**avg.**	**med.**	**90p.**	**avg.**	**med.**	**90p.**
RUBiS	Global	397.08	36.05	1366.83	323.81	23.94	867.77	327.27	35.47	1090.05	55.78	12.78	142.67	68.57	5.23	340.00
Browsing	Sub-Model	106.11	17.74	340.60	42.78	4.16	145.01	103.39	18.78	332.80	21.51	5.19	68.44	**15.95**	**2.94**	45.87
RUBiS	Global	314.05	30.87	1096.58	110.53	21.81	341.54	317.52	29.37	1126.94	68.86	14.21	100.67	19.52	1.97	80.00
Bidding	Sub-Model	9.19	2.43	34.72	8.37	1.94	34.64	9.10	2.39	35.08	7.38	**1.79**	**22.50**	**7.18**	**1.79**	30.01
Filebench	Global	18.33	15.75	33.41	17.82	12.63	35.58	14.02	10.41	28.74	7.40	4.70	13.14	11.59	8.82	21.08
OLTP	Sub-Model	6.31	2.74	13.42	9.47	3.44	29.48	7.55	3.57	10.95	**5.65**	**2.90**	15.19	5.90	3.02	**13.66**
Filebench	Global	76.03	62.99	138.31	50.54	44.91	96.71	51.44	33.41	113.78	25.02	15.57	53.50	19.85	12.88	38.60
Webserver	Sub-Model	32.13	21.85	53.01	27.83	20.31	55.09	29.85	20.72	55.79	19.38	12.25	36.02	**15.57**	**8.49**	**33.07**
Filebench	Global	49.02	30.70	98.18	54.94	21.20	123.36	27.59	21.10	55.75	13.87	9.38	24.56	12.89	6.80	28.78
Fileserver	Sub-Model	21.34	13.63	46.48	16.22	8.91	42.64	18.35	11.26	38.91	10.89	6.72	25.14	**10.60**	**5.57**	**23.22**

Table 5.5: % Error statistics (average, median, and 90^{th} percentile) for different modeling techniques by using global models and sub models.



(a) SVM Sub Model



(b) ANN Sub Model

Figure 5.4: Error Distribution of RUBiS Bidding Mix when using sub models for prediction. The X axis represents CPU Limit (MHz); the Y axis represents Memory Limit (MB). Each box is divided into three columns - representing low, medium, and high (from left to right) VM I/O latency. Error value 0-3% = white, 3-9% = light grey, 9-27% = dark grey, 27% and more = black.

the effectiveness of clustering. When the clusters are bigger, regression models typically perform poorly (average error of 42.78% for R-Browsing using regression-Q) . On the other hand, ANN and SVM are able to provide reasonable accuracy in all the scenarios (average errors between 5.90% to 15.95% and 5.65% to 21.51% respectively, using sub-modeling). To provide a more detailed view of accuracy when using sub-modeling with ANN and SVM, Fig 5.3 shows the actual and predicted output values for each of the testing points for benchmark. For clarity, we present the actual and predicted values sorted in increasing order of actual performance obtained. Predicted values closely follow the actual values in majority of the cases.

5.5.1 Measure of Confidence

While summary error metrics are valuable indicators, the distribution of error values across various combinations of input parameters can be a useful guide to the system administrator while choosing a specific set of allocation values. Particularly, if it is known that in certain regions of the resource allocation space, the model error tends to be higher, system administrators can choose to compensate with a greater

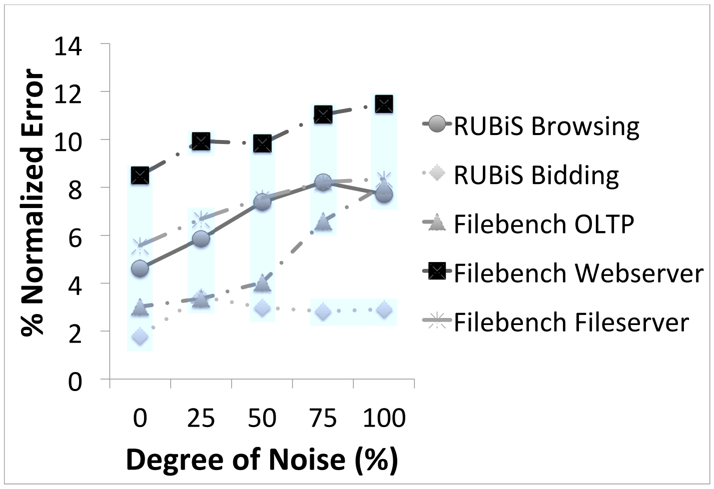


Figure 5.5: Change in median error when noise is introduced in the training data set.

degree of over-provisioning. Figure 5.4 shows a heatmap of error distributions of RUBiS bidding mix workload using sub models as an example to illustrate this point. As evident, the higher errors are concentrated towards the region of low memory allocations (256 MB), informing administrators of caution while choosing allocations surrounding that memory region based on the model predictions of performance. While we do not address this in our work, it is possible to modify predictions to be conservative for regions prone to higher prediction errors within the modeling framework itself once the error distribution is known using a sample set of testing points.

5.5.2 Robustness to Noise

Our experiments were performed in a controlled environment. However, production environments can pose additional challenges, especially with respect to performance variability due to noise. To evaluate the applicability of our models under noise, we simulated noise in 10% of our training data set points; we modified their observed performance values by randomly varying them within a fixed percentage specified by

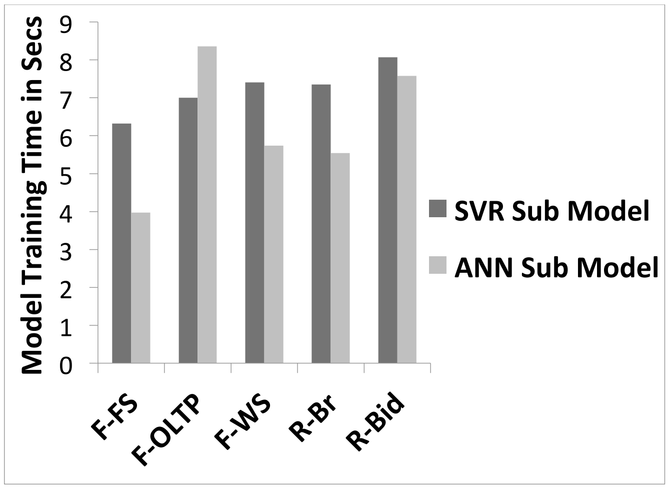


Figure 5.6: Training time. Benchmarks appear in increasing training data set size from left to right

the *degree of noise*. For example, if the degree of noise is 20%, the original value was changed either by +20% or -20%. For clarity, the testing data set was unmodified. In Figure 5.5, we report the % median modeling errors for each benchmark as we vary the degree of noise when using the ANN-based sub-modeling technique. Trend lines indicate a largely linear dependency between noise and error. Additionally, a key take away from this analysis is that even after 100% modifications of as much as 10% of the training data points, the modeling accuracy does not suffer substantially. In case of the RUBiS bidding workload, the % median error just increased from 1.79% (in case of perfect data) to 2.91% (in case of ±100% change of 10% of the points in the original training data set). In case of Filebench-Webserver, the degradation is from 8.49% to 11.48%. This validates one aspect of robustness under potential noise as is possible in a production environment.

5.5.3 Modeling Overhead

Training a model is usually proportional to the number of points in the training data set. We show the modeling time for each type of modeling in Figure 5.6, with benchmarks sorted by their training data set size, appearing from left to right. As we can see, the modeling overhead for the largest training data set (198 points) is limited to 8 seconds using SVM sub-modeling. The training time for ANN is calculated as an aggregate of five different model runs, as required by ensemble averaging. We believe that the presented overhead is sufficiently low to make these models usable in an on-line production environment where a model is (re)trained at regular intervals.

5.6 Summary

In this chapter, we show the limited power of regression models to characterize complex workload behaviors and justify the use of sophisticated machine learning techniques e.g. ANN and SVM for accurate performance modeling. We also demonstrate how even ANN and SVM can sometimes fail to deliver improved accuracy; thereby necessitating the application of multiple sub models. In summary, these new optimizations substantially improve the prediction accuracy and reduce the average and 90^{th} percentile prediction errors from 26.48% and 101.68% respectively (averaged over all applications) for a correctly configured single global model to 11.04% and 29.17% by using sub-modeling with ANN. Similarly, for SVM the average and 90^{th} percentile prediction errors respectively drop from 34.19% and 66.91% for a single global model to 12.96% and 33.46% using sub models. We also emphasize that between ANN and SVM, there is no clear winner. Although Table 5.5 shows that ANN is slightly better in prediction statistics than SVM for most of the workloads, these differences are not statistically significant. Moreover, experimental results confirm that the proposed optimizations are robust to handle noises in the training data set. Also, the training incurs small overhead which makes these usable in practice for online modeling. In next chapter, we show how our proposed modeling techniques can help realizing optimal VM sizing for cloud environment.

CHAPTER 6

VM SIZING

Cloud service providers (e.g. Amazon EC2) charge customers based on the rented computing capacity . For the sake of simplicity, capacity is usually represented using coarse-grained choices (e.g. small, large, and extra-large for standard on-demand instances in Amazon's EC2 cloud service [ec2]) that map to a certain amount of CPU, memory and other resources. These choices have proportional as well as skewed allocation of resources, where one can even get an instance with more CPU and less memory. In private virtualized environments, administrators have more flexibility in assigning the resource allocations for a VM. In either case, it is the customers' responsibility to determine the VM sizes (CPU and memory capacity) that they need to meet application-level performance targets. Given the lack of application-based model customers choose more conservative sizes and over-provision to avoid seeing performance problems. This leads to sub-optimal sizing and higher costs throughout the life of the VM. A fine-grained, tailored sizing of VMs, on the other hand, can allow meeting target performance while minimizing over-provisioning.

In this chapter, we show that given a target application performance metric and a VM I/O latency level available to the application, our performance models proposed in previous chapter can be used to find the optimal CPU and memory sizes. We used a specific I/O latency as an input because it is not configurable in many cloud environments. However, our modeling can even determine a desired I/O latency value so as to minimize the overall cost of the VM. We experimentally demonstrate that, for a range of performance targets across RUBiS Browsing and Filebench webserver workload, the suggested CPU and memory sizes indeed deliver the required performance in all cases.

