DYNAMIC MONITORING, MODELING AND MANAGEMENT OF PERFORMANCE AND RESOURCES FOR APPLICATIONS IN THE CLOUD

A Thesis
Presented to
The Academic Faculty

by

Pengcheng Xiong

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology
December 2012

DYNAMIC MONITORING, MODELING AND MANAGEMENT OF PERFORMANCE AND RESOURCES FOR APPLICATIONS IN THE CLOUD

Approved by:

Professor Calton Pu, Committee Chair
Department of Computer Science
Georgia Institute of Technology

Professor Calton Pu, Advisor
Department of Computer Science
Georgia Institute of Technology

Professor Ling Liu
Department of Computer Science
Georgia Institute of Technology

Professor Ed Omiecinski
Department of Computer Science
Georgia Institute of Technology

Professor Leo Mark
Department of Computer Science
Georgia Institute of Technology

Professor Jia Zhang
Silicon Valley Campus
Carnegie Mellon University

Date Approved: August 2012

To my beloved wife, father and mother

ACKNOWLEDGEMENTS

I am indebted to all the people who helped and supported me during my PhD life in Georgia Tech. Especially,

I wish to thank my advisor, Dr. Calton Pu, for his guidance and support for both my research and life. His relentless passion, great enthusiasm and unending energy to do great research is always a source of constant inspiration to me. Most importantly, his broad and long-term view for both research and life will benefit me for a life-time.

I wish to thank my wife, Qianqian, who has given me continuous love, encouragement, and support over the years, especially the most difficult times during my PhD study. I wish to thank my parents, who raised me and gave me the best education that they could think of, and kept watching my progress with their love and considerations.

I appreciate many professors that have directly or indirectly enlightened me, including Dr. Ling Liu, Dr. Ed Omiecinski, Dr. Leo Mark, Dr. Sham Navathe at Georgia Tech; Dr. Jia Zhang at Carnegie Mellon University-Silicon Valley; and my previous advisor Dr. Mengchu Zhou at New Jersey Institute of Technology.

I also would like to thank my collaborators, Dr. Xiaoyun Zhu and Dr. Rean Griffith at VMware, Dr. Zhikui Wang at HP Labs. They helped me a great deal in learning control theory, and in particular how to apply it to computer system management. I would like to thank Dr. Yun Chi and Dr. Hakan Hacıgümüş at NEC Labs for offering me internship opportunities.

Finally, I thank my fellow PhD students in the Distributed Data Intensive Systems Lab, especially Dr. Jinpeng Wei, Dr. Gueyoung Jung, Dr. Qinyi Wu, Dr. Younggyun Koh, Dr. Simon Malkowski, Dr. Danesh Irani, Qingyang Wang, Deepal Jayasinghe,

Junhee Park, Aibek Musaev, De Wang, Yi Yang and Jack Li. Life would have been a lot less fun without them around. Their help and support has contributed to making my PhD fun.

This research has been partially funded by National Science Foundation by IUCRC/FRP (1127904) , CISE/CNS (1138666), RAPID (1138666), CISE/CRI (0855180), NetSE (0905493) programs,and gifts, grants, or contracts from DARPA/I2O, Singapore Government, Fujitsu Labs, Wipro Applied Research, and Georgia Tech Foundation through the John P. Imlay, Jr. Chair endowment. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the thesis committee members or the National Science Foundation or other funding agencies and companies mentioned above.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Emerging trends in Cloud computing bring numerous benefits, such as higher performance, fast and flexible provisioning of applications and capacities, lower infrastructure costs, and almost unlimited scalability. However, the increasing complexity of automated performance and resource management for applications in Cloud computing presents novel challenges that demand enhancement to classical control-based approaches.

An important challenge that Cloud service providers often face is a resource sharing dilemma under workload variation. Cloud service providers pursue higher resource utilization, because the higher the utilization, the lower the hardware cost, operating cost and maintenance cost. On the other hand, resource utilizations cannot be too high or the service provider's revenue could be jeopardized due to the inability to meet application-level service-level objectives (SLOs).

A crucial research question is how to generate as much revenue as possible by satisfying service-level agreements while reducing costs as much as possible in order to maximize the profit for Cloud service providers. To this end, the classical control-based approaches show great potential to address the resource sharing dilemma, which could be classified into three major categories, i.e., admission control, queueing and scheduling, and resource allocation. However, it is a challenging task to apply classical control-based approaches directly to computer systems, where first-principle models are generally not available. It becomes even more difficult due to the dynamics seen in real computer systems including workload variations, multi-tier dependencies, and resource bottleneck shifts.

Fundamentally, the main contributions of this thesis are the efforts to enhance classical control-based approaches by leveraging other techniques to address the increasing complexity of automated performance and resource management in the Cloud through dynamic monitoring, modeling and management of performance and resources. More specifically, (1) an admission control approach is enhanced by leveraging decision theory to achieve the most profitable service-level compliance; (2) a critical resource identification approach is enhanced by leveraging statistical machine learning to automatically and adaptively identify critical resources; and (3) a resource allocation approach is enhanced by leveraging hierarchical resource management to achieve the highest resource utilization.

Concretely, the enhanced control-based approaches are implemented in a collection of real control systems: ActiveSLA, vPerfGuard and ERController. The control systems are applied to different real applications, such as OLTP and OLAP database applications and distributed multi-tier web applications, with different workload intensities, type and mix, in different Cloud environments. All the experimental results show that the prototype control systems outperform existing classical control-based approaches.

Finally, this thesis opens new avenues to address the increasing complexity of automated performance and resource management through enhancement of classical control-based approaches in Cloud environments. Future work will consistently follow the direction of new avenues to address the new challenges that arise with the advent of new hardware technology, new software frameworks and new computing paradigms.

CHAPTER I

INTRODUCTION

1.1 Cloud computing

Cloud computing is the delivery of computing as a service whereby shared resources, software and information are provided as a utility (like the electricity grid) over a network (typically the Internet) [3, 93, 116].

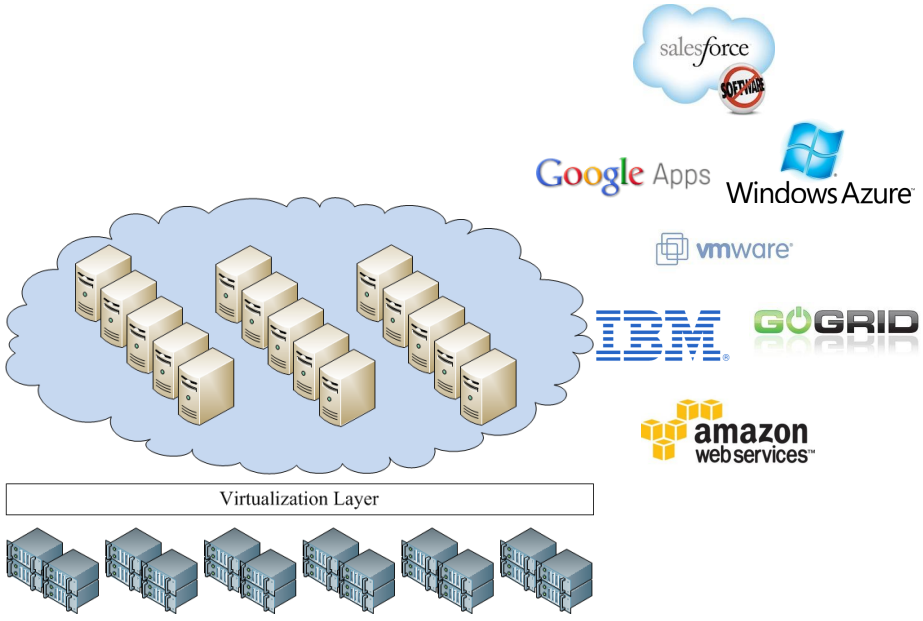


Figure 1: Cloud computing

Most of Cloud providers offer their services according to three fundamental models [3, 93], i.e., infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) as shown in Figure 1. For example, a platform as a service (PaaS) provider such as Windows Azure offers virtualized computing resources to many small to medium businesses (SMBs). These business customers may run a mix of web-based applications in that Cloud environment, e.g., online news forums similar to Slashdot or Digg, auction and e-commerce sites similar to eBay and Target.com,

social media sites similar to Facebook, and business intelligence applications backed by Oracle or Microsoft SQL Server.

Cloud computing offers numerous benefits, such as higher performance, fast and flexible provisioning of applications, lower infrastructure costs, and the ability to combine above-average energy efficiency with the ability to scale-out future applications to an almost unlimited extent. For example, Audi, a German automobile manufacturer, was facing challenges to scale its IT systems because of the increased use of business-critical applications in areas such as production and logistics, supplier relationship management and human resources that challenged their IT infrastructure regarding reliability and flexibility [13]. Audi selected IBM to build a Cloud environment for Audi's SAP infrastructure to rebuild their existing SAP infrastructure [13], including consolidation and virtualization of the server hardware, process standardization, opportunities for performance-related billing and a much higher operational flexibility.

1.2 Research challenges

Although Cloud computing brings advantages over traditional computing paradigms, Cloud service providers often face a resource sharing dilemma. Figure 2 uses economic theory to depict the resource sharing dilemma.

The clients submit jobs to the service providers, e.g., a PaaS provider. The clients express the value of their jobs in a utility function or a service-level agreement [82, 128] that is a contract between the service providers and the clients. Such agreements consist of one or more service-level objectives (SLOs) where the accomplishment of objectives will bring gain (generate revenue inflows) and the breach of objectives will result in penalties. Example objectives include targets concerned with service latency, throughput, availability, security, etc. The service provider makes revenue by accepting the jobs according to the utility function. The service provider may rent

resources from other providers, such as an IaaS provider, and pays for those costs. The gap between the revenue and the cost to run the job is simply the job's profit to the service provider. The ultimate goal of the service provider is to maximize its profit (its net return) while maintaining adequate quality of service, user satisfaction, and other intangibles. In the commercial world, it seems self-evident and undoubted that profit is a valid metric. In academic and scientific circles, we also believe that it is a useful metric because it offers a clear, numerical measure for the amount of net value added by a service.

$$\textbf{Maximize} \quad \text{Profit} = \text{Revenue} - \text{Cost}$$

The resource sharing dilemma appears under workload variation. Cloud service providers pursue higher resource utilization [24], because the higher the utilization, the lower the hardware cost, operating cost and maintenance cost. On the other hand, resource utilizations cannot be too high or the service provider's revenue could be jeopardized due to the inability to meet application-level service-level objectives (SLOs).