6.1 VM Sizing Problem Definition

We define the optimal VM sizing problem as follows:

Problem definition: Given a performance target P_{target} and a VM I/O latency $iolat$ and a performance model PM, the VM sizing algorithm generates suggested CPU c and a memory m which are able to meet P_{target} and satisfy the following constraints:

$$\frac{P_{c,m,iolat}}{P_{target}} \geq 1 \tag{6.1}$$

subject to:

$$c_{min} \leq c \leq c_{max} \tag{6.2}$$

$$m_{min} \leq m \leq m_{max} \tag{6.3}$$

$$\frac{P_{c-\delta_c,m,iolat}}{P_{target}} < 1 \tag{6.4}$$

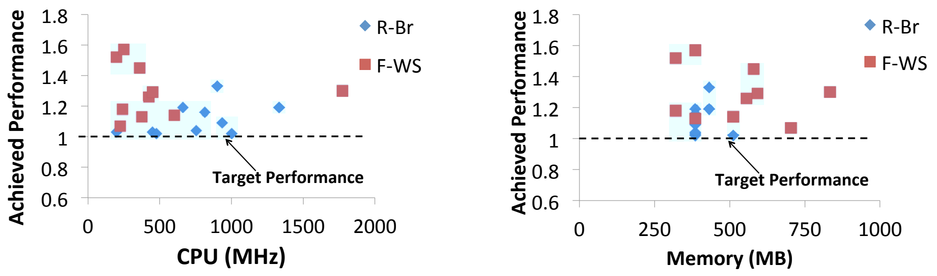
$$\frac{P_{c,m-\delta_m,iolat}}{P_{target}} < 1 \tag{6.5}$$

under the following assumptions:

$$P_{c+\delta_c,m,iolat} \geq P_{c,m,iolat}; P_{c,m+\delta_m,iolat} \geq P_{c,m,iolat} \tag{6.6}$$

$$P_{c-\delta_c,m,iolat} \leq P_{c,m,iolat}; P_{c,m-\delta_m,iolat} \leq P_{c,m,iolat} \tag{6.7}$$

Intuitively, these constraints force us to find a VM size such that less of any resource would make us miss the performance target. Equation 6.1 ensures that application performance for the suggested c, m, and iolat should be at least equal to or greater than P$_{target}$. Equations 6.2 and 6.3 bound allocations to the feasible range; additionally, allocation choices for CPU and memory in these range can only be made in units of δ_c and δ_m respectively. Equation 6.4 guarantees that the performance achieved for a smaller CPU allocation fails to meet P$_{target}$. Similarly, equation 6.5



(a) Suggested CPU allocations (b) Suggested memory allocations

Figure 6.1: Application performance relative to performance target for model-based VM sizing. Each point represents a specific sizing query.

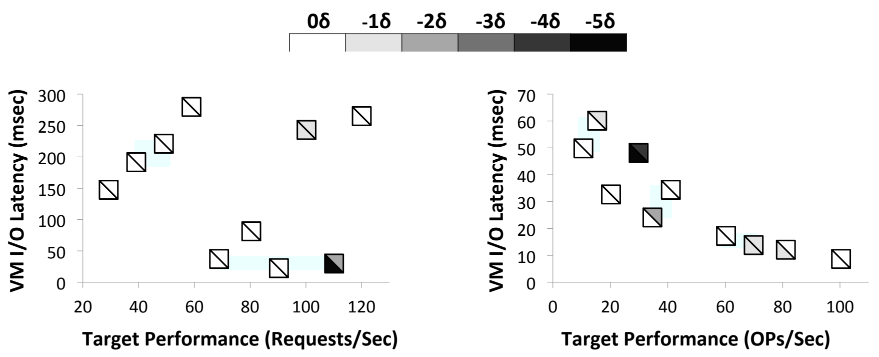
checks for memory optimality. Finally, we assume that allocating additional resources to a VM will not degrade its performance (Equation 6.6) and taking away resources cannot improve performance (Equation 6.7).

6.2 Model-based VM Sizing

We follow a simple approach to VM sizing, i.e., determining the values of c and m. First, assuming a memory allocation of m_{max}, the maximum possible memory allocation, we use binary search on c to determine its optimal value that would allow meeting the application's performance target by querying the performance model PM using the given **iolat (input)**, m_{max}, and c. After the optimal value of c is obtained, we perform a second binary search as above fixing c for various values of m.

To handle modeling inaccuracy, we use a query performance target 10% higher than the actual performance target P_{target}. To accommodate our hardware platform, we used c_{min}=200MHz, c_{max}=2GHz, m_{min}=256MB, and m_{max}=2GB. We use δ_c=100 MHz and δ_m=64 MB in our experiments.

We experimented with two workloads for VM sizing: RUBiS Browsing and Filebench Webserver workloads. We randomly selected 20 performance targets and



(a) RUBiS Browsing (b) Filebench Webserver

Figure 6.2: Optimality of VM Sizing under a target performance and a given VM I/O latency. Each box is divided into two triangles - lower triangle represents optimality of CPU, upper triangle stands for memory. The degree of optimality is determined by the color code (shown above) of each triangle.

VM I/O latency levels as inputs to evaluate the accuracy of our models and sizing algorithm. We validate the results by running individual workloads with the CPU and memory allocation suggested by our sizing technique. VM I/O latency is controlled by issuing I/Os from a contending VM to the same storage LUN.

Figure 6.1 demonstrates that P$_{target}$ is met or exceeded for all the 20 configurations using our sizing technique; P$_{target}$ is normalized to 1 in all the cases. In other words, the charts plot the $\frac{P_{achieved}}{P_{target}}$ for each performance target; essentially showing how far are the experimentally observed performances obtained under suggested CPU, memory allocations from the respective target queries. The X-axis in the two plots marks the suggested CPU and memory sizes for each query. The wide range of suggested sizes indicates that the required memory and CPU may vary quite significantly based on the target performance over most of the available allocation range, underscoring the need for fine grained VM sizing.

To demonstrate the optimality of the suggested allocations, we run each workload with smaller CPU (c-δ) and memory (m-δ) allocations. We deem sizing results as *optimal* with respect to a specific resource dimension (CPU or memory) if a δ

reduction in allocation results in the performance target P_{target} being violated. If P_{target} is met under a reduced allocation, we investigate the degree of sub-optimality by running the workload under varying number of δ reductions. In Figure 6.2, we depict sizing optimality along CPU and memory dimensions separately. Out of 20 randomly selected performance targets for a wide range of chosen VM I/O latency values, the performance models deliver optimal results on both dimensions in 65% cases; rest of the points are within 2δ of the optimal, except for two cases where allocations are sub-optimal by 4δ or 5δ.

6.3 Summary

This analysis indicates that our model-based VM sizing approach can suggest optimal sizes in majority of cases while meeting performance targets. In some cases, we suggested a higher size but that is still better than picking a size without any information. The higher can also be attributed to the fact that we set a higher performance target and we want to be conservative in picking our sizes so that performance is not impacted. This is very critical to increase the confidence of administrators in such a tool and over time one can make the estimate more aggressive. In Chapter 9.1, we will discuss the work still needed to fully realize the potential of modeling-based VM sizing. The next chapter demonstrates how our performance models guide another important data center management task.

CHAPTER 7

DYNAMIC RESOURCE MANAGEMENT

The machine-learning based application performance models, elaborated in chapter 5, help accurately configuring user VMs to meet desired performance objectives when renting resources from cloud service providers. We will now discuss how the models can be the building blocks for online and automatic resource management of virtual machines in data centers.

Data centers are revenue driven and they generate revenue by charging individual client applications according to Service-Level-Agreements (SLAs). Lack of automation and imprecise understanding of the effect of resource usage on applications performance force data centers to use simple SLAs where the customers are charged a flat fee based on the resource capacity they are renting or buying. However, this is not ideal for either customers or server administrators. For customers, there is no easy way to choose an appropriate capacity. Consequently, they either pay more for unnecessary resources or experience performance violation. Administrators, on the other hand, use over-provisioning to avoid the heavy penalties for performance violations. An alternate charging model is performance oriented whereby a client pays rent for application performance. This will help data centers to adopt a resource provisioning scheme where applications are penalized or rewarded in the allocations of physical resources according to the revenue lost or generated by their respective SLAs. It will also boost clients' confidence since they will be paying for exactly the performance experienced by their applications.

To achieve this goal, we envision a framework and process at the data center level that will dynamically partition resources among hosted VMs with the goal of maximizing SLA-generated revenue for the data center. Specifically, we address a deployment scenario where a set of VMs are sharing a set of resources of varying

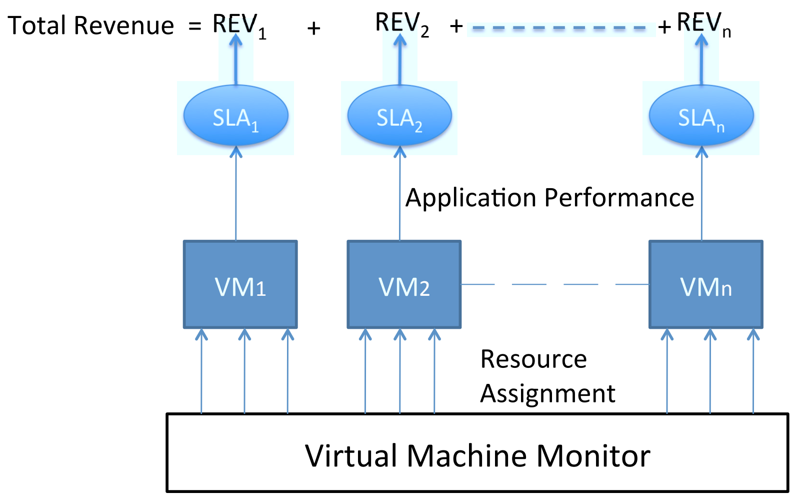


Figure 7.1: Architectural Overview for Revenue Maximization.

types (CPU, memory, disk) and each VM is generating a certain revenue for the data center per unit amount of time at a given performance level. A high-level architectural overview of this framework is depicted in Figure 7.1. The revenue generated by a single VM at any instant is a function of the performance level supported for the application and is defined by its Service Level Agreement (SLA).