1.3 Classical control-based approaches and technical challenges

The classical control-based approaches to automated performance and resource management for applications in Cloud computing such as the resource sharing dilemma could be classified into three major categories as shown in Figure 2. (1) Admission control [46, 112]. When a job arrives, the service provider first executes an admission control algorithm, which decides whether it should accept the job. If it decides to reject the job, then no further action is taken. (2) Queueing and scheduling [52, 63]. If the admission control algorithm accepts the job, it is placed into a work queue,

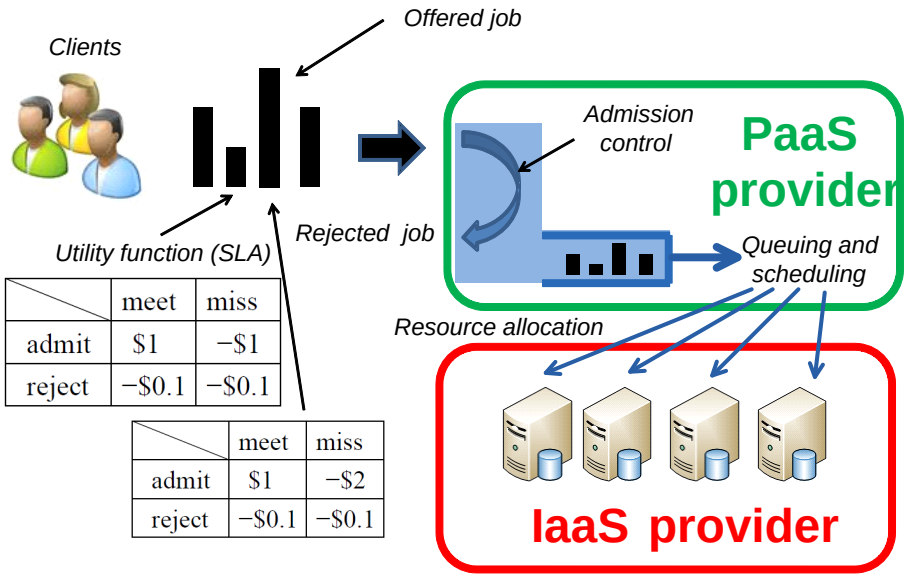


Figure 2: Resource sharing dilemma

from which it is selected at some future time by a job scheduler. The job scheduler determines the order those jobs will be executed in. The job scheduler maintains a preferred schedule, based on its estimates of resource availability, and attempts to execute that. The scheduler is invoked whenever a job arrives, a job completes, or the number of available resources changes; it may choose to run a job, or decide that it cannot do so yet. (3) Resource allocation [66, 105, 114, 124]. In a Cloud environment, resources to run the jobs can be obtained from a separate resource provider such as an IaaS provider, who offers different types of resources with different prices at different times in the future.

The technical challenges of the classical control-based approaches can be summarized into three aspects due to the increasing complexity of automated performance and resource management for applications in Cloud computing.

1.3.1 Admission control

A classical admission control-based approach limits the total number of requests, thereby avoids overload situations where requests violate service-level objectives. Some classical admission control-based approaches make admission decisions merely

based on the number of requests [46] while some other classical admission control-based approaches make admission decisions merely based on the number of each type of request [112]. However, classical admission control-based approaches do not consider that incoming requests have different gains and penalties or put different loads on resource pool, thus making it difficult for service providers to achieve the most profitable service-level compliance. For example, service providers are more likely to admit a request with higher gain than the one with lower gain if both of them have the same probability of meeting the deadline. For another example, service providers are more reluctant to admit a request when the resource pool is heavily-loaded.

The diverse gain and penalty values, uncertain resource pool load and other issues relevant in an admission decision present novel technical challenges that demand enhancement to a classical admission control-based approach in order to achieve the most profitable service-level compliance.

1.3.2 Critical resource identification

A critical resource that affects an application's performance needs to be identified before resource allocation. A classical critical resource identification approach depends on a performance engineer's experience and knowledge. However, the application's performance and resource utilization are stochastically related in a Cloud environment. For example, the critical resource that affects the application's performance can change from CPU to disk I/O from time to time. Although a performance engineer with profound expert experience and deep domain knowledge could determine that one or more critical resources are correlated to the application's performance, this human-based method essentially introduces a bottleneck due to poor scalability and adaptability in a highly time-varying Cloud environment where resolutions are required on the order of minutes rather than hours.

High scalability and adaptability requirements in critical resource identification

in the Cloud present novel technical challenges that demand enhancement to a classical critical resource identification approach in order to effectively identify critical resources for a complex distributed application.

1.3.3 Resource allocation

A classical resource allocation controller which requires/releases critical resources when workload increases/decreases can be used to maintain service-level compliance for a single-component application with the lowest cost. However, a classical resource allocation controller which allocates the same resource amount or keeps the same resource utilization for all the components within the application is not appropriate to apply to a multi-component application where different components within an application have heterogenous requirements for both resource types and amounts. For example, in a typical multi-tier web application, the application tier usually has much higher CPU demand than the other tiers, while the database tier can require much more memory and disk I/O bandwidth. Thus, a classical resource allocation controller will only increase the resource cost for the whole application by allocating erroneous type or inappropriate amount of resource.

Heterogenous resource type and amount requirements for components in a multi-component application in the Cloud present novel technical challenges that demand enhancement to a classical resource allocation controller in order to achieve the highest resource utilization.

1.4 Thesis statement

The increasing complexity of automated performance and resource management for applications in the Cloud demands enhancement to classical control-based approaches. Admission control, critical resource identification and resource allocation can be enhanced by leveraging decision theory, statistical machine learning and hierarchical resource management, outperforming the classical control-based approaches.

The thesis statement is well-supported in this thesis. Concretely, a classical control-based approach is enhanced by leveraging decision theory, statistical machine learning and hierarchical resource management in three solid systems, namely, ActiveSLA, vPerfGuard and ERController. The experimental results show that all the systems outperform existing classical methods.

ActiveSLA demonstrates that a classical admission control approach enhanced by decision theory achieves the most profitable service-level compliance. Different from a classical admission control approach, ActiveSLA also considers the diverse gain and penalty values, uncertain resource pool load and other issues relevant in an admission control.

Due to the enhancement, experimental results show that ActiveSLA is able to make admission control decisions that can obtain at least 20% more profit than several classical admission control approaches.

vPerfGuard demonstrates that a classical critical resource identification approach enhanced by statistical machine learning automatically and adaptively determines the critical system resource metrics that are most strongly correlated with application's performance. Different from a classical critical resource identification approach, vPerfGuard also considers the high scalability and adaptability requirements in critical resource identification for applications in Cloud computing. Due to the enhancement,

experimental results show that vPerfGuard automatically and adaptively selects the correct critical system resources that affect the application's performance.

ERController demonstrates that a classical resource allocation controller enhanced by hierarchical resource management ensures the highest resource utilization while guaranteeing service-level compliance. Different from a classical resource allocation controller, ERController makes high-level resource allocation for the application to guarantee service-level compliance as well as low-level resource allocation partitions for the components of the application to achieve the highest resource utilization. Due to the enhancement, experimental results show that ERController is not only robust to different dynamic workload types, but also can achieve 20% higher resource utilization with the same service-level compliance compared to classical resource allocation approaches.

1.5 Technical contributions

The technical contributions of this thesis are summarized following the development and application of three concrete systems, namely, ActiveSLA, vPerfGuard and ERController. These contributions exhibit how decision theory, statistical machine learning and hierarchical resource management are leveraged into the current control-based approaches to outperform existing classical methods.

ActiveSLA integrates decision theory into a classical admission control approach to achieve the most profitable service-level compliance by two component modules. First, a prediction module provides the probability of a new query finishing its execution before its deadline. The prediction is made by a non-linear classification model using query features, database management system features and operating system features. Second, a decision module determines whether or not to admit the query into a database system. The decision is made by considering (1) the probability risk of the query to meet/miss service-level agreements and (2) the gain/penalty for the

query to meet/miss service-level agreements between a service provider and its clients. Extensive real system experiments are run with standard database benchmarks, under different traffic patterns and service-level agreements. The results demonstrate that ActiveSLA is able to make admission control decisions that are not only more accurate but also more effective profit (at least 20% better) than several classical admission control methods.

vPerfGuard integrates statistical machine learning into a classical control-based approach to automatically and adaptively determine the critical system resource that affects the application's performance through three modules - a sensor module, a model building module, and a model updating module. Once the application is running, vPerfGuard's sensor module collects two categories of system metrics - *VM metrics* from the operating systems within individual VMs and *host metrics* from the physical hosts running the hypervisors and the virtual machines. The sensor module also collects the application's performance metrics. These metrics are processed through the model building module, which will output a model with an appropriate set of metrics. The model updating module will identify when the model's predictions have significantly diverged from the observed performance via hypothesis testing over the residuals. If the model passes the hypothesis testing, this shows that it still accurately captures the relationship between the system metrics and application's performance. However, if the model fails the hypothesis testing, it is considered unsuitable for the current situation and a new model will be constructed. vPerfGuard is evaluated through experiments using a set of common benchmarks in a number of usage scenarios common in Cloud environments, including VM colocation and consolidation. vPerfGuard can (1) automatically identify the critical system resource that affects the application's performance, and (2) adaptively change the critical system resource when the application workload or the execution environment changes.

ERController integrates hierarchical resource management into a classical resource allocation controller on both application and container levels to achieve the highest resource utilization while guaranteeing service-level compliance. More concretely, the application-level resource allocation is implemented by an adaptive feedback controller, which dynamically decides the total resource allocation that is required for the application to meet service-level agreements for the time-varying workload. The container-level resource allocation is implemented by a globally-optimizing resource partitioner, which partitions the total resource budget among the components of the application so that the highest resource utilization is achieved. ERController is evaluated with standard web application benchmarks, under different traffic patterns, service-level agreements, and three different workload models—open, closed, and semiopen. Our evaluation indicates two major advantages of ERController in comparison to classical resource allocation controllers. First, fewer resources (20% less) is provisioned to the applications to achieve the same service-level compliance compared to the classical methods. This shows that ERController achieves higher resource utilization while guaranteeing service-level compliance. Second, our approach is robust enough to address various types of workloads with time-varying resource demands without reconfiguration.

Some of the technical contributions such as ActiveSLA and ERController have already been published. The related systems and papers [125, 124, 126, 127, 121, 123, 122, 117, 65] are summarized as shown in Table 1.

1.6 Organization of this thesis

The remainder of this thesis is organized as follows. The building blocks and tools are introduced in Chapter 2. Chapter 3 to Chapter 5 present ActiveSLA, vPerfGuard and ERController, respectively. Chapter 6 compares our work with classical methods. Finally, Chapter 7 gives conclusion.