Several steps need to be taken to implement a revenue-maximizing resource allocation strategy. The first task is identifying the parameters both for partitionable resources (e.g. CPU, memory) and for non-partitionable resources (e.g. storage, network) that will be the control-knobs for administrators to distribute physical resources. We have dealt with this in Chapter 4. The second task is characterizing the impact of resource allocation on application performance. We evaluated accurate and robust performance model building in Chapter 5. The third task is developing an efficient and effective algorithm which will partition server resources to maximize the collective revenue across all applications at any given instant. This task is non-trivial due to the complex mapping of resource allocations to applications performance and diverse SLA curves. Moreover, complete reallotments of resources from scratch are challenging due to the use of observed VM I/O latency in our prediction model. Since VM I/O latency cannot be effectively controlled, a drastic change in resource allocation can abruptly impact VM I/O latency significantly and thus impact the stability of model-based performance prediction. To minimize the impact of the changing VM I/O latency, we take an incremental approach where allocations do not change significantly within a single resource reallocation operation to reduce the alteration of the VM I/O latency levels observed.

7.1 Modeling Resource Allocation

We now formally model the problem of dynamic multi-resource allocation in virtualized systems to explicitly take into account the influence of both resource allocation and resource competition. The model maps resource allocations of individual VMs to revenues generated in US dollars as dictated by individual SLA functions. Performance models are being contained within this encompassing framework. We identify that the optimal multi-resource reallocation problem is at least NP-hard and that exact solutions are infeasible in practice. The formal problem statement and the proof of NP-hard are described next.

We have already established that the performance of individual virtualized applications are determined by the resource assignments and current competition levels posed by other applications sharing the host. To determine the evenue generatable by a virtualized application at a future instant, the application performance under possible future resource assignments must be determined. The obtained application performance is mapped to the revenue in USD by the given application-specific SLA curve. As complete redistribution is not possible due to the variance of observed VM I/O latency within the prediction framework, administrator-defined parameters, k and δ, serve to bound the maximum change we make to the allocations for

Parameter	Description
n	Number of application VMs
m	Number of resource types
k	Maximum number of times the resource allocation of any VM can be changed for any resource dimension in a single resource reallocation operation
δ	A vector of length m denoting the units of changes in m resource dimensions.
I	Set of application VMs
J	Set of resource types
R^{alloc}	Current resource allocation vector of dimension m×n
R^{total}	A vector of length m for total available resources
$R_{i,j}$	Resource allocation for VM $i \in I$ of resource type $j \in J$
R^{opt}	Optimal resource allocation of vector m×n after the redistribution
S(R)	A vector of n SLA functions mapping the application performance to revenue in USD
REV	Revenue vector of length n
T	Time interval of running reallocation algorithm.
PM	A vector of length n, each member is a separate performance model for one VM App $i \in I$
$\Psi(R_{i,j}, \quad \forall j \in J)$	Revenue for application $i \in I$, where amount of resources allocated to application i is $R_{i,j}$, $\forall j \in J$.

Table 7.1: Description of symbols used in resource allocation problem formulation

each resource type within a single resource reallocation operation. An additional important aspect of our resource allocation problem formulation is that we assume that SLA curves are complex, non-linear descriptions of revenue dependent on the application performance metric and not based on simple priority values.

7.1.1 Problem Formulation

We list the various parameters employed in the resource allocation problem formulation in Table 7.1. The SLA-based optimal resource allocation problem can be formally specified as follows:

"Given $\mathbf{n}$ *application VMs (denoted as set I),* $\mathbf{m}$ *allocatable resource dimensions (denotes as set J), current resource allotments* $\mathbf{R}^{alloc}$*, performance models* $\mathbf{PM}$*, and SLA-based revenue function* $\mathbf{S(R)}$*, determine a set of new resource assignments* $\mathbf{R}^{opt}$ *for the VMs which will result in maximizing the total revenue REV generated across all VMs for certain time interval* $\mathbf{T}$*, given that any change to resource assignment* $\mathbf{R}_{i,j}$*,* $i \in I$*,* $j \in J$ *is bounded by* $\mathbf{-k}\delta$ *to* $\mathbf{+k}\delta$*."*

At each interval T, the problem may be formalized as:

$$Maximize \quad \sum_{i \in I} \Psi(R_{i,j}^{opt}, \ \ \forall j \in J) \tag{7.1}$$

subject to :

$$\textstyle\sum_{i \in I} R_{i,j}^{opt} \leq R_j^{total} \quad \forall j \in J \tag{7.2}$$

$$R_{i,j}^{opt} - R_{i,j}^{alloc} \leq \pm k\delta \quad \forall i \in I, j \in J \tag{7.3}$$

Equation 7.1 maximizes the revenue across all resources. Equation 7.2 restricts the total resource allocation for each resources across all application VMs to be less than the total available resources. Equation 7.3 restricts the resource allocation change of each resource type for each application VM to a maximum of $k\delta$.

The revenue derived by a data center from a particular application VM depends on the SLA and on the performance of the application, which in turn depends on the resources allocated to the application VM. However, all these dependencies are non-linear. Finally, $R_{i,j}$, R^{alloc}, R^{total} and R^{opt} are assumed to be integer values.

Theorem 7.1.1 *The resource allocation problem is at least NP-hard.*

Proof. Let us assume that the function Ψ is a linear summation function as follows $\Psi(R_{i,j}^{opt}, \ \ \forall j \in J) = \sum_{j \in J} A_j R_{i,j}^{opt}$, $\forall i \in I$, where A_js are constants. Let us also assume, $\delta = \infty$. Then the resource allocation problem reduces as follows,

$$Maximize \quad \sum_{i \in I} \sum_{j \in J} A_j R_{i,j}^{opt} \tag{7.4}$$

subject to :

$$\textstyle\sum_{i \in I} R_{i,j}^{opt} \leq R_j^{total} \quad \forall j \in J \tag{7.5}$$

The above problem is the integer knapsack problem, a well-known NP-Complete problem [Ham]. As the knapsack problem can be reduced to a specific reduced instance of the resource allocation problem in polynomial time, we can conclude that this reduced subset problem of the resource allocation problem is NP-Complete. Consequently, with the additional constraints of equation 7.3 the resource allocation problem is at least NP-hard. $\square$

7.1.2 How Expensive is Exhaustive Search?

Exhaustive or brute-force search techniques may be applied to the resource allocation problem to find the most optimal solution. For many NP-hard problems, the small input size allows trivial, brute-force, exact solutions in practice; we examine if this true in the present case. Per our problem formulation, each resource reallocation can assume *2k* different values, k positions for increments and the same for decrements. Comparing *2k* different possible changes in revenue values for each of the n VMs and for each of the m resource types to find R^{opt} will incur an asymptotic time-complexity of $O((2k)^{mn})$ which is infeasible for even small n and k. If we assume that $m = 4$, $n = 10$, and $k = 5$; the time taken to run brute-force search will take 10^{40} time units. Alternate, efficient heuristic solutions are thus needed for realistic deployment.

7.1.3 Other Heuristic Solutions

The resource allocation problem has considerable similarities to the class of classic knapsack optimization problems with a common objective of determining a set of items to include in a sack of finite weight with a goal of maximizing the total value of the sacked items and with the constraint that the sum of all weights should be less than or equal to the capacity of the sack. Specifically, the total available capacity of any type of resource can be treated as the capacity of the sack and the resource assignments of individual items can be mapped to the weights of the items selected to place in the sack, the revenue from each VM corresponding to value of each item. Then, maximizing total knapsack value will map maximizing total revenue. Thus, we can contemplate applying heuristic solutions from the class of knapsack problems to the resource allocation problem.

Despite substantial similarity, the revenue maximization problem has several distinguishing characteristics when compared to the basic knapsack formulation. First, it is multi-dimensional with m number of resource types, which substantially increases the size of the solution search space. Second, because SLA-based revenue functions can be nonlinear, solutions to linear knapsack problems cannot be used as-is to solve our problem. Third and the most important distinguishing feature is that for the sake of system stability, our problem requires incremental resource reallocation, instead of reallocating resources from scratch at every decision time, and such resource change is constrained according to Equation 7.3. This additional constraint makes our problem substantially more challenging, whereby the existing solutions to multi-dimensional, nonlinear knapsack problems [BS02] cannot be directly applied. Finally, the reduction of an instance of our problem to a complex variant of knapsack is possible only by eliminating Equation 7.3. With the addition

Parameter	Description
i	Index for application VMs
j	Index for resource types
$\delta P^{g}_{i,j}$	Gain in revenue for application VM i as resource allocation of type j is increased by δ keeping all other resource dimensions constant
$\delta P^{l}_{i,j}$	Loss of revenue for application VM i as resource allocation of type j is reduced by δ keeping all other resource dimensions constant
MaxGain_j	Maximum Gain obtained for resource type j
MinLoss_j	Minimum Loss incurred for resource type j
MaxNetProfit	Maximum net profit obtained globally i.e. across all VMs and all resource dimensions.
VM_g	VM whose gain is maximum for a specific j
VM_l	VM whose loss is minimum for a specific j
VM_{gg}	VM whose gain is maximum for some j and which is globally selected as a candidate for allocating more resources.
VM_{lg}	VM whose loss is minimum for some j and which is globally selected as a victim for taking away resources.
R_{max}	Resource type for which the net profit is maximized.
i^{j}_{pkval}	Number of times the resource allocation of type j for VM i is increased.
i^{j}_{nkval}	Number of times the resource allocation of type j for VM i is decreased.

Table 7.2: Description of symbols used in the algorithm

of equation 7.3, the problem complexity increases to a level that we believe it is unsolvable using existing approximation algorithms for the known Knapsack family of problems.

7.2 A Heuristic Solution

In this section, we present an algorithm with an acceptable time-complexity for dynamic resource allocation discussed in the previous section. Given a current set of resource assignments for a pool of application VMs, the algorithm aims to find a new set of allocations for each resource which attempts to maximize the revenue at current application demand. We provide detailed descriptions of the parameters used in our algorithm in Table 7.2 and the entire pseudocode is presented in Algorithm 1. We quantify the effect of incremental changes using $\delta P^g_{i,j}$ and $\delta P^l_{i,j}$ which denote the gain or loss in revenue as the assignment of resource type j for application i is increased or decreased respectively by an amount of δ.

Let us assume that the the current resource allocation of resource type j for application i is r which provides a revenue of p dollars. Our previously developed ANN model [KRG+12] is used to predict application output for an allocation of $r \pm \delta$ which is subsequently mapped by S_i to find the corresponding revenue p_δ. The difference between p_δ and p indicates the gain or loss under δ increment or decrement, defined as $\delta P^g_{i,j}$ or $\delta P^l_{i,j}$. We assume that $\delta P^g_{i,j} \geq 0$, i.e., increasing resource allocation to a VM always results in no change or an increase in performance and consequently no change or an increase in revenue for the VM, but can never cause a revenue reduction. Similarly, $\delta P^l_{i,j} \leq 0$ or taking away resources does not cause an increase in revenue.

Our proposed polynomial-time algorithm uses an iterative, greedy approach to revenue maximization. In each iteration, it transfers resources from the VM that offers the least reduction in revenue due to a reduction in resources to the most revenue-generating VM. In doing so, it also chooses the resource type for which the relative gain is maximized.

The main algorithm (`MaxRevenue` Algorithm 1) implements an incremental re-allocation of resources across application VMs. This algorithm is run each time a resource redistribution across VMs is considered; this could be either periodic or based on administrator initiation. The algorithm `MaxRevenue` identifies the VMs offering the maximum gain and minimum loss for δ change of all resource types j (the `for` loop at line 5 of Algorithm 1). In line 6, the algorithm invokes `FindMaxMinVM` (Algorithm 2) to identify the VM that offers the minimum loss of revenue due to loss

Algorithm 1 *MaxRevenue*: Revenue Maximization Algorithm

```
1: while (1) do
2:   $MaxNetProfit = 0$
3:   $reshuff_g = FALSE$
4:   $reshuff_l = FASLE$
5:   for $j = 1$ to $m$ do
6:     Call $FindMaxMinVM$ function to get $MaxGain_j$, $MinLoss_j$, $VM_l$, $VM_g$
7:     $CheckReshuffle()$
8:     $CompareGainNLoss()$ /*Compare gain with the loss */
9:   Call $StepAdjustments()$ to change the $VM_{gg}$ or $VM_{lg}$ if reshuffling is required
10:  if $MaxNetProfit > 0$ then
11:    /* Actual resource distribution occurs */
12:    $R_{VM_{gg},R_{max}} + = \delta$
13:    $VM_{gg_{pkval}}^{R_{max}} + = 1$
14:    $R_{VM_{lg},R_{max}} - = \delta$
15:    $VM_{lg_{nkval}}^{R_{max}} + = 1$
16:  else
17:    /* No gain in net profit, so the algorithm stops */
18:    $break$
```

Algorithm 2 *FindMaxMinVM*: Find VMs with Maximum and Minimum Gain

```
1: OUTPUT: $MaxGain_j$, $MinLoss_j$, $VM_l$, $VM_g$
2: $MaxGain_j = 0$
3: $MinLoss_j = \propto$
4: for $i = 1$ to $n$ do
5:   /* Finding the VM whose gain is maximum */
6:   if $\delta P_{i,j}^g > MaxGain_j$ and $i_{pkval}^j < k$ then
7:     $MaxGain_j = \delta P_{i,j}^g$
8:     $VM_g = i$
9:     $kg = i_{pkval}^j$
10:  /* Finding the VM whose loss is minimum */
11:  if $\delta P_{i,j}^l < MinLoss_j$ and $i_{nkval}^j < k$ then
12:    $MinLoss_j = \delta P_{i,j}^l$
13:    $VM_l = i$
14:    $kl = i_{nkval}^j$
15: return $MaxGain_j$, $MinLoss_j$, $VM_l$, $VM_g$
```

Algorithm 3 $CompareGainNLoss$: Compare gain with the loss

```
1: INPUT: $MaxGain_j$, $MinLoss_j$, $MaxNetProfit$, $VM_g$, $VM_l$, $reshuff$
2: OUTPUT: $VM_{gg}$, $VM_{lg}$, $R_{max}$
3: if $VM_g \neq VM_l$ and $MaxGain_j + MinLoss_j > MaxNetProfit$ then
4:     $MaxNetProfit = MaxGain_j + MinLoss_j$
5:     $VM_{gg} = VM_g$
6:     $VM_{lg} = VM_l$
7:     $R_{max} = j$
8:     if $reshuff_g == TRUE$ or $reshuff_l == TRUE$ then
9:         $break$
```

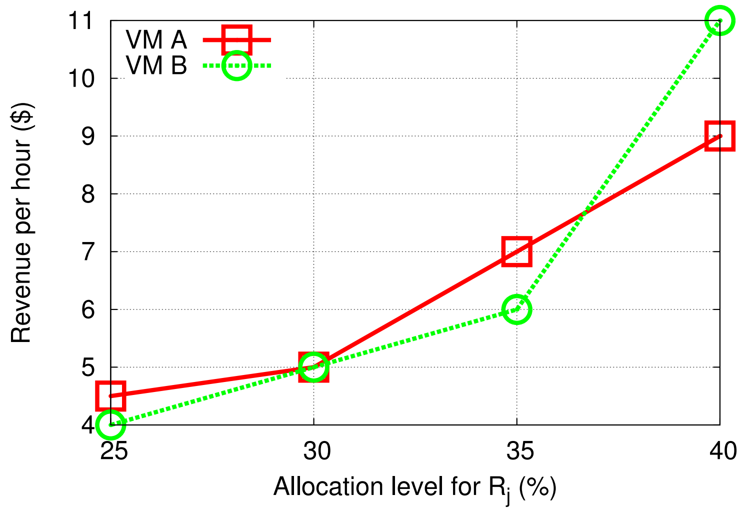


Figure 7.2: Illustration of sub-optimal allocation with unit δ increments.

of δ amount of resource j and the VM that provides the maximum gain in revenue for the addition of the same amount.

Our approach is based on making potentially multiple changes to resource allocation across multiple iterations, with only a small, incremental (δ) resource allocation change within a single iteration. This enables the algorithm to partition resources at a fine granularity, allowing the redistribution of a resource from a single donor VM to multiple recipient VMs and from multiple donor VMs to a single recipient VM. Thus the algorithm is able to consider a large number of resource reallocation configurations. However, one disadvantage of such an approach is that the allocation result achieved at each iteration due to a δ change may not be cumulatively optimal, i.e., for multiple δ change. This is illustrated in Figure 7.2 which depicts the change in revenue for two hypothetical VMs A and B as their allocation for resource R_j changes. Let us assume that the current allocation level of R_j for both VM A and VM B are 30% which generates a revenue of 5\$/hr for both VMs. Let us further assume for simplicity that δ=5 and k=2; these values will typically be different in a real setting with k being greater and δ being either larger or smaller depending on the accuracy of the model w.r.t. modeling the impact of the specific resource. During the next consolidation event, the algorithm would determine in the very first iteration that VM A offers a greater increase in revenue (2\$/hr) for a δ (5%) increment in R_j allocation from 30% to 35% than VM B which offers a lower increase (1\$/hr) for the same increment. In the second iteration, once again VM A offers a greater increment (2\$/hr) for an increment from 35% to 40%, while VM B offers only (1\$/hr) for an increment of 5% from 30% to 35% of R_j. However, if we make the allocation granularity more coarse-grain in the first iteration (say 2δ) then the 10% allocation recipient would have been VM B which offers a greater cumulative increase in revenue (5\$/hr) as opposed to VM A (4\$/hr).

The `CheckReshuffle` optimization module addresses the above shortcoming. It compares the sum of all changes determined as piece-wise optimal in previous iterations with the entire reallocation made as a single unit made at once (i.e., effectively increasing the size of the allocation unit). If `CheckReshuffle` establishes that that the larger granularity allocation of a single resource is more beneficial than incremental δ reallocations, the `StepAdjustments` function accordingly modifies the VMs assigned for maximum gain and minimum loss during the current iteration.

In the final section, (lines 10-18), the Algorithm 1 checks if the *MaxNetProfit* is greater than 0, i.e., there exists an additional revenue benefit from resource redistribution. Upon success, the resource transfers and other manipulations occur from lines 12 to 16. Otherwise, the algorithm is unsuccessful in finding a better resource allocation than the current one and no resource redistribution would take place.

7.3 Evaluation

The goal of this section is to evaluate the effectiveness of our revenue driven resource allocation algorithm which applies our previously-built machine-learning based performance models for virtualized applications [KRG$^+$12]. We compare our method with an intelligent industry technique deployed using current technology of dividing allocations based on relative VM priority e.g. VM *shares*. We demonstrate how starting with some initial VM resource assignments, our algorithm at each iteration, suggests changes to the resource assignments, that ultimately lead to an increase in total revenue for the data center.

7.3.1 Experimental Setup

For experiments, we created a cluster of identical AMD-based Dell PowerEdge 2970 servers with dual sockets and six 2.4 GHz cores per socket. Each server has 32 GB of physical memory and ran the VMware ESXi- 5.1 hypervisor. All the VMs ran Ubuntu-Linux-10.04. The virtual machine disks (VMDKs) were placed in a centralized 1.2 TB LUN located in a separate storage server using RAID-0 with four SAS drives. The VMDKs were mounted on the compute servers using NFS. The experimental test-bed and resource allocation procedure are depicted in Figure 7.3. The VMs were administered using VMWare vCenter Server [vCe]. We used the VMware implemented concept of *resource pool* that extends the per-VM controls