Table 1: Summary of ActiveSLA, vPerfGuard and ERController

	ActiveSLA	vPerfGuard	ERController
Assumption	Observable and controllable systems		
Classical control based approach	admission control	critical resource identification	resource allocation
Enhancement	decision theory	statistical machine learning	hierachical resource management
Monitoring	Application's performance, workload and system resources		
Modeling	non-linear classification model	time-varying non-linear/linear models	M/G/1/PS queueing model
Experiments	TPC-W	RUBBoS, TPC-H	RUBiS benchmark
	Stationary/Non-stationary workload with different types, different intensities and different SLAs		
Improvements	obtain the best service-level compliance, 20% more total profit	automatically and adaptively identify critical resources, significantly save manpower	achieve the highest resource utilization, 20% less resource cost
Publications	SOCC [125] SIGMOD-PhD-Symp. [122]	SOCC(submitted) [127]	NOMS [121], ICDCS [126]

CHAPTER II

PRELIMINARY BUILDING BLOCKS AND TOOLS

Experienced readers who are familiar with the preliminary building blocks and tools in the thesis can skip this chapter. Interested readers will learn from this chapter that, although the preliminary building blocks and tools are sophisticated and seem rather difficult to apply to computer systems, they are quite useful for the fascinating problems after careful application and enhancement. This is exactly one of the major contributions of the thesis.

In this chapter, each building block or tool is introduced by references for more detailed information. The potential and promising application scenarios of each building block or tool are shown. The limitation of each building block or tool when it is directly applied to the scenario is illustrated. Finally, the efforts made to overcome the limitation to solve the interesting problems through enhancement of the building block or tool as well as integration with other building blocks or tools are presented.

This chapter is organized as follows. An overview of the relationship between the individual building blocks and the entire systems is given in Section 2.1. The control theory basis is discussed in Section 2.2. Then intelligent control, optimal control, adaptive control and hierarchical control are discussed in more details from Section 2.3 to Section 2.6. Section 2.7 presents the ELBA framework before Section 2.8 summarizes the chapter.

2.1 Road map

In this thesis, three systems (i.e., ActiveSLA, vPerfGuard and ERController) are presented to address the increasing complexity of automated performance and resource management for applications in Cloud computing. The building blocks and tools for

the systems are primarily composed of control strategies [43, 33, 61] and the ELBA [11] (Automated N-Tier Application Deployment) framework . Different control strategies are combined and enhanced as shown in Table 2. All the systems leverage the ELBA [11] framework for application deployment, evaluation, reconfiguration, and redesign.

Table 2: Building blocks and tools used in ActiveSLA, vPerfGuard and ERController

	ActiveSLA	vPerfGuard	ERController
Adaptive control	Yes! Accepted query with execution time is feedback into the model	Yes! Online change point detection module adapts to the changes	Yes! RLS method is used to build adaptive models
Hierarchical control	No	No	Yes! The outer-level application controller works closely with the inner-level resource partition controller
Intelligent control	Yes! Statistical methods Logitboost and Additive Regression are used	Yes! Statistical method t-testing is used for online change-point detection	No
Optimal control	Yes! The objective is to maximize the total profit(revenue)	No! The objective is to track the application's performance	Yes/No. The objective is to track SLA or minimize resource cost
The ELBA framework	Yes! Support to run TPC-W application	Yes! Support to run RUBBoS, TPC-H applications	Yes! Support to run RUBiS application

ActiveSLA, which will be illustrated in Chapter 3, is a combination of adaptive, intelligent and optimal control-based on the ELBA framework. An accepted query and its execution time are fed into the model which keeps the model adaptive. Statistical methods such as Logitboost and Additive Regression are used to enable the intelligent control decisions. Finally, the control objective is to optimize the total profit by achieving the best service-level compliance.

vPerfGuard, which will be illustrated in Chapter 4, is a combination of adaptive and intelligent control-based on the ELBA framework. An online change-point detection module triggers the update of the model when it detects the change-point, which makes the model adaptive. Statistical method such as t-testing is used for

online change-point detection. Finally, the control objective is to track the application's performance reference, which is similar to the tracking objective of a classic controller.

ERController, which will be illustrated in Chapter 5, is a combination of adaptive, hierarchical and optimal control-based on the ELBA framework. A Recursive least square(RLS) method is used to build adaptive models, which work at different operating points. A hierarchical control approach where an outer-level application controller works closely with an inner-level resource partition controller is adopted. Finally, the control objective, to maintain the application's performance within the SLA bound, is achieved. At the same time, another control objective, to optimize resource cost, is also achieved.

2.2 Control theory basis

Control theory [43, 33] is an interdisciplinary branch of engineering and mathematics that deals with the behavior of dynamical systems [10, 129].

Control systems can be thought of as having three functions, measure, compare/compute, and correct [43, 33] (i.e., corresponding to monitoring, modeling and management functions in this thesis). Figure 3 illustrates a standard feedback control loop. The system being controlled is referred to as the target system, which has a set of metrics of interest (referred to as measured output) and a set of control knobs (referred to as control input). The controller periodically adjusts the value of the control input such that the measured output can match(track) its desired value (referred to as reference input) specified by the system designer. In other word, the controller aims to maintain the difference between the two (referred to as control error) at zero, in spite of any disturbances in the system. The disturbance is defined as interference, which is not under control that affects the measured output of the target system.

Although control theory [43, 33] has been established for more than a century, it is

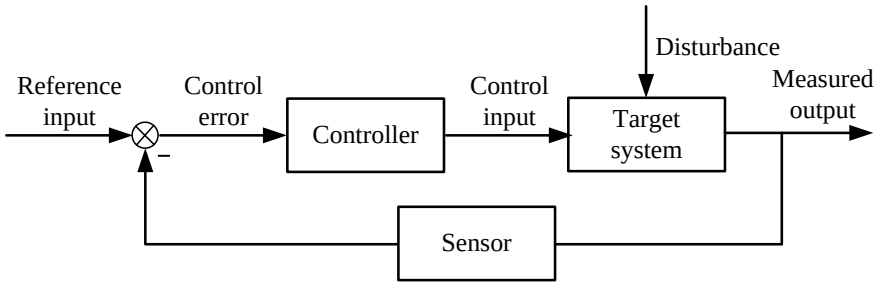


Figure 3: Standard feedback control loop

still an evolving field to apply control theory to computer systems [61]. Although this field is promising and receives more attention in recent work, a major problem still remains, i.e., how to apply control theory or how to enhance a classical control-based approach to solve important and interesting computer system problems.

2.2.1 Closed-loop

Introduction If we assume that the controller C, the plant P, and the sensor F are linear and time-invariant (i.e., elements of their transfer function $C(s)$, $P(s)$, and $F(s)$ do not depend on time) as shown in Figure 4, the systems above can be analyzed using the Laplace transform on the variables. This gives the following relations:

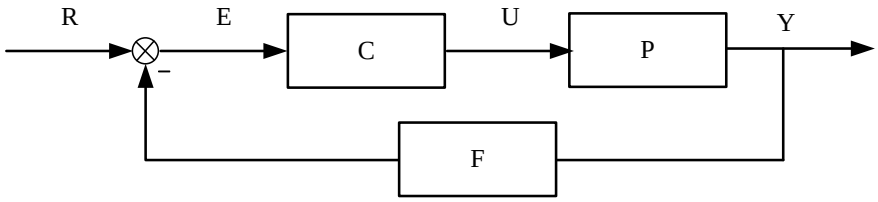


Figure 4: Closed-loop transfer function

$$Y(s) = P(s)U(s)$$

$$U(s) = C(s)E(s)$$

$$E(s) = R(s) - F(s)Y(s)$$

Solving for $Y(s)$ in terms of $R(s)$ gives:

$$Y(s) = (\frac{P(s)C(s)}{1 + F(s)P(s)C(s)})R(s) = H(s)R(s)$$

The expression

$$H(s) = \frac{P(s)C(s)}{1 + F(s)P(s)C(s)}$$

is referred to as the closed-loop transfer function of the system [43, 33]. The numerator is the forward (open-loop) gain, and the denominator is one plus the gain in going around the feedback loop, the so-called loop gain [43, 33]. If $|P(s)C(s)| \gg 1$, i.e., it has a large norm with each value of s, and if $|F(s)| \approx 1$, then $Y(s)$ is approximately equal to $R(s)$ and the output closely tracks the reference input. Note that the Laplace transform is often used in continuous time domain while the Z transform is often used in discrete time domain [61], accordingly.

Application scenario and limitations The classical control theory with the concepts of closed-loop could be applied to interesting problems in system management. For example, a service-level agreement (or SLA) is a contract between a service provider (e.g., a platform-as-a-service provider) and its clients. An agreement consists of one or more service-level objectives (SLOs). An example of an SLO is: "Gold customer response times should be less than 5 seconds." [61]. An SLO is composed of three parts: the metric (e.g., response time), the bound (e.g., 5 seconds), and a relational operator (e.g., less than). Intuitively, service providers want to have sufficient resources to meet their SLOs. But they do not want to have more resources than required since doing so imposes unnecessary costs. As a result, SLO enforcement often becomes a regulation problem in classical control theory, which ensures that the measured output is equal to (or near) the reference input. In terms of the control architecture in Figure 3, the SLO metric is the measured output, and the SLO bound is the reference input. This problem is discussed further in Chapter 5.

2.2.2 PID controller

Introduction The most-used feedback control design is probably the PID controller [43, 33, 61]. PID is an acronym for Proportional-Integral-Derivative, referring to the three terms operating on the error signal to produce a control signal. If $u(t)$ is the control signal sent to the system, $y(t)$ is the measured output and $r(t)$ is the desired output, and tracking error $e(t) = r(t) - y(t)$, a PID controller has the general form

$$u(t) = K_P e(t) + K_I \int e(t)\mathrm{d}t + K_D \frac{\mathrm{d}}{\mathrm{d}t} e(t).$$

The desired closed-loop dynamics are obtained by adjusting the three parameters K_P, K_I and K_D. The proportional term ensures stability. The integral term ensures that there is no static error in the stable state. The derivative term is used to provide damping or shaping of the response [43, 33, 61].

Applying Laplace transformation results in the transformed PID controller equation

$$u(s) = K_P e(s) + K_I \frac{1}{s} e(s) + K_D s e(s) u(s) = (K_P + K_I \frac{1}{s} + K_D s) e(s)$$

with the PID controller transfer function

$$C(s) = (K_P + K_I \frac{1}{s} + K_D s)$$

Note that while a PID controller could be specified using the Laplace transform in a continuous time domain, a PID controller could also be designed using the Z transform in a discrete time domain [61], accordingly.

Application scenario and limitations As PID controllers are the most well established class of control systems, they show great potential for the SLO enforcement problem. For example, a PID controller could be designed to maintain the application's performance metric within the SLO bound [42].