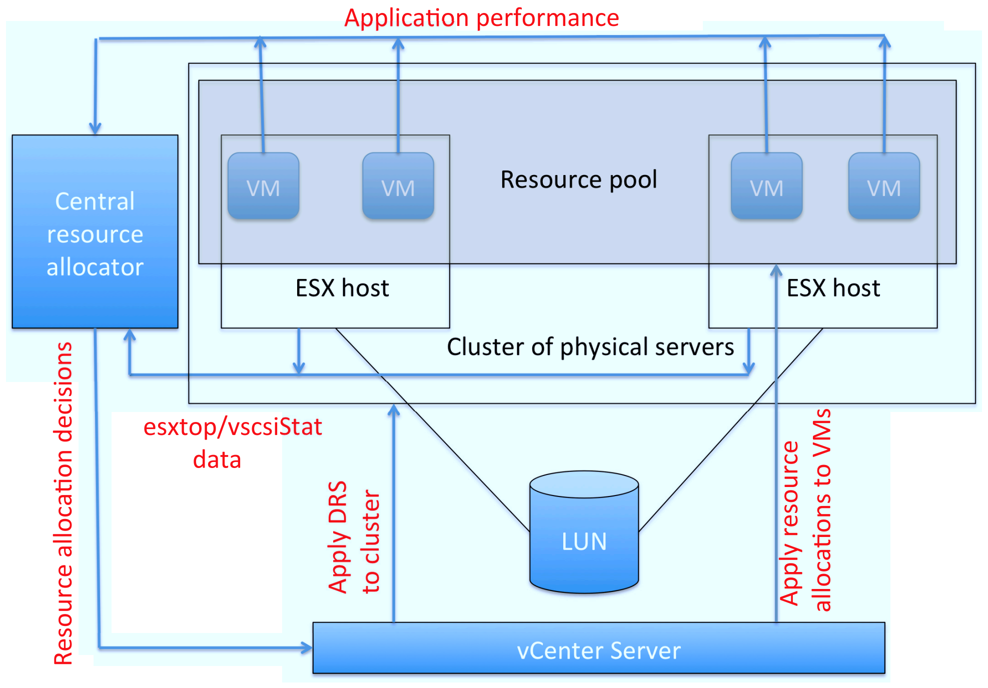


Figure 7.3: Experimental framework for revenue driven resource allocation.

to be applicable to a group of VMs [GHJ+12]. VMs are placed on the resource pool with a condition that the sum of reservations on any resource dimension to the pool of VMs is not to exceed the reservation on the pool. The advantage of using resource pools is that it aggregates physical resources from multiple hosts creates the illusion of a single virtual server. In other words, a resource pool is constructed using a cluster and the cluster in turn comprises of multiple physical machines. This resource virtualization achieves transparent migration of the pooled VMs between the hosts in the cluster at run-time. Migration may occur either as a result of a certain allocation assignment to a particular VM being deemed unsupportable by the current server or due to internal load balancing methods [GHJ+12]. In our setup, we created a pre-configured resource pool and distributed its resources to individual VMs.

A *central resource allocator* executes on a separate Dell PowerEdge T105 machine with a quad-core AMD Opteron processor (1.15GHz×4), 8 GB of physical memory, and a 7.2k RPM disk running Ubuntu-Linux-10.10. VM workloads were run previously in one of the cluster nodes in a staging environment and the application performance models were built and recorded in the central resource allocator. Each ESX host runs the *esxtop* tool to collect per-VM level as well as host-level performance data. We also ran vscsiStats [vsc] on each ESX host which reports storage I/O latency statistics for each VMDK.

7.3.2 Alternate Solutions

We now discuss alternate solutions to dynamic resource management that are possible in the context of a VMware ESX host. An ESX host provides several control knobs for managing resource assignments to individual VMs. In particular *Limit*, *Reservation*, and *Share*; each can be used to control the allocation of CPU and memory to VMs [GHJ+12]. *Limit* places an upper bound on the amount of resource a VM can consume; *reservation* guarantees a certain minimum amount of resource a VM can utilize. *Share* is a prioritization scheme by which the ESX can dynamically vary the resource allocated to a VM between its reservation and limit values based on a specified priority or weight value as demand varies. In other words, shares have the potential to dynamically distribute physical resources proportional to the SLA weights of the VMs based on the actual demand. The work-conserving nature of shares makes it an instant choice of tool for resource distribution based on application-specified SLA priorities for server administrators. On the other hand, limit is non-work conserving but it provides strict isolation in resource multiplexing (for CPU and memory) between VMs on the same host. To compare, we executed our resource management algorithm by using either *shares* as the control knob or applying *limits* as the control knob to enforce resource assignemnts. Although the algorithm was designed to use limit values, while applying shares, we mapped each suggested limit to its share value by dividing the resource assignment with the total capacity of the resource pool. Specifically, we evaluate four schemes: *(i) Share_Reservation:* this case applies shares in combination with some reservations for both CPU and memory; *(ii) Share_noReservation:* shares are used without any reservation for both CPU and memory; *(iii) Limit_Reservation:* limits are used as the controlling knobs for VMs alone with reservations for both CPU and memory; and *(iv) Limit_noReservation:* limits are applied without any specified reservations.

Irrespective of the resource assignment mechanism (i.e. limits or shares), the resource allocation process works as follows. Initially, the available CPU and memory in the resource pool are divided among the running VMs either equally or in proportion to the application-specified SLA weights. VMs were allowed to run for an epoch of 5 mins. VMs continuously report the application performance to the central resource allocator. Once the interval elapses, the VM-level statistics are collected from each ESX host. The SLA functions are applied to transform the application performance to corresponding revenue values in USD. Application performance models are consulted next to determine the relative gain or loss in revenue if resources were to be added or subtracted from individual VMs. The greedy algorithm (listed in Algorithm 1) is then run using the model-predicted revenue data and a new set of resource assignments for the pool of VMs are generated. The new assignments are informed to the vCenter Server which guarantees the successful completion of the new allocations. The VMs are run for another epoch and the whole process repeats. The procedure stops if there is no additional gain in total revenue across multiple iterations (2) or if the models do not predict any further net profit by reassignment of resources to the current pool of VMs. However, it is important to note that reallocation will again become necessary if a new VM is added to the pool, or a running

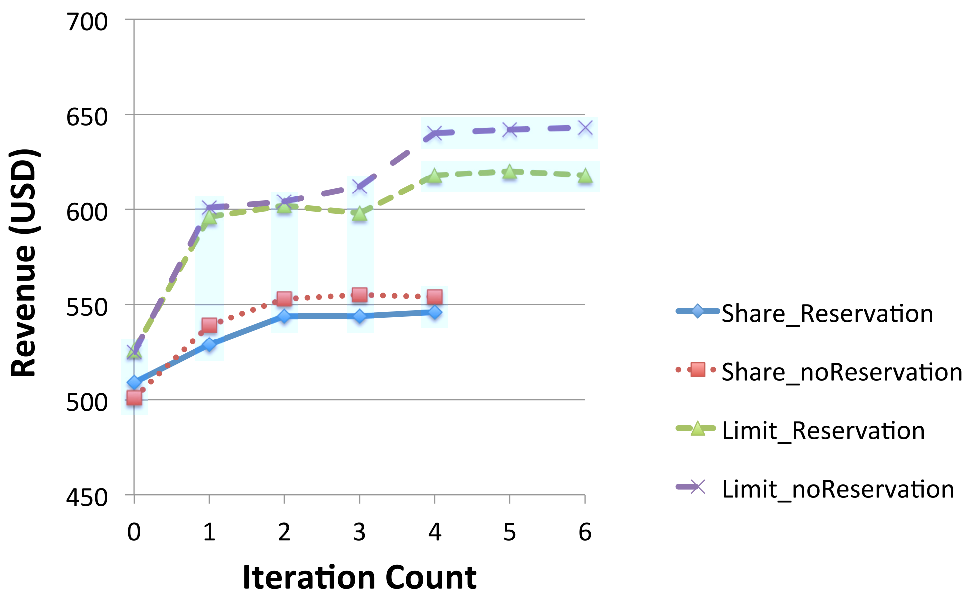


Figure 7.4: Change in revenue when started with equal resource allocations

VM is stopped, or the loads inside the VMs change.

7.3.3 Quantitative Evaluation

To illustrate the power of our framework, we created a pool of 10 VMs each running an identical instance of the Filebench Webserver [fil] workload. The resource pool capacity for CPU was set at 4 GHz and that for memory was 4 GB. Initially, resources were distributed equally among the VMs. We chose δ, the lowest granularity of resource movement, to be 100 MHz for CPU and 64 MB for memory. The value of k was selected as 2. In other words, in each epoch of the resource allocation procedure, a VM was allowed to have a maximum change of 200 MHz of CPU and 128 MB of memory from its previous assignment. Caution was taken to minimize the modeling inaccuracy that may arise due to significantly low δ or high kδ. Since models were trained with data points separated from each other in the parameters space at coarse granularity, choosing a really small value for δ lead to prediction inaccuracy. On the other hand, since the performance models use the current observed VM storage I/O latency to predict the application performance for next iteration, it is important that the I/O latency remains stable. The value of kδ (the maximum resource change) should not be too high as it may destabilize the VM storage I/O latency across multiple iterations within the revenue maximization algorithm.

Figure 7.4 shows how the total revenue from 10 VMs changes as we apply our algorithm after each epoch or iteration. We compare all the four scenarios we have described before. In case of reservation, we set each VM to have atleast 200 MHz CPU and 256 MB of memory. In cases of no reservations, the values were set at zero. While using shares, we set the limit of each VM to the capacity of the resource pool thereby forcing SLA-based prioritization in resource partitioning. On the other hand, in cases using limits, all the VMs were initialized with equal shares. Further, the sum of limits of the pool of VMs were set to be equal to the capacity of the resource pool. SLAs were chosen as simple weight values to transform the normalized application performance metrics to US Dollars. As we see from Figure 7.4, in each iteration, our resource allocation algorithm drove the assignment from a state of lower revenue to a state of higher revenue. However, our proposed limit-based approaches provided much higher revenue than the share scheme. *Limit_noReservation* delivered a 22% increase in revenue with respect to the initial placement. On the other hand, *Share_noReservation* rendered 10% revenue increment. More importantly, the end state achieved when using VM resource limits delivered 18% higher revenue in comparison to that obtained when using shares. This result underscores the usefulness of employing limit-based approach in combination with the greed heuristic to automatically increase the revenue of virtualized data centers. We observed a gain of 7% when we started with a different initial configuration where resources are