The main obstacle is how to guarantee the controller's performance, especially the stability of the controlled system. The Root Locus [61] method is applied to design the controller parameters so that the controller's performance such as the setting time and the overshoot are within an acceptable limit, which guarantees the stability of the system. More details of deriving the controller parameters could be found in Chapter 5.

2.3 Intelligent control and statistical methods

Intelligent control [22] uses various artificial intelligence computing approaches such as statistical methods [60, 49], machine learning [90], decision theory [53, 92] to control a dynamic system.

2.3.1 Statistical method

Introduction Statistics is the study of the collection, organization, analysis, and interpretation of data [60, 49].

A common goal for a statistical method is to investigate causality, and in particular to draw a conclusion on the effect of changes in the values of predictors or independent variables on dependent variables or response as shown below.

$$\textit{predictors or independent variables} \Rightarrow \textit{dependent variables or response}$$

There are two major types of causal statistical studies [60, 49]: experimental studies and observational studies. These two types of studies are similar to each other because the effect of differences of an independent variable (or variables) on the behavior of the dependent variable are observed in both types of studies. These two types of studies are different from each other in aspect of how the study is actually conducted. The former one involves taking measurements of the system under study, manipulating the system, and then taking additional measurements which use the same procedure to determine if the manipulation has modified the values of the measurements. In

contrast, the latter one does not involve experimental manipulation. Instead, data are gathered and correlations between predictors and response are investigated.

Application scenario and limitations Both the experimental studies and the observational studies are promising approaches to apply to computer system management.

For example, a common scenario for a platform-as-a-service provider who hosts a distributed web application is to identify the critical resource that affects the application's performance. A statistical method with an observational study is promising to solve this problem. The main motivation is two-fold, (1) the common goal for a statistical method is to investigate causality, and in particular to draw a conclusion on the effect of changes in the values of critical resource metrics on application's performance as shown below; and (2) the objective of the statistical method is to identify the critical resource where no manipulation is involved.

$$\textit{which critical resource metric ?} \Rightarrow \textit{application's performance metric}$$

However, there is also an obstacle that we need to overcome before we apply the statistical method. For a distributed web application in a Cloud environment, each component of the application could be deployed in a different host while each host could have thousands of metrics to observe, which makes the critical resource identification using statistical methods difficult. Our solution is to employ a filtering method, i.e., first select the metrics according to the correlation coefficients and then select the metrics according to the prediction accuracy of a specific model. Finally, the model will point out the critical resource. More details could be found in Chapter 4.

For another example, a common scenario for a Database-as-a-service(DaaS) provider is to estimate query execution time and make an admission control decision based on the estimation. A statistical method with experimental study is potentially helpful in deriving the query execution time distribution. The main motivation is two-fold, (1)

the common goal for a statistical method is to investigate causality, and in particular to draw a conclusion on the effect of changes in the values of query features, database system settings and parameters on query execution time as shown below; and (2) the objective of a statistical method is to assist admission control decisions that involve manipulating the workload to the system.

$$\textit{query features, settings and parameters} \Rightarrow \textit{query execution time}$$

However, there is also an obstacle that we need to overcome before we apply statistical methods. A major limitation is that, some of the query features, system settings and parameters are not explicitly available. Our solution is to develop specific sensors to collect those information. For example, we append a sensor into PostgreSQL database engine, which reports the number of sequential I/O for a query. For another example, we write a sensor program inside the Linux operating system kernel to report the proportion of a database table file in system cache. More details could be found in Chapter 3.

2.3.2 Machine learning

Introduction Machine learning [90], a branch of artificial intelligence [101], is a scientific discipline concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data. Data is considered as examples that illustrate relations between observed variables. The examples (data) are utilized by a learner to capture characteristics of interest of their unknown underlying probability distribution. One of the major objectives of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data.

Application scenario and limitations There is a well-founded skepticism about whether the simple models often used in the research literature can capture complex

real-life workload/resource/performance relationships and keep up with changing conditions that might invalidate the models [129]. Due to the data-driven characteristic of machine learning, a model based on statistics and machine learning method is promising to apply to the above scenario.

However, there are also obstacles that we need to overcome. The most critical problem is how to find out the most appropriate machine learning model. Our solution is to choose the most suitable model that fits our control objective. For example, we use a non-linear regression machine learning model called "Additive Regression" when we model the relationship between the SLA penalty cost and the system resources such as CPU shares and memory size in [124]. The motivation to choose such a non-linear regression model is because the memory size affects the SLA penalty cost in a non-linear way. We compare different machine learning modeling approaches in Chapter 3 and Chapter 4 and show that different machine learning models need to be adopted according to different control objectives.

2.3.3 Decision theory and risk assessment

Introduction Decision theory [53] is concerned with identifying the values, uncertainties and other issues relevant in a given decision, its rationality, and the resulting optimal decision. It is widely applied in many domains, such as economics, psychology, philosophy, mathematics, and statistics. It is closely related to the field of game theory [92], which is defined as "the study of mathematical models of conflict and cooperation between intelligent rational decision-makers".

"Choice under uncertainty" represents the heart of decision theory. The procedure now referred to as expected value was known from the 17th century [53]. The idea of expected value is that, when faced with a number of actions, each of which could give rise to more than one possible outcome with different probabilities, the rational procedure is to identify all possible outcomes, determine their values (positive or

negative) and the probabilities that will result from each course of action, and multiply the two to give an expected value. The action to be chosen should be the one that gives rise to the highest total expected value as shown below.

$$\left.\begin{array}{l} \textit{Action 1} \Rightarrow \sum_{outcome} \textit{value} * P_{outcome} \Rightarrow \textit{Expected value 1} \\ \textit{Action 2} \Rightarrow \sum_{outcome} \textit{value} * P_{outcome} \Rightarrow \textit{Expected value 2} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \ldots \\ \textit{Action n} \Rightarrow \sum_{outcome} \textit{value} * P_{outcome} \Rightarrow \textit{Expected value n} \end{array}\right\} \Rightarrow \textit{Action ?}$$

Application scenario and limitations Consider a service-oriented, utility-computing scenario where service providers will execute jobs on behalf of their clients on systems rented from resource providers, e.g., a Database-as-a-service(DaaS) provider who executes queries on behalf of their clients. Each query is related with a utility function or a service-level agreement where the satisfaction of SLAs within the agreement such as meeting the deadline will bring gain and the violation of SLAs will result in penalties. It is impossible for a DaaS provider to admit all the queries because that will cause thrashing due to buffer pool over-utilization for the DBMS. The DaaS provider needs to selectively admit queries that can produce the most revenue. To complicate matters, the DaaS provider is uncertain about whether admitting query will bring in revenue due to the indeterminacy of the query execution environment. This motivates us to use decision theory, which is promising to identify the uncertain issues relevant in a given decision and rationally assess the risk of admitting a query as shown below.

$$\left.\begin{array}{l} \textit{Admit query} \Rightarrow \sum_{outcome} \textit{value} * P_{outcome} \Rightarrow \textit{Expected value for admit} \\ \textit{Reject query} \Rightarrow \sum_{outcome} \textit{value} * P_{outcome} \Rightarrow \textit{Expected value for reject} \end{array}\right\} \Rightarrow \textit{Action ?}$$

Although it seems quite promising to apply decision theory for a DaaS provider, there is also an obstacle that we need to overcome. The most critical problem is how

to obtain the expected values under different actions such as admit or reject where an expected value under an action is a multiplication of the possible outcome values and the probabilities of the outcome that will result from an action. Although the possible outcome values could be easily obtained from service-level agreements, it is really difficult to acquire the probabilities of the outcome, such as meeting/missing service-level agreements.

Our solution is to adopt a machine learning method called "LogitBoost" [48], which is a type of regression analysis used for predicting the outcome of a categorical (a variable that can take on a limited number of categories) criterion variable based on one or more predictor variables. Following the "LogitBoost" machine learning method, we could derive the probabilities of the outcome, such as meeting/missing service-level agreements, based on query features, database system settings and parameters. More details of how to use decision theory and risk assessment for a DaaS provider could be found in Chapter 3.

2.4 Optimal control

Introduction Compared with a classic control technique in which the control signal enables the measured output to track the reference input, optimal control [107] is a particular control technique in which the control signal optimizes a certain "cost index". For example, in the case of a satellite, optimal control is used to regulate the jet thrust to bring the satellite to desired trajectory with the least amount of fuel consumption. Here the "cost index" refers to the fuel consumption.

Application scenario and limitations Different from regulatory control, which could be applied to help a service provider to ensure that the measured output (SLO) is equal to (or near) the reference input (SLO), optimal control could be used to help a service provider optimize a certain "cost index". The most common "cost index" is profit, e.g., a DaaS provider who executes queries on behalf of their clients needs

to selectively admit queries that can optimize the profit.

However, there is also an obstacle that we need to overcome. The most critical problem is how to build a model that connects the "cost index" with the control actions. For example, in the scenario for a DaaS provider, how to relate the admission/rejection control action with the expected revenue. Our solution is to apply decision theory and risk assessment to derive the expected value of revenue for the admission/rejection control actions. During this process, statistical machine learning techniques are also leveraged. More details of how to use optimal control method to optimize the profit could be found in Chapter 3.

2.5 Adaptive control

According to Section 2.2, the following relation can be derived using Laplace transform on the variables.

$$U(s) = C(s)E(s)$$

Here $C(s)$ is the controller gain from error $E(s)$ to the input of system $U(s)$. If a PID controller is applied,

$$C(s) = (K_P + K_I \frac{1}{s} + K_D s)$$

As shown in Figure 5 with dotted lines, in adaptive control [25], the controller gains [43, 33] are modified online by identification of the system parameters. Compared with non-adaptive control where the controller gains are static, adaptive control is designed to work assuming that the parameters of the system being controlled are slowly time-varying or uncertain. The adaptive characteristic guarantees strong robustness properties for the controller. We mainly focus on two techniques, i.e., system identification and online change-point detection.

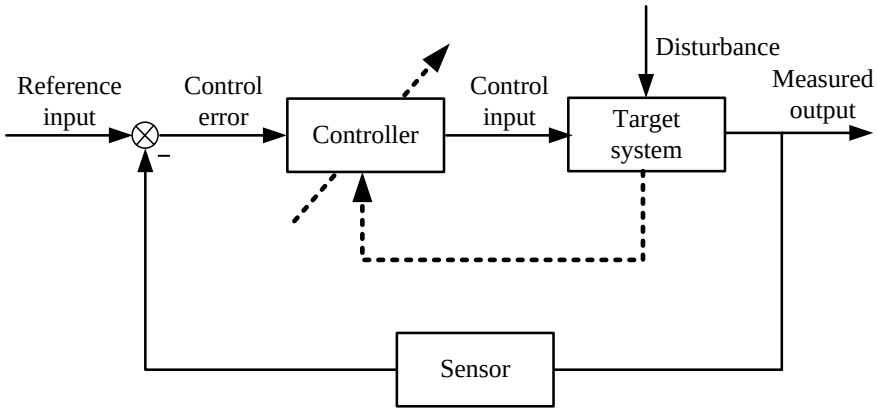


Figure 5: Adaptive control

2.5.1 System identification

Introduction In control engineering, the field of system identification [83] uses statistical methods to build mathematical models of dynamical systems from measured data. A dynamical mathematical model in this context is a mathematical description of the dynamic behavior of a system or process in either the time or frequency domain.

Application scenario and limitations Due to the complexity of a computer system, a system identification approach is promising to derive a mathematical model of a computer system by varying the inputs in the operating region and observing the corresponding outputs. For example, under different workload intensities, the system models that describe the relationship between the system input (e.g., resource allocation) and the system output (e.g., mean response time) could change. A system identification approach shows the potential to derive a mathematical model for the relationship. Moreover, due to the constantly varying workload intensities, a system identification approach ought to be done in an online manner because the system model is continuously evolving. After that, a new controller needs to be designed based on the mathematical model.

However, there is also an obstacle that we need to overcome. The most important problem is what type of model should we adopt. Different type of models will have different effects on the design of controllers. Our solution is to leverage an Autoregressive-moving-average(ARMA) model [32]. The model consists of two parts, an autoregressive (AR) part and a moving average (MA) part. This model not only captures relationship between system inputs and outputs, but also captures the relationship among system inputs and outputs in time series. Moreover, this model could be seamlessly integrated with standard controller design approach such as the Root Locus [61] method. More details of how to construct online system identification using ARMA model could be found in Chapter 5.

2.5.2 Online change-point detection

Introduction Online change-point detection is a model management technique from statistical machine learning domain to adjust the models when changes are observed in application's performance [28]. A performance model should be discarded or modified when it no longer accurately captures the relationship among workload, resource and performance. In practice, this relationship could be altered by software upgrades, transient hardware failures, or other changes in the environment.

The accuracy of a model is usually estimated from the residuals, i.e., the difference between the measured performance of the application and the prediction of the model [28]. Under steady state, the residuals should follow a stationary distribution, thus a shift of the mean or increase of variance of this distribution indicates that the model is no longer accurate and should be updated. Online change-point detection techniques use statistical hypothesis testing [60] to compare the distribution of the residuals in two subsequent time intervals of constant or different lengths. If the difference between the distributions is statistically significant, we start training a new model. The magnitude of the change will influence the detection time; an

abrupt change should be detected within minutes, while it might take days to detect a slow, gradual change. The result of the change-point test is a p-value; the lower the p-value, the higher the probability of a significant change in the mean of the normalized performance signal.

Application scenario and limitations In a consolidated Cloud environment, a performance model of an application could be altered not only by the workload characteristic change such as migration from CPU-intensive workload to IO-intensive workload, but also by the resource contention from co-located neighbor virtual machines. Online change-point detection shows great potential to detect the changes in the application's execution environment. By comparing the distribution of the residuals of the model through hypothesis testing, i.e., the difference between the measured performance of the application and the prediction of the model, online change-point detection identifies the possible shift of the performance model of an application.

However, there is also an obstacle that we need to overcome. The most critical problem is how to build a model using thousands of metrics. A performance model that considers all combinations of the thousands of metrics can be computationally expensive to construct and can easily lead to model over-fitting. Our solution is to use a two-phase algorithm that reduces the modeling complexity by: first (in phase 1) selecting a small number of candidate metrics that are most strongly correlated with the application's performance; and then (in phase 2) identifying even fewer predictor metrics that can give the best prediction accuracy for a specific model from among the candidate metrics. More details of the two-phase algorithm could be found in Chapter 4.

2.6 Hierarchical control

Introduction Most of the human-built systems with complex behavior are often organized as a hierarchy. A hierarchical control system [68, 22] is a type of control

system in which a set of devices and governing software is arranged in a hierarchical tree. The hierarchical control system is also a form of networked control system if the links in the tree are implemented by a computer network.

Each element of the hierarchy is a linked node in the tree. Commands and goals to be achieved flow down the tree from superior nodes to subordinate nodes, whereas sensations and command results flow up the tree from subordinate to superior nodes [68]. If necessary, nodes may also exchange messages with their siblings. The two distinguishing features of a hierarchical control system are related to its layers. (1) Superior nodes in a higher layer of the tree operate with a longer interval of planning and execution time than subordinate nodes in its immediately lower layer. The superior nodes, having relaxed time constraints, are capable of reasoning from an abstract world model and performing planning in a global manner. (2) The subordinate nodes in a lower layer have local tasks, goals, and sensations, and their activities are planned and coordinated by superior nodes in higher layers. The subordinate nodes, having time constraints, are capable of reasoning from a more concrete world model and performing planning in a local manner.

Application scenario and limitations A hierarchical control system is promising to apply to solve the multi-level resource allocation problem for a multi-tier web application. The higher level of the control system operates with a longer interval of planning and execution time to decide the global resource budget for the whole application. The lower level of the control system partitions the total resource budget to each tier and enforces the resource allocation for each tier in a shorter interval of planning and execution time. The two levels of the control system collaborate closely with each other in a hierarchical way.

However, there are also problems that we need to overcome in order to apply the hierarchical control system. The first problem is how to decide the total resource

budget while meeting service-level agreements. The second problem is how to partition the total resource budget to each tier. Our solution to the first problem is to leverage a proportional and integral controller. The difference between the real performance and the reference performance in service-level agreements will be the input of the controller. The controller will then output the total resource budget so that the real performance will track the reference performance. Our solution to the second problem is to leverage queueing theory and Lagrange multiplier. Then the optimal partition could be achieved. More details about applying hierarchical control system to solve the multi-level resource allocation problem for a multi-tier web application could be found in Chapter 5.

2.7 The ELBA framework

Introduction One of the main research challenges in the adaptive enterprise vision is the automation of large application system management [111]. The detailed staging process not only encompasses design, deployment, and production use but also captures application monitoring, evaluation, and evolution.

An application system deployment plan needs to be verified and tested before committed to a production environment. Manual verification of a deployment is cumbersome, time consuming, and error prone. This problem will grow in importance in the deployment of increasingly larger and more sophisticated applications. For example, a typical multi-tier distributed application such as RUBiS [14] or RUBBoS [7] is composed of tens of servers [87]. Manual deployment of such sophisticated application is really burdensome. Therefore, it will be increasingly important to have an automatic method for executing an application on the deployment plan to validate the deployment during staging, instead of debugging a deployment during production use.

However, current approaches to the staging process are mostly manual, complex

and time-consuming. ELBA [11, 111, 99, 110, 70] is the proposed framework for automated design, configuration, evaluation and tuning. The framework intends to automate the staging process thus reducing the time and manual labor involved in the process, increase confidence, and extract predictive performance data. Further, the automation will support a more thorough application test and validation in a larger state space.

The overall ELBA [11] framework is shown in Figure 6, where it achieves full automation in system deployment, evaluation, and evolution, by leveraging a code generation tool "Mulini" to link the different steps of deployment, evaluation, reconfiguration, and redesign in the application lifecycle. Mulini [111] adopts XSLT/XPath tools and aspect-oriented programming (AOP) techniques to manipulate XML-encoded high-level specifications and weave non-functional specifications into staging implementation. Mulini transforms input specification files (XML-encoded) into an intermediate XML-encoded specification with annotations at the first transformation step. During the rest of transformation steps, modules (aspects) are woven into the intermediate specification using the annotations. The important advantages of the Mulini code generation process over compilation stem from its extensibility and flexibility by utilizing XML and XML manipulation standard languages (i.e., XPath, XSLT, XQuery, and XQueryUpdate).

Application scenario and limitations The ELBA framework (particularly, the Mulini code generator) could be leveraged to generate and manage the experiments. Then automated analysis techniques and tools could be used to digest the information and create a performance model. Finally, the management actions are taken based on the model.

For example, the code generation process for installing an Apache web server which is used in RUBBoS [7] in Chapter 4 and RUBiS [14] in Chapter 5 is shown

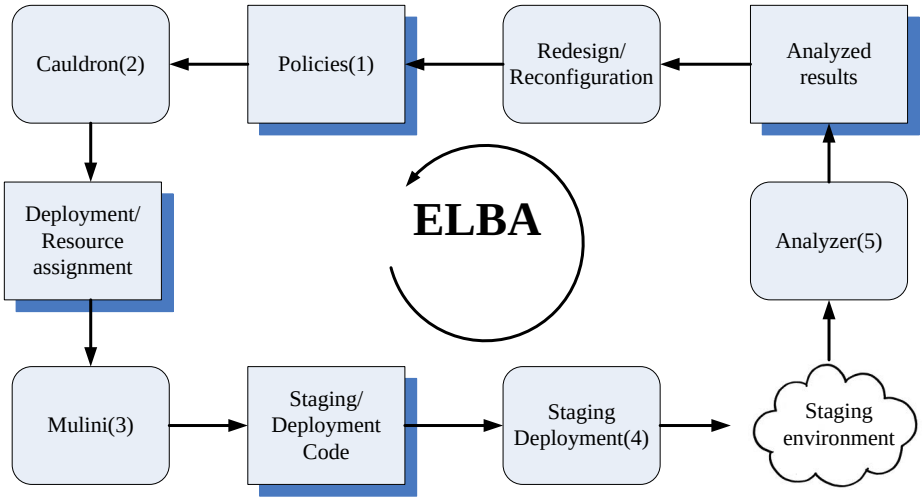


Figure 6: The ELBA framework

in Figure 7. The initial template for the installation of an Apache web server is specified in the file "WEBinstall.xsl". This file is then used by Mulini to translate the experiment specification file into the executable shell script. For example, the notation "OUTPUT_HOME" is translated into a real file path "output".