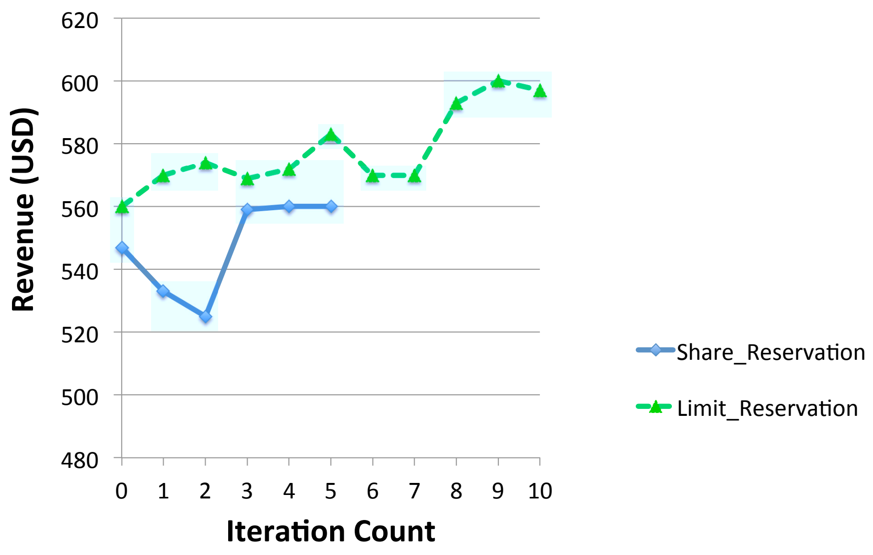
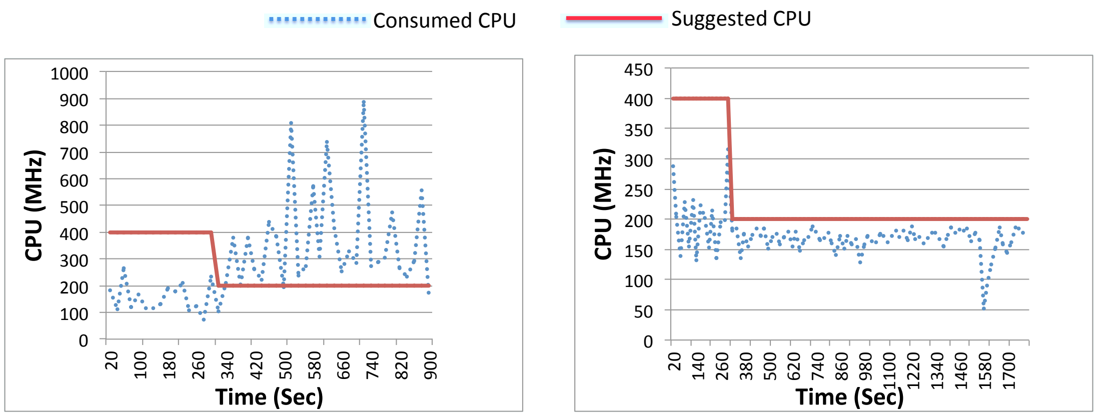


Figure 7.5: Change in revenue when the initial resource allocations to VMs are assigned proportionally to respective SLA weights

assigned proportionally to the application-specified SLA weights (Figure 7.5). Here too, using limits provided higher revenue than using shares.

To understand why using shares led to relatively poor outcome when compared to using limits, we analyzed the VM with the lowest SLA weight and plotted its average CPU consumption against time for both the allocation strategies. We compared the CPU consumption in each graph with the allocation suggested by our algorithm. As we see from Figure 7.6, as expected, the shares allow the allocation of the VM to fluctuate arbitrarily irrespective of its suggested assignment as demand varies. Despite this particular VM achieving higher performance, it did so by reducing the allocations of the higher priority VMs. Since this particular VM had the lowest priority, it contributed little to the collective revenue across the pool of VMs while reducing the chances of extracting a higher revenue from the VMs with higher SLA weights. On the other hand, limit-based allocation enforced an useful isolation among individual VMs which contributes to much higher total from multiple hosted VMs.



(a) Impact of Shares (b) Impact of Limits

Figure 7.6: Trend of actual CPU consumption and suggested CPU allocation for a VM across multiple iterations of the resource allocation algorithm. The chosen VM has the lowest SLA priority. Suggested allocations are applied to the VMs using either *shares* or *limits*.

7.4 Summary

In this chapter, we proposed and evaluated a novel revenue driven resource allocation method to distribute available physical resources to VMs hosted in a data center. We provided a formalization of this problem and proved that this revenue maximization problem is NP-hard. A greedy but fast heuristic solution to the revenue maximization problem was proposed and implemented. We experimentally validated that our machine-learning based models can be applied in cooperation with the revenue maximization algorithm to substantially increase the data center revenue. The results showed up to 22% gain in revenue by using the proposed algorithm, starting with equal assignments of resources. When starting with allocations in proportion to the application-specified SLA weights, our algorithm registered 7% gain in total revenue. Moreover, our scheme increased the total revenue by 18% when compared to a share-based allocation technique. Fine-grained analysis revealed better isolation in sharing of CPU and memory by the VMs while applying limit-based allocation compared to share-based allocation. In Section 9.3, we identify several directions for future research to strengthen the impact and to broaden the scope of this framework.

CHAPTER 8

CONCLUSIONS

Optimal management of data center resources is a crucial yet cumbersome task. While promising to relieve administration complexity, advancement of virtualization technologies has temporarily compounded the resource management problem due to higher degree of consolidation within a single host. In this thesis, we devised machine-learning based application performance models for virtualized applications and employed our models to serve two critical operations: *(i)* estimating the virtual machine capacity based on the target performance while renting computing and memory from Infrastructure-as-a-Service (IaaS) providers, and *(ii)* developing a revenue driven resource distribution algorithm to maximize the Service-Level-Agreement (SLA) based revenue for the data centers.

This thesis made the following advances in the state-of-the-art of server resource management in virtualized data centers. First, we identified the key resource parameters characterizing the application performance. We chose high level control knobs that are easily available in any hypervisor platform to tune the allocation of CPU cycles and memory capacity for each VM. We also demonstrated how varying those control parameters affect application performance in a complex, non-linear fashion and that these trends are significantly diverse across different applications. A difficult challenge of parameter identification was picking the accurate parameter for characterizing the performance of a shared storage system which is typically centrally located in a separate server in data centers. Although we could not completely control the partitioning of storage I/O bandwidth, we found that the observed VM I/O latency for a virtual machine disk (VMDK) correctly represents the current contention level in a storage LUN irrespective of the degree of randomness, read/write ratio, number of outstanding I/Os, and the I/O size distribution. The VM-level I/O latency metric also works regardless of if the storage is locally attached or remotely located.

Second, we investigated several machine-learning and non-machine-learning techniques to accurately predict the application performance for a VM given an assignment of CPU and memory, and the observed storage contention (in the form of VM I/O latency). Of these, we selected two popular evolutionary tools, artificial neural network (ANN) and support vector machine (SVM), to be our building blocks for modeling. We tuned these tools and evaluated them on both the XEN and ESX hypervisors by training and testing the accuracy of the models using multiple workloads. Initial evaluation revealed large modeling errors with certain applications. To reduce the modeling errors, we implemented a novel technique of clustering and subdividing the input parameter space based on the application performance values for a given application and then building separate ANN or SVM models for individual clusters. The new optimizations improved the prediction accuracy and reduced the average and 90th percentile prediction errors substantially.

Third, we proposed and implemented a framework for calculating the required compute power and memory capacity of a VM given a target performance objective and an allowable range of VM I/O latency while renting server space from a cloud host. The performance models were queried to obtain the optimal VM sizes. We experimentally demonstrated that model-based VM sizes not only achieved the desired performance for all of our chosen target data points spread across two different workloads, but also the proposed sizes were optimal in most cases for both the CPU and memory dimensions and close to optimal for the rest. This empirical evidence underscores the power of model-based VM sizing for cloud service providers.

Last, we designed, built, and evaluated a novel revenue driven resource allocation algorithm which partitions the available physical resources among a pool of VMs with the goal of attaining high SLA-based revenue for the data center operators. Performance models were integrated with a version of the hill-climbing algorithm to achieve the objective of maximizing SLA-based revenue. To recreate a miniature data-center like environment, we evaluated our framework by building a cluster of multiple ESX machines hosting a large number of VMs with a centralized storage server hosting the virtual disks. Results indicated substantial gain in revenue in comparison to some static partitioning of available resources. More importantly, our system delivered significantly higher revenue in comparison to the share-based allocation method that is deployable in current production systems.

To conclude, this thesis contributes to accelerate the progress of autonomous and dynamic resource provisioning of virtual machines in a data center. However, several aspects of our work require further research. These are the subject of our future work and elaborated upon in the next chapter.

CHAPTER 9
FUTURE WORK

This thesis has addressed indicator VM parameters selection for controlling and inferring VM performance, building and optimization of VM performance models based on suitable machine-learning techniques, VM sizing based on these performance models, and revenue driven dynamic resource allocation in virtualized data centers. In this chapter, we identify the directions in which the work contained in this thesis can be extended in the future.

9.1 VM Sizing

Chapter 6 corroborated the accuracy and efficacy of VM performance models to deliver desired application performance targets when hosting the application VMs in cloud environment. The evaluations depicted 100% success in achieving target performance levels. More importantly, the configured VM sizes were optimal in 65% of the cases. However, further investigations are required to inquire why optimal allocations are not being realized in some cases. Although, our data indicate that the sub-optimality (for the 35% of the target points) is within a few allocation units for most of the points, in one or two cases our solution is farther away from the optimal. The first extension to this work should identify whether this sub-optimality is a consequence of poor performance prediction or of any procedural shortcoming in our sizing algorithm or both.

In the version of the VM sizing problem that was explored in this dissertation, the storage performance indicator parameter, *VM I/O latency*, was considered as a static input. The sizing tool thus only estimated optimal CPU and memory for the application VM at a given VM I/O latency. No calculation of optimal VM

I/O latency was performed; the VM I/O latency parameter merely served as an input along with the target performance metric. Since current storage systems and hypervisors do not provide fine-grained control of I/O latency, we did not explore this direction in our initial work on the VM sizing problem. In the future, we anticipate that VM I/O latency will be virtualized and allocated similar to CPU and memory resources. This is attractive for both the cloud service provider who can now optimize storage I/O and charge for various levels of I/O performance as well as for the customer who can expect a specific storage performance for their VMs. The second extension should incorporate this estimation of acceptable VM I/O latency level within our sizing framework. If I/O latency gets provisioned in data center, the critical question that arises is trading-off between memory and I/O latency since these two are inter-dependent. We believe that our modeling techniques will be able to characterize the inter-dependence between these control variables because we train the models under various combinations of memory and observed VM I/O latency values. The future work should evaluate this version of the problem by applying I/O latency controlling techniques on a shared storage system to calculate optimal VM I/O latency for the target VM.

9.2 Performance Modeling

Chapter 5 demonstrated how advanced and optimized use of certain machine-learning techniques rendered highly accurate performance prediction of virtualized workloads that in turn delivered optimal VM sizing (chapter 6) and data center resource distribution (chapter 7). However, there are important and challenging new directions yet to be explored.