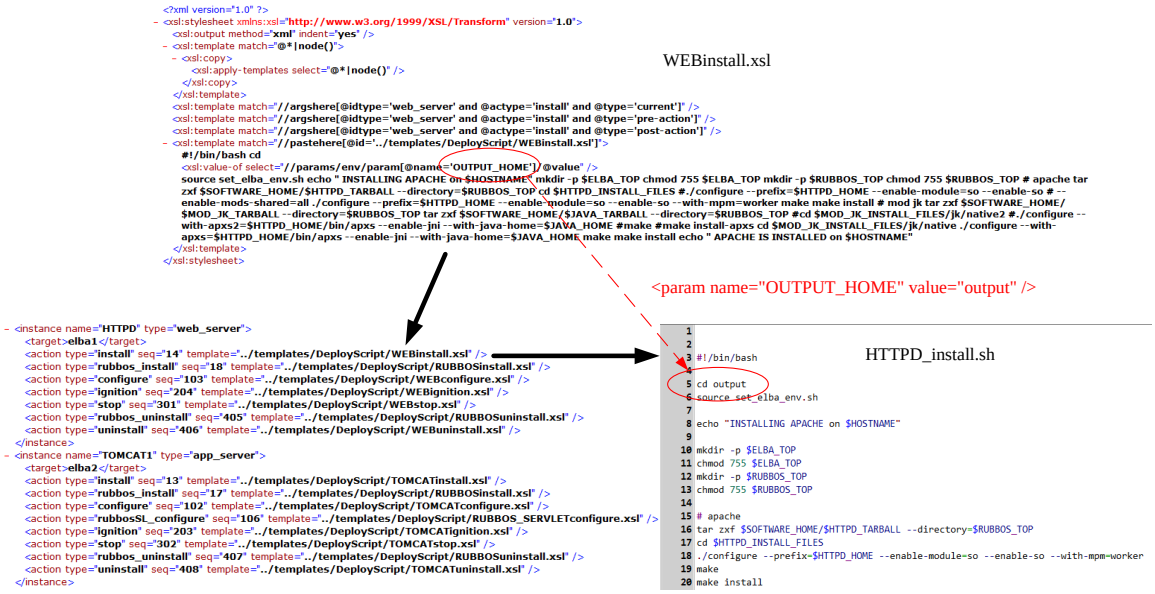


Figure 7: Application deployment

However, there are also problems that we need to solve before the ELBA framework is applied. The first problem is to write application and environment specific

templates. As we run our experiments using different applications, such as RUBBoS [7] and RUBiS [14], and different environments, such as Xen enabled virtualized environment [27] and ESX enabled virtualized environment [17], different templates should be generated based on different applications and different environments. Our solution is to study the characteristics of specific applications and environments and then generate the templates by taking into consideration of the characteristics. The second and the most important problem is to seamlessly connect the ELBA framework with the control-based approaches. More specifically, the problem is how to incorporate the control-based approaches with the analyzer in Figure 6. Our solution is to carefully study the input and the output of the staging environment, the control-based approaches and the redesign/reconfiguration. Then the communication parts from the staging environment to the control-based approaches and the communication parts from the control-based approaches to the redesign/reconfiguration are appended into the new templates.

2.8 Summary

In this chapter, the preliminary building blocks and tools are summarized, which will be used to build three systems, i.e., ActiveSLA, vPerfGuard and ERController in the following chapters. This chapter not only provides references to the building blocks and tools for craving readers looking for detailed information but also illustrates both the promising side and the difficult side of applying the building blocks and tools to solve the interesting computer system problems. The most important part is that, it also shows how to overcome the limitations to solve the fascinating problems through careful application and enhancement of the building blocks and tools. More detailed application and enhancement could be found in the following chapters.

CHAPTER III

ADMISSION CONTROL FEATURING RISK ASSESSMENT

The classical control-based approaches addressing the resource sharing dilemma could be classified into three major categories, i.e., admission control, queueing and scheduling, and resource allocation as shown in Figure 2. In this chapter, a classical admission control is enhanced by leveraging decision theory to address novel technical challenges due to the diverse gain and penalty values, uncertain resource pool load and other issues relevant in an admission decision in order to achieve the most profitable service-level compliance for Database-as-a-service(DaaS) providers.

3.1 Background

Efficient data processing is always a fundamental issue for almost every scientific, academic, or business organization. Advances in Cloud computing technologies have triggered a Database-as-a-service(DaaS) model [39, 40, 41, 56] for data processing. The model allows organizations to leverage hardware and software solutions provided by DaaS providers, without having to develop them on their own, thereby freeing them to concentrate on their core businesses.

In a Cloud computing environment, when many queries are submitted to the database management system during a busy period, most of them will not finish on time. This usually has direct economic impact on the service provider, who has to pay penalties if the application's performance does not meet clients' service-level agreements (SLAs). Classical admission control often limit the total number of active

transactions at any point in time, requiring some transactions to wait to be admitted in order to prevent thrashing and buffer-pool over-utilization [37, 34]. In this chapter, we propose ActiveSLA, which enhances classical admission control by leveraging risk assessment to help DaaS providers achieve the most profitable service-level compliance.

The rest of this chapter is organized as follows. Section 3.2 and Section 3.3 give the problem definition and the solution overview, respectively. Sections 3.4 and 3.5 describe the prediction module of ActiveSLA and corresponding experimental studies, respectively. Sections 3.6 and 3.7 describe the decision module of ActiveSLA and corresponding experimental studies, respectively. Finally, Section 3.8 summarizes the chapter.

3.2 Problem definition

Figure 8 shows the scenario where admission control is adopted as a control-based approach to help a DaaS provider achieve more profit.

We assume that the satisfaction of SLAs will bring gain while the violation of SLA will result in penalties. The satisfaction of SLA determines the revenue that a DaaS provider can obtain. Following the previously defined equation where "Profit = Revenue - Cost", if we assume that the cost for a service provider is constant, we need to maximize the revenue in order to maximize the profit. Under this assumption, we use "profit" and "revenue" interchangeably in this chapter. The problem is how to make admission control decisions (i.e., admit/reject the queries) to the database management system so that a DaaS provider can achieve the most profitable service-level compliance.

A classical admission control-based approach makes admission decisions based on the number of requests [46] or based on the number of each type of request [112]. A classical admission control-based approach offers limited help for a DaaS provider to

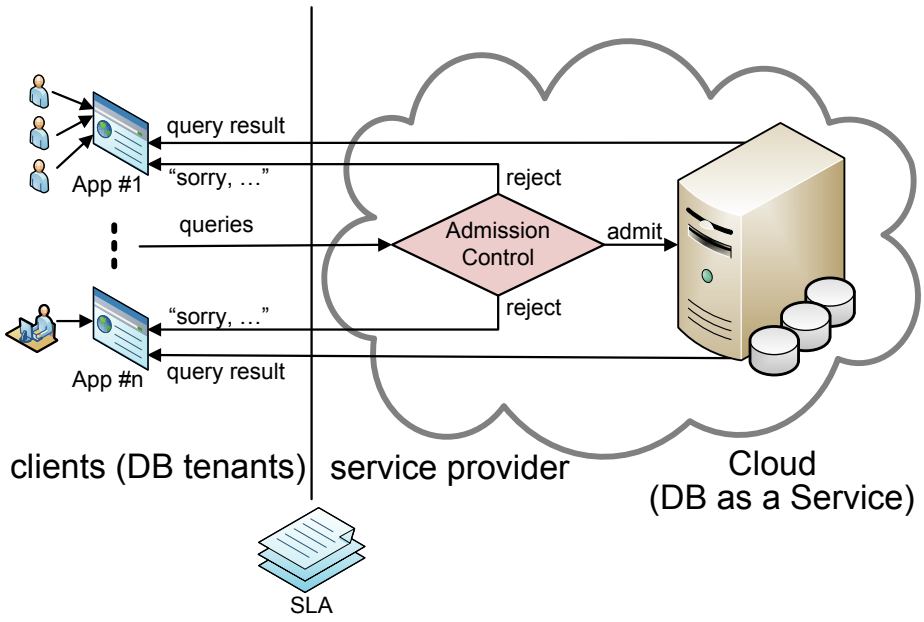


Figure 8: Admission control problem definition

achieve the profitable service-level compliance because the diverse gain and penalty values, uncertain resource pool load and other issues relevant in an admission decision present novel technical challenges. The challenges demand enhancement to a classical admission control approach in order to achieve the most profitable service-level compliance. There are two key technical challenges:

The first challenge is that merely estimating the query execution time is not enough to make profit-oriented decisions in a Cloud environment. In Figure 9 we show the probability density functions (PDFs) of the execution time for two queries, where both queries have the same estimated execution time Est and the same deadline τ (specified by the SLA). Although $Est < \tau$, the chances that the two queries miss their deadlines are dramatically different. Compared with q_1, about which we are more confident that it will meet the deadline, it is more difficult to tell whether q_2 will meet or miss the deadline. If a classical admission control approach is adopted, because Est is less than τ, both of the queries will be admitted. However, from the PDF of q_2, it is much riskier to admit q_2 than to admit q_1. Thus, the probabilities of

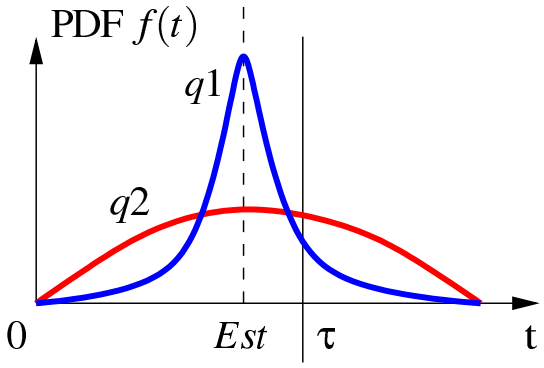


Figure 9: Two queries with the same query time estimation but different query time distributions.

a query meeting and missing its deadline are crucial in making SLA-based admission control decisions.

The second challenge is that because of SLAs, we may have to make different admission control decisions even when the queries have the same deadline and the same probability of meeting the deadline, as illustrated in the example in Figure 10. Assume that two queries q_1 and q_2 arrive simultaneously and according to certain estimates, for both the queries, the probabilities of meeting and missing their deadline are 60% and 40% respectively, as shown in Figure 10(a). In Figure 10(b) we show two different SLAs that are carried by q_1 and q_2, as described in the two utility functions. For q_1, if the query is admitted and meets the deadline, the service provider earns \$1; else if the query is admitted but misses the deadline, the service provider loses \$1; otherwise, if the service provider decides to up-front reject the query, it pays a penalty of \$0.1. For q_2, it is the same as q_1 except that the penalty for missing the deadline is \$2. If a classical admission control approach is adopted, because they have the same deadline and the same probability of meeting the deadline, we will make the same decision for both of the queries. However, simple derivations show that the expected revenue for admitting q_1 is $0.6 \times \$1 + 0.4 \times (-\$1) = \$0.2$, which is better than the penalty for rejecting the query ($-\$0.1$). Thus, the service provider should admit q_1. However, the expected revenue for admitting q_2 is $0.6 \times \$1 + 0.4 \times (-\$2) = -\$0.2$,

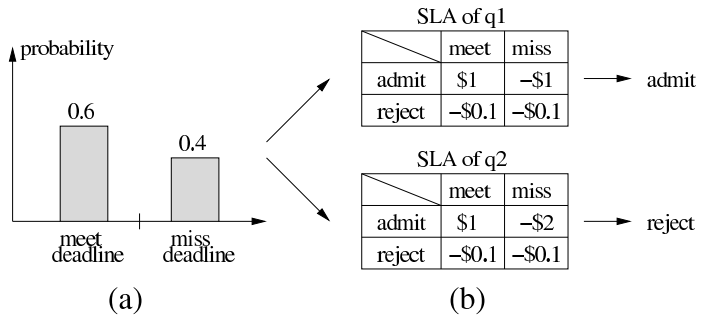


Figure 10: SLA-based admission control decisions.

which is worse than the penalty for rejecting the query. Thus, the service provider should reject q_2.