9.2.1 Online Updating of Performance Models

In this thesis, we have assumed that the application behavior is stable and addressed the offline modeling of its performance. Such a performance model is valuable to a variety of applications that are mainly concerned with average case performance and have static workloads. However, for applications that service dynamically changing workloads, online training of the performance model becomes necessary. Several issues need to be answered to make online modeling practical.

First, it is necessary to identify changes in the resource demand level of client applications. One approach is to incorporate load/demand as an another input parameter to our performance model and constantly monitor the load level of the target VM. However, this approach may require domain knowledge and thus can be potentially intrusive. Another approach is to constantly monitor application performance under allotted allocations and observed contentions. If the newly recorded results under the same allocations and contention differ significantly from the predictions of the original performance model, a change in load level could be the cause. This approach is application agnostic and non-intrusive. The future research should evaluate the pros and cons of both the techniques. Second, it is important to distinguish between short-term and long-term variations. Application performance can substantially change at a certain instant in time due to short-lived spikes in clients load or abrupt and temporary change of machine environments. Short-lived outliers would need to be isolated from sustained performance variations to reduce noise in the training data. Third, an approach to efficiently retrain the model online is required. We believe that our current approach can be extended to work in a dynamic setting primarily because of the relatively low training times incurred as shown in Chapter 5. The future work should develop techniques for effective construction and updating of performance models online.

9.2.2 Cross-platform Performance Modeling

In the current version, the performance models were pre-built in a staging environment with the assumption that the application VMs will run later in an identical deployment infrastructure. Our current experimental cluster consists of identical physical machines. However, data center clusters may consist of dissimilar physical machines with varying underlying hardware and thus potentially different effects of resource allocations on application performance metrics. The performance model built on a particular type of hardware may be erroneous in its predictions if the application migrates to a machine with a non-identical hardware configuration. One approach to deal with this challenge is to build separate performance models for all distinct machine configurations in the cluster. But the training data collection for performance models is a time-consuming task and constructing numerous models will prolong the staging process. The smarter solution should transform an application performance model trained on a specific machine hardware to an updated model when the target VM is migrated to a server with a dissimilar configuration. Such cross-platform modeling techniques will be valuable for future generation of data centers.

9.2.3 Modeling Cache Contention

Our performance models have explicitly included the parameter for I/O contention in shared storage system. However, our work so far has ignored another key shared resource i.e. CPU cache. As demonstrated in recent works [KVR, GLKS11], the shared processor cache introduces significant performance interference among the co-located VMs. In our experiments, we maintained low VM-to-core ratio to minimize any potential cache contention effects on the running workloads. However, in an actual production environment, a single core is often shared across large number of VMs. In such environments, CPU cache interference modeling will be necessary. Such modeling should identify parameters which accurately characterize the interference of co-located VMs in shared CPU cache.

9.3 Dynamic Resource Allocation

Chapter 7 motivated the impact of revenue driven framework for distributing cluster resources among client VMs. The implementation and evaluation of the greedy heuristic in cooperation with the apriori constructed performance models registered up to 18% total revenue gain in comparison to the existing share-based proportional allocations. We envision several extensions to complement and strengthen our system.

First, the allocation of storage was addressed indirectly by characterizing contention in a shared LUN with virtual disk I/O latency as the indicator parameter. Unlike CPU and memory, storage is not easily partitionable; or in other words, allocation of certain levels of I/O latency or I/Os Per Second (IOPS) to VMs is non-trivial and not readily available yet. Nevertheless, the effect of I/O latency on application performance is significant. We have sidestepped this issue by applying the VM I/O latency collected in previous allocation decision to suggest future assignments and guide the pool of VMs to progress slowly towards a more optimal resource allocation state. Recent techniques to partition and/or prioritize the allocation of disk I/O bandwidth [HPcC04, KKZ05, WAEMTG07, GSZV12, SFS12] suggest that future work on cluster resource management would likely borrow and extend some of these ideas for storage provisioning. Given such storage capabilities, storage provisioning will involve tunable *reservation/limit* on *IOPS* or *I/O latency* in conjunction with the observed metrics.

Second, we had intuitively come up with SLA functions of the VMs based on our knowledge of the workloads and their resource consumption characteristics. Although this naive approach was sufficient for our evaluation purposes, future research should address the profitability of the SLA functions in the production environment. A SLA function maps application performance to revenue in US dollars. The application performance in turn is a function of resource assignments to that target VM. The key question here is to determine whether SLA functions are profitable for the data center and provide good value to clients at the same time. A good SLA function would need to be designed by considering the resource requirements and adjusting the revenue curve accordingly. The VM sizing framework (Chapter 6) can be utilized to provide resource requirement hints to the data center administrators.

Third, the typical web and online analytical processing workloads running in data centers today are multi-tier. Usually, the topology of multi-tier workloads consists of several inter-dependent tiers of the same application, each tier being encapsulated in separate VMs. Resource allocation to such applications will entail reconfiguring multiple VMs at the same time. In our work, this problem did not manifest as all the tiers were run on the same VM. An intuitive approach to deal with multi-tier applications is to logically group all the correlated VMs into a single one and apply resource allocation to that group as a whole. However, caution is required in distributing the resources allocated to that logical group across individual VM tiers. Another approach will be to treat each VM tier separately in terms of resource allocation decisions. Since the tiers are not required to be co-located, this method helps easier migration of VM tiers across hosts for load-balancing in the cluster. Future work should explore alternate strategies to manage multi-tier applications.

Fourth, the cost of a specific resource type may serve as an important input to our proposed revenue model. For example, an application read request can be served both from memory or from storage without violating the SLA requirement. If we are achieving the same revenue level by serving either from main memory or from a storage device, it may be worthwhile to allocate a resource type which is less expensive. Additionally, if a certain resource type is over-utilized, higher priority may be given to the under-utilized resource for obtaining similar revenue levels. In this thesis, we have not differentiated between resource types in terms of their costs or current demands. If the same higher level of revenue can be achieved with either CPU or memory, we have not enforced any specific order to pick up a resource type between them. Cost of hardware was also not accounted for. Future studies should address cost models in our revenue framework and seek to attain higher revenue by allocating less expensive or more available resource type first before allocating more expensive or less available ones.

Fifth, the greedy heuristic proposed in this dissertation provided an approximate solution to the optimization problem which was proved to be NP-hard in Section 7.1. Although the heuristic delivered significant increase in revenue, a thorough analysis of the optimality of the solution in achieving the revenue maximization objective is necessary. Moreover, our implemented greedy heuristic is incremental and thereby can be driven towards local maxima points, a common problem with hill-climbing algorithms. Future extensions should propose and evaluate strategies to overcome local maxima issues.

Finally, the load inside the client workloads running in cloud environment(s) to-day are constantly changing. Any resource allocation tool should possess high degree of elasticity to dynamically shrink or expand VM size as the load drops or escalates. Future research should incorporate this demand induced resource reconfiguration in the cluster to extract the full potential from our revenue guided framework. This extension is tied to the successful online updating of the performance models as documented in the previous section. A comprehensive demand based, revenue driven, dynamic resource allocation approach and accompanying suite of tools will be very valuable to the optimized management of cloud data center resources.

BIBLIOGRAPHY

[BDF+03] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, and R. Neugebauer. Xen and the art of virtualization. In *Proc. of ACM SOSP*, 2003.

[BGF+10] Peter Bodik, Moises Goldszmidt, Armando Fox, Dawn B. Woodard, and Hans Andersen. Fingerprinting the datacenter: Automated classification of performance crises. In *EuroSys '10 Proceedings of the 5th European conference on Computer systems*, pages 111–124, 2010.

[BM05] Mohamed N. Bennani and Daniel A. Menascé. Resource allocation for autonomic data centers using analytic performance models. In *ICAC*, pages 229–240. IEEE Computer Society, 2005.

[BS02] Kurt M. Bretthauer and Bala Shetty. The nonlinear knapsack problem – algorithms and applications. *European Journal of Operational Research*, (138):459–472, 2002.

[BTI+11] Roy Bryant, Alexey Tumanov, Olga Irzak, Adin Scannell, Kaustubh Joshi, Matti Hiltunen, H. Andres Lagar-Cavilla, and Eyal De Lara. Kaleidoscope : Cloud micro-elasticity via vm state coloring. In *Proceedings of the sixth conference on Computer systems (EuroSys)*, pages 273–286, 2011.

[capa] Vmware capacity planner. http://www.vmware.com/products/capacity-planner/.

[capb] Vmware vcenter capacityiq. http://www.vmware.com/products/vcenter-capacityiq/.

[CAVL09] Austin T. Clements, Irfan Ahmad, Murali Vilayannur, and Jinyuan Li. Decentralized Deduplication in SAN Cluster File Systems. In *Proc. of USENIX ATC*, June 2009.

[CGK+04] Ira Cohen, Moises Goldszmidt, Terence Kelly, Julie Symons, and Jeffrey S. Chase. Correlating instrumentation data to system states: A building block for automated diagnoses and control. In *Proc. of the 6th USENIX OSDI)*, 2004.

[CZG^{+}05] Ira Cohen, Steve Zhang, Moises Goldszmidt, Julie Symons, Terence Kelly, and Armando Fox. Capturing, indexing, clustering, and retrieving system history. In *Proc. of ACM SOSP*, 2005.

[DCA^{+}03] Ronald P. Doyle, Jeffrey S. Chase, Omer M. Asad, Wei Jin, and Amin Vahdat. Model-based resource provisioning in a web service utility. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[DO00] Peter A. Dinda and David R. O'Hallaron. Host load prediction using linear models. *Cluster Computing*, 3(4), 2000.

[Dre08] Ulrich Drepper. The Cost of Virtualization. *ACM Queue*, Feb. 2008.

[ec2] Amazon elastic compute cloud (amazon EC2). http://aws.amazon.com/ec2/.

[fil] *Filebench: a framework for simulating applications on file systems.* http://www.solarisinternals.com/wiki/index.php/FileBench.

[fio] *fio: Flexible I/O tester.* http://freshmeat.net/projects/fio/.

[GAW09] Ajay Gulati, Irfan Ahmad, and Carl Waldspurger. PARDA: Proportionate Allocation of Resources for Distributed Storage Access. In *Proc. of USENIX FAST*, Feb. 2009.