3.3 Solution approach overview

In order to address the above two challenges, in this chapter, we propose a framework, called ActiveSLA[1]. ActiveSLA demonstrates that a classical admission control approach can be enhanced by leveraging decision theory, especially risk assessment to achieve the most profitable service-level compliance.

ActiveSLA framework is an end-to-end solution that consists of two main modules: a *prediction module* and a *decision module*, as shown in Figure 11 (from a DaaS provider's point of view at the database layer [41]). When a new query arrives, the query first enters the prediction module. The prediction module uses machine learning techniques and considers both the characteristics of the query and the current system conditions. The prediction module outputs the probability that the query meets its deadline. The calculated probability and the query's SLA are sent to the decision module. The decision module decides either to admit the query or to reject the query up-front. Finally, the result of each admitted query is returned to the client and the actual execution time is sent back to the prediction module in real time. This feedback information can further help the prediction module to improve the accuracy

[1] ActiveSLA stands for: Admission Control for Profit Improving under Service Level Agreements.

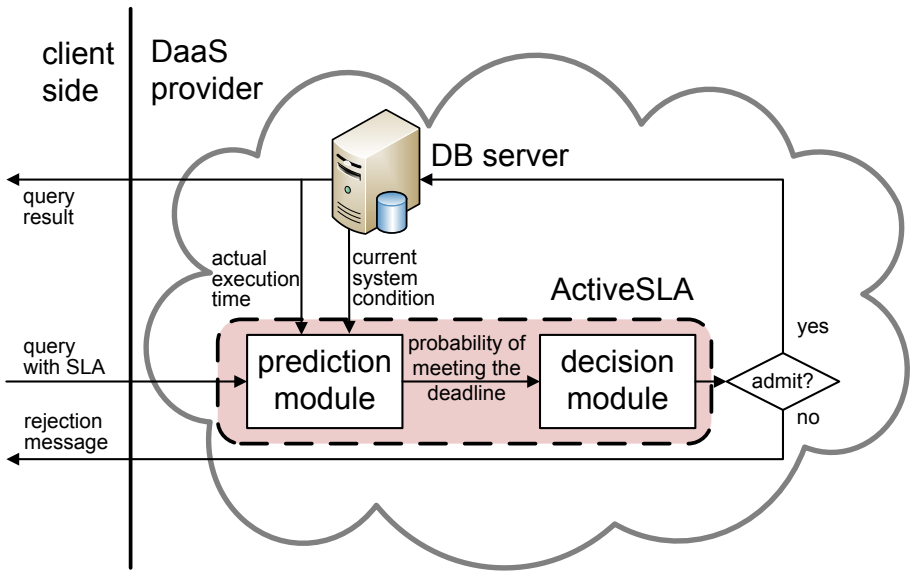


Figure 11: Overview of ActiveSLA.

of its future predictions by introducing new training data.

The main contributions of this chapter are twofold:

1. We show, by using both theoretical reasoning and empirical validation, how appropriate machine learning techniques can be successfully leveraged to answer a key question of admission control in DaaS: "What is the probability that a query meets or misses its deadline?" We implement the solution in the prediction module of ActiveSLA. The machine learning techniques (1) take many query related features as well as database system related features into consideration, (2) recognize complex patterns from the data in an automatic way, and (3) provide detailed probabilities for different outcomes.

2. We develop an SLA-based, profit optimization approach in the decision module of ActiveSLA. Decisions are made in a holistic fashion by considering (1) the probability that a new query meets its deadline under the current system condition, (2) the expected consequences of alternative actions and outcomes, and (3) the potential impact of admitting a newly arrived query on the currently running queries as well as on the future queries.

3.4 Prediction Module

In this subsection, we introduce the prediction module of ActiveSLA, which estimates the probability that a query meets its deadline in a real-time fashion.

Our approach for making such a prediction is based on machine learning techniques. Due to their data-driven characteristics, machine learning techniques can automatically recognize complex patterns from the data and provide models with remarkable performance, which is often comparable to that of from domain experts [50]. The algorithms used in this chapter are all from the off-the-shelf machine learning package WEKA [57].

However, compared with the frameworks used in previous work (e.g., [46, 50, 112]), our prediction module is addressing a new data management problem in DaaS, namely predicting the probability that a query meets its deadline. With this new problem in mind, in this subsection we discuss several design considerations, including (1) the model selection between linear and nonlinear models, between regression and classification models, and (2) the rich set of features used in our model. In the next section, we will provide detailed empirical studies to validate our design choices.

3.4.1 The Machine Learning Techniques

It is impractical to construct an accurate model that can perfectly predict the execution time of a query because of the numerous factors and their complicated interactions in modern database systems. Such a situation is captured by a well-known quote in the machine learning community, "All models are wrong, but some are useful" [31]. In this subsection, we show how we take the specific domain, namely profit-oriented admission control in DaaS, into consideration in order to select a "less wrong" model, which will be fully justified later in the experimental studies.

We compare our models with those of two classical methods, namely, TYPE (a version of Gatekeeper [46] implemented by Tozer *et al.* [112] with load shedding added)

and Q-Cop [112].

3.4.1.1 TYPE and Q-Cop, using Linear Regression

Both TYPE and Q-Cop approaches start by predicting the execution time of a query for each query type. Assuming that there are T query types (i.e., queries that share the same query template), both TYPE and Q-Cop build a model for each query type.

In TYPE, the estimated execution time of a query q_i of type i is $Est_i = e_i \times Num + E_i$, where E_i is the query execution time of q_i in a dedicated server, Num is the total number of other queries currently running in the system, and e_i is the extra delay that each additional currently running query brings to q_i. Note that TYPE is a *query-type-oblivious* approaches that is based merely on multiprogramming level (MPL) and on the mean response time of queries over all requests.

Compared with TYPE, Q-Cop uses more detailed information. Instead of counting Num, Q-Cop considers $\{n_1, \ldots, n_T\}$, i.e., the number of currently running queries of each query type (with $\sum_{j=1}^{T} n_j = Num$), which is referred to as the query mix. Based on the query mix, Q-Cop uses a linear regression model to estimate the running time of q_i as $Est_i = (e_{i1} \times n_1) + (e_{i2} \times n_2) + \cdots + (e_{iT} \times n_T) + E_i$. Here e_{ij} is the extra delay that each additional currently running query of type j brings to q_i.

3.4.1.2 ActiveSLA, using Non-linear Classification

Compared with TYPE and Q-Cop, the prediction module of ActiveSLA uses a non-linear classification model to directly predict the probability of a new query meeting its deadline, as shown in Figure 12.

Linear vs. Nonlinear Both TYPE and Q-Cop use *linear* regression, which models the relationship between the input features and the output variables by using *linear* functions. However, the execution time of a query depends on many factors in a *non-linear* fashion. In addition, many database settings, e.g., isolation levels

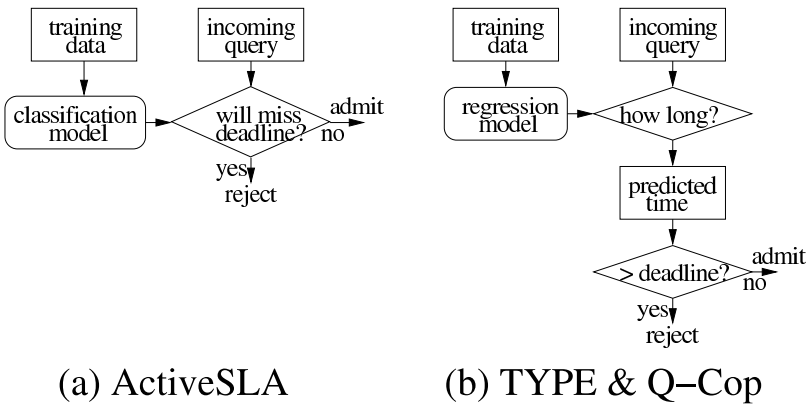


Figure 12: Comparison of machine learning models.

and available buffer size, also influence query execution time in a non-linear fashion. Furthermore, it is well known in database and queueing theories that when a system is at a borderline overload condition, a small amount of additional workload will disproportionally degrade the system performance. Such reasoning motivates us to use *non-linear* models.

Regression vs. Classification There are two reasons why we prefer a classification model over a regression model.

First, for admission control decisions, we care about the probability that a query meets its deadline or not rather than the *exact* execution time of the query. From the machine learning point of view, a direct model of classification usually outperforms a two-step approach (i.e., in step 1, regression is used to get an estimation on execution time, with an objective of minimizing the mean square error; and in step 2, this estimated value is compared with the deadline).

Second, regression models only give Est_i, i.e., the estimated execution time of q_i. This single point estimation does not contain enough information for us to make profit-aware decisions as illustrated in Section 3.2. In comparison, the classification model used in ActiveSLA provides us with the probabilities of a query meeting and missing its deadline. This information will be shown crucial for us to make SLA-based

admission control decisions.

3.4.1.3 Non-linear Classification Models in ActiveSLA

The main machine learning technique that we use for non-linear classification is "LogitBoost" algorithm [48]. We also use another algorithm called "Additive Regression" for non-linear regression for a comparison purpose. Both of them are boosting approaches where a set of weak learners (namely, models that may not have exceptionally good performance by themselves but collectively contribute to the final performance) are iteratively learned and combined in order to obtain a strong classifier with good performance. For the weak learners, we use a standard tree model (REP [48]) which partitions the parameter space in a top-down and non-linear fashion. Both of them are implemented in the well-known off-the-shelf WEKA package [57].

3.4.2 ActiveSLA Features

A key to the accuracy of a machine learning model is the features used to learn the model. In addition to the features used by TYPE and Q-Cop (Section 3.4.2.1), ActiveSLA exploits a number of additional features from query characteristics (Section 3.4.2.2) and system conditions (Section 3.4.2.3).