[GCF^{+}10] Archana Ganapathi, Yanpei Chen, Armando Fox, Randy Katz, and David Patterson. Statistics-driven workload modeling for the cloud. In *SMDB*, 2010.

[GHJ^{+}12] Ajay Gulati, Anne Holler, Minwen Ji, Ganesha Shanmuganathan, Carl Waldspurger, and Xiaoyun Zhu. *VMware Distributed Resource Management: Design, Implementation and Lessons Learned.* http://labs.vmware.com/publications/gulati-vmtj-spring2012, 2012.

[GLKS11] Sriram Govindan, Jie Liu, Aman Kansal, and Anand Sivasubramaniam. Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC)*, 2011.

[GMV10] Ajay Gulati, Arif Merchant, and P. Varman. mClock: Handling Throughput Variability for Hypervisor IO Scheduling. In *9th USENIX OSDI*, October 2010.

[GRCK07] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, and Alfons Kemper. Capacity management and demand prediction for next generation data centers. In *ICWS*, July 2007.

[GSA$^+$11] Ajay Gulati, Ganesha Shanmuganathan, Irfan Ahmad, Carl Waldspurger, and Mustafa Uysal. Pesto: Online storage performance management in virtualized datacenters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC)*, 2011.

[GSZV12] Ajay Gulati, Ganesha Shanmuganathan, Xuechen Zhang, and Peter Varman. Demand based hierarchical qos using storage resource pools. In *Proceedings of USENIX Annual Technical Conference*, 2012.

[Ham] P. L. Hammer. *Studies in integer programming.* IBM Deutschland.

[HPcC04] Lan Huang, Gang Peng, and Tzi cker Chiueh. Multi-dimensional storage virtualization. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 2004.

[Kat97] J. Katcher. Postmark: A new file system benehmark. Technical report, Network Appliance, 1997.

[KKZ05] Magnus Karlsson, Christos Karamanolis, and Xiaoyun Zhu. Triage: Performance differentiation for storage systems using adaptive control. In *ACM Transactions on Storage (TOS)*, pages 457–480, Nov'2005.

[Kot11] Evangelos Kotsovinos. Virtualization: Blessing or Curse? *ACM Queue*, Jan. 2011.

[KRDZ10] Sajib Kundu, Raju Rangaswami, Kaushik Dutta, and Ming Zhao. Application Performance Modeling in a Virtualized Environment. In *Proc. of IEEE High Performance Computer Architecture (HPCA)*, January 2010.

[KRG+12] Sajib Kundu, Raju Rangaswami, Ajay Gulati, Ming Zhao, and Kaushik Dutta. Modeling Virtualized Applications using Machine Learning Techniques. In *Proceedings of the 8th ACM conference on Virtual Execution Environments (VEE)*, March 2012.

[KVR] Ricardo Koller, Akshat Verma, and Raju Rangaswami. Estimating application cache requirement for provisioning caches in virtualized systems. In *Proceedings of the 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS).*

[LZSA05] X. Liu, Xiaoyun Zhu, Sharad Singhal, and Martin F. Arlitt. Adaptive entitlement control of resource containers on shared servers. In *IM*, pages 163–176. IEEE, 2005.

[MAC+11] John C. McCullough, Yuvraj Agarwal, Jaideep Chandrashekar, Sathyanarayan Kuppuswamy, Alex C. Snoeren, and Rajesh K. Gupta. Evaluating the effectiveness of model-based power characterization. In *Proc. of USENIX Annual Technical Conference*, 2011.

[MIK+10] Xiaoqiao Meng, Canturk Isci, Jeffrey Kephart, Li Zhang, Eric Bouillet, and Dimitrios Pendarakis. Efficient resource provisioning in compute clouds via vm multiplexing. In *ICAC*, 2010.

[NKG10] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *EuroSys '10*, pages 237–250, 2010.

[pam] *fpc: Flexible procedures for clustering.* http://cran.r-project.org/web/packages/fpc/index.html.

[PHZ+09] Pradeep Padala, Kai-Yuan Hou, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kang G. Shin. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European conference on Computer systems/EuroSys*, pages 13–16, 2009.

[pla] Novell platespin recon. http://www.novell.com/products/recon/.

[PSZ+07] Pradeep Padala, Kang G. Shin, Xiaoyon Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive control of virtualized resources in utility computing environments. In *Proc. of Eurosys*, pages 289–302, 2007.

[R] *The R Project for Statistical Computing.* http://www.r-project.org/.

[rac] The rackspace cloud. http://www.rackspace.com/cloud/.

[RBX+09] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Le Yi Wang, and Gang George Yin. VCONF: a reinforcement learning approach to virtual machines auto-configuration. In *ICAC*, pages 137–146. ACM, 2009.

[rub] *RUBiS: Rice University Bidding System.* http://rubis.ow2.org/.

[Sar94] Warren S. Sarle. Neural networks and statistical models, 1994.

[SFS12] David Shue, Michael J. Freedman, and Anees Shaikh. Performance isolation and fairness for multi-tenant cloud storage. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation (OSDI)*, 2012.

[SKZS08] Christopher Stewart, Terence Kelly, Alex Zhang, and Kai Shen. A dollar from 15 cents: Cross-platform management for internet services. In *Proceedings of the USENIX Annual Techinal Conference*, pages 199–212, 2008.

[SLG+09] G. Soundararajan, D. Lupei, S. Ghanbari, A. D. Popescu, J. Chen, and C. Amza. Dynamic resource allocation for database servers running on virtual storage. In *Proceedings of FAST*, 2009.

[SMA+08] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 953–966, 2008.

[SS05] Christopher Stewart and Kai Shen. Performance modeling and system management for multi-component online services. *Proc. of the 2nd USENIX NSDI*, 2005.

[SSGW11] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: Elastic resource scaling for multi-tenant cloud systems.

In *Proceedings of ACM Symposium on Cloud Computing (SOCC)*, 2011.

[sys] *Sysbench: a system performance benchmark.* http://sysbench.sourceforge.net/.

[UYA+05] Sandeep Uttamchandani, Li Yin, Guillermo A. Alvarez, John Palmer, and Gul Agha. Chameleon: a self-evolving, fully-adaptive resource arbitrator for storage systems. In *Proc. of USENIX Annual Technical Conference*, 2005.

[vCe] Vmware vcenter server. http://www.vmware.com/products/vcenter-server/.

[VMw10a] VMware, Inc. *Introduction to VMware Infrastructure.* 2010. http://www.vmware.com/support/pubs/.

[VMw10b] VMware, Inc. *vSphere Resource Management Guide: ESX 4.1, ESXi 4.1, vCenter Server 4.1.* 2010.

[VNM+12] Nedeljko Vasic, Dejan Novakovic, Svetozar Miucin, Dejan Kostic, and Ricardo Bianchini. Dejavu: accelerating resource allocation in virtualized environments. In *ASPLOS XVII Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, 2012.

[Vog08] Werner Vogels. Beyond Server Consolidation. *ACM Queue*, Feb. 2008.

[vsc] *Using vscsiStats for Storage Performance Analysis.* http://communities.vmware.com/docs/DOC-10095.

[WAEMTG07] Matthew Wachs, Michael Abd-El-Malek, Eno Thereska, and Gregory R. Ganger. Argon: performance insulation for shared storage servers. In *Proceedings of the 5th USENIX conference on File and Storage Technologies (FAST)*, 2007.

[WCOS08] Timothy Wood, Ludmila Cherkasova, Kivanc Ozonat, and Prashant Shenoy. Profiling and modeling resource usage of virtualized applications. In *Proc. of ACM/IFIP/USENIX Middleware*, 2008.

[WSW08] Jonathan Wildstrom, Peter Stone, and Emmett Witchel. CARVE: A cognitive agent for resource value estimation. In *ICAC*, pages 182–191. IEEE Computer Society, 2008.

[WVA$^+$12] Andrew Wang, Shivaram Venkataraman, Sara Alspaugh, Randy Katz, and Ion Stoica. Cake: Enabling high-level slos on shared storage systems. In *Proceedings of the 3rd ACM Symposium on Cloud Computing (SOCC)*, 2012.

[WZS05] Z. Wang, X Zhu, and S. Singhal. Utilization and slo-based control for dynamic sizing of resource partitions. In *Proc. of 16th IFIP/IEEE Distributed Systems: Operations and Management (DSOM)*, October 2005.

[XCZ$^+$11] Pengcheng Xiong, Yun Chi, Shenghuo Zhu, Junichi Tatemura, Calton Pu, and Hakan Hacigumus. Activesla: A prot-oriented admission control framework for database-as-a-service providers. In *Proceedings of ACM Symposium on Cloud Computing (SOCC)*, 2011.

[XZF$^+$08] Jing Xu, Ming Zhao, José A. B. Fortes, Robert Carpenter, and Mazin S. Yousif. Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. *Cluster Computing*, 11(3):213–227, 2008.

[ZBJ$^+$09] Wei Zheng, Ricardo Bianchini, G. John Janakiraman, Jose Renato Santos, and Yoshio Turner. JustRunIt: Experiment-Based Management of Virtualized Data Centers. In *Proceeding of the USENIX Annual Technical Conference*, 2009.

VITA

SAJIB KUNDU

May 23, 1984	Born, Kolkata, India
2006	Bachelor or Computer Science and Engineering Jadavpur University Kolkata, India

PUBLICATIONS AND PRESENTATIONS

Sajib Kundu, Raju Rangaswami, Ajay Gulati, Ming Zhao, and Kaushik Dutta (2013). *Revenue Driven Resource Allocations for Virtualized Data Center.* To be submitted.

Sajib Kundu, Raju Rangaswami, Ajay Gulati, Ming Zhao, and Kaushik Dutta (2012). *Modeling Virtualized Applications using Machine Learning Techniques.* 8th Annual International Conference on Virtual Execution Environments (VEE).

Sajib Kundu, Raju Rangaswami, Kaushik Dutta, and Ming Zhao (2010). *Application Performance Modeling in a Virtualized Environment.* 16th IEEE International Symposium on High Performance Computer Architecture (HPCA).

Kaushik Dutta, Raju Rangaswami, and Sajib Kundu (2008). *Workload-based Generation of Administrator Hints for Optimizing Database Storage Utilization.* ACM Transactions on Storage, vol. 3, no. 4, February.