3.4.2.1 Query Type and Mix (TYPE, Q-Cop, ActiveSLA)

Both TYPE and Q-Cop use the number of currently running queries as the feature in their model for each query type. For a query q_i of query type i, TYPE uses Num, the total number of currently running queries in the system, as the only feature to predict query execution time of q_i. Q-Cop improves over TYPE by splitting Num into a set of features $n_1, \ldots, n_T$, which are referred to as the query mix. That is, Q-Cop takes into consideration that different query types (e.g., j) may impact the execution time of q_i in different ways (reflected by e_{ij} in the Q-Cop model).

3.4.2.2 Query Features (ActiveSLA)

We believe that even for the queries of the same query type, the parameters of a query may affect its execution time, especially when the query contains aggregations or range selections. To extract features related to query execution time, we leverage query optimization techniques, which have been studied for the past decades in both research and practice, by looking into the details of the query plan and query cost estimation from the database system. We take PostgreSQL and MySQL as examples, although the same idea applies to other databases.

In PostgreSQL, the query cost estimation depends mainly on five features, i.e., the number of sequential I/O (*seq_page*), the number of non-sequential I/O(*random_page*), the number of CPU tuple operations (*cpu_tuple*), the number of CPU index operations (*cpu_index*), and the number of CPU operator operations (*cpu_operator*). We extract these features from PostgreSQL using the light-weighted "explain" command before it executes the query, and we provide a more detailed explanation in the Appendix. Although these parameters are used mainly for PostgreSQL query optimizer to compare the relative costs among different query plans, we show the estimations of these features have strong correlation with the execution time of some query. Thus, we take these five estimations from the query optimizer as a set of features for the ActiveSLA prediction module. MySQL uses a similar "explain" command to illustrate how MySQL handles the queries and again, we provide the details in the Appendix.

3.4.2.3 Database and System Conditions (ActiveSLA)

In addition to studying queries themselves, ActiveSLA also considers the environment in which the queries will be running. More specifically, ActiveSLA monitors the following features from the database server and operating system. We choose these features as they are the most dynamic features that can affect the query execution time. Other more static features like "CPU frequency", "Memory size", "Disk

Table 3: Comparison of different models.

	Learning model	Learning method	Features used
TYPE	linear	regression	query type
Q-Cop	linear	regression	query type/mix
ActiveSLA-R	non-linear	regression	query type/mix
ActiveSLA-C	non-linear	classification	query type/mix
ActiveSLA	non-linear	classification	query type/mix query features database/system conditions

capacity" are related to a DaaS provider's assets and will be added in the future work.

Buffer cache: the fraction of pages of each table that is currently in the database buffer pool and therefore are available without accessing the disk;

System cache: the fraction of pages of each table that are currently in the operating system cache and therefore are available without accessing the disk;

Transaction isolation level: a boolean variable that indicates if the database is currently supporting Read Committed(FALSE) or Serializable(TRUE).

CPU, memory, and disk status: the current status of CPU, memory, and disk in the operating system.

A more detailed description of these features as well as the methods that we use to collect these features are provided in the Appendix. We summarize various approaches in Table 3, in terms of machine learning methods and features used.

3.4.3 The Summary of Models

As mentioned above, ActiveSLA has three unique characteristics: (1) it uses a non-linear learning method, (2) it is based on a classification model, and (3) it includes more comprehensive features. In order to distinguish the contribution of each of these characteristics, we also include two intermediate versions of ActiveSLA, named

ActiveSLA-R and ActiveSLA-C. For a fair comparison, both of these intermediate versions use only the same feature as Q-Cop does. Compared to Q-Cop, ActiveSLA-R uses a regression model that is *non-linear*, to estimate the query execution time and then makes comparison with the deadline. Compared with ActiveSLA-R, ActiveSLA-C uses a non-linear *classification* model to directly estimate whether a query will miss the deadline. ActiveSLA-R shows the benefit of non-linear over linear models while ActiveSLA-C shows the benefit of classification over regression. ActiveSLA shows the gain of using more features in the model.

3.5 Prediction Module Evaluation

In this subsection, we conduct empirical studies to evaluate the design choices that we make in the prediction module of ActiveSLA.

3.5.1 Experimental Settings

3.5.1.1 Query Sets

The experiments in this chapter are conducted by using the relations in the TPC-W benchmark [113]. Based on the TPC-W schema, we use three query sets in our experiments, i.e., TPC-W1, TPC-W2, and TPC-W3.

TPC-W1 *(browsing queries)* The TPC-W1 query set follows the same query set that is used in a main baseline approach, i.e., Q-Cop [112]. We use exactly the same query set in order to make a fair comparison. The set includes the query types in the Browsing Mix distribution in the TPC-W. The query types are extracted from the corresponding servlets, i.e., Q1(Author Search), Q2(Title Search), Q3(New Products), Q4(Related Products), Q5(Best Sellers Setup) and Q6(Best Sellers Main). This query set follows the Browsing Mix distribution specified by the benchmark. The query mix is skewed in that all the queries other than Q6 can be executed in a very short time (less than 10ms) whereas the execution time of Q6 is around 5 seconds.

TPC-W2 *(mixture of browsing and administrative queries)* For this query set, we

replace the short queries in the TPC-W1 by three other queries, which are variations of the administrative queries in the TPC-W benchmark, to evaluate the impact of longer running queries in the system.

Q6: select i_id, i_title, a_fname, a_lname from item, author, order_line where item.i_id = order_line.ol_i_id and

item.i_a_id = author.a_id and order_line.ol_o_id > (select

max(o_id)-3333 from orders) and item.i_subject = ? group by

i_id, i_title, a_fname, a_lname order by sum(ol_qty)

desc limit 50;

Q7: select c_fname, c_lname, c_email, o_sub_total from customer, orders where c_since > ? and c_id = o_c_id and

o_sub_total > ? and o_sub_total < ?;

Q8: select i_title, i_cost, i_desc, o_id, o_status, o_total, ol_id, ol_qty, ol_discount from item, order_line, orders where

i_pub_date > ? and i_id = ol_i_id and ol_o_id = o_id and

o_total > ? and o_total < ?;

Q9: select co_id, cx_type, ol_discount, avg(ol_qty),

avg(o_sub_total) from order_line, orders, cc_xacts, country

where ol_o_id = o_id and o_id = cx_o_id and cx_co_id = co_id

and ol_discount <= ? and o_total > ? and cx_type like '%?%' group by co_id, cx_type, ol_discount;

TPC-W3 *(mixture of browsing, administrative, and updating queries)* The TPC-W1 and the TPC-W2 only have read-only queries. We add the following update query Q10 to the TPC-W2 to get the TPC-W3. Q10 is extracted from servlet "Admin Confirm" in the TPC-W benchmark.

Q10: update item set i_cost = ?, i_pub_date = ? i_image = ?

where i_id = ?;

For all the query sets, we generate individual queries in the following way. When we want a query from a specific query type(template), we generate random data according to the database size and use this random data to fill in the "?" part, in order to get a real query.

3.5.1.2 Workload Generators and System Setting

We design workload generators to provide two sets of workloads. The first set of workloads follow as faithfully as possible those used by Q-Cop [112], i.e., with Poisson arrival and static arrival rates. The second set of workloads are derived from a real trace – the Web trace from the 1998 World Cup site [23], with the dynamic arrival rate scaled proportionally to fit into the experimental environment. Compared with the first set of workloads which has a *stationary* arrival rate, the second one has *non-stationary* arrival rates, and so is likely to be closer to the real DaaS scenario.

In terms of the ratio of queries of different types, for the TPC-W1 we follow that used in [112] (i.e., the same ratio as specified by the benchmark); for the TPC-W2 and the TPC-W3, we use a uniform distribution to randomly assign the query type to each query. We use PostgreSQL 9.0 as the database server. The total size of the TPC-W database is 5.2GB.

We implement, deploy, and evaluate the system on real machines. The physical machines to run the database server, the client, as well as ActiveSLA all have Intel Xeon E5620 2.4GHz Quad-Core CPU, 16GB of RAM, 1TB 7200rpm disk running Linux with kernel 2.6.18-164.15.1.el5. Machines are connected by Gigabit Ethernet switches. Note that, to further analyze the robustness of our system, we also repeat most of the experiments on lower-end machines with AMD244 1.8GHz Dual-Core, 2GB memory, and a 500GB 7200rpm disk. It turns out that the conclusions are very similar, and therefore we omit them in the chapter.

3.5.2 Result for the TPC-W1 Query Set

Since the TPC-W1 query set is the only query set used by Q-Cop, in order to make a fair comparison, we set the database settings as close as possible to those described in [112] (e.g., with the 5.2GB data fitting in 16GB memory). We also use the same Latin Hyper-cube Sampling (LHS) protocol to sample the space of query mixes. As a result, we collect around 12,000 data samples in total. To study the performance, we use 10-fold cross validation, which is a standard approach in the machine learning area.

Figure 13 shows the comparison of relative absolute error and root relative squared error among TYPE, Q-Cop and ActiveSLA-R for queries of query type Q6 (Best Sellers Main) in terms of query execution time estimation. From the figure we can see that TYPE, which only considers the number of currently running queries, has very large error; Q-Cop, which considers the query mix by using linear regression, reduces the error rate by a half from that of TYPE; ActiveSLA-R, which is a non-linear regression model, further reduces the error[2].

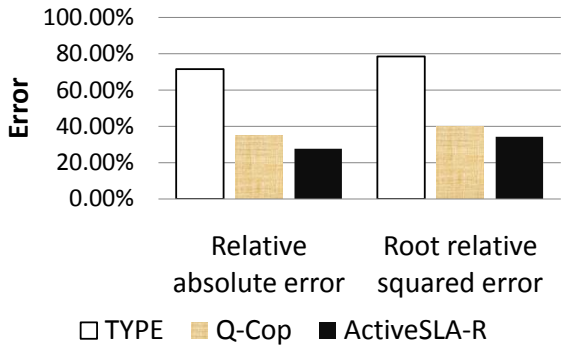


Figure 13: Regression errors for the TPC-W1 query set by different approaches.

[2]For short queries in Q1-Q5, since the execution time is usually less than 10ms, the prediction module will predict with high probability that the queries will be finished on time before their long deadlines, e.g., 30s. So in most cases ActiveSLA will admit these queries.

3.5.3 Result for the TPC-W2 Query Set

In this experiment, we reduce the memory size to 3GB so that the data set cannot fit in memory any more. We still use PostgreSQL and set the buffer pool size to 1GB and the effective cache size to 2GB. The four queries in the TPC-W2 have comparable execution times, with difference within a factor of 2. We collect around 12,000 data samples in total. In the experiments, we study the cases where the query deadlines are set to 30, 45, and 60 seconds. (As a justification, the default limit time for a PHP script that connects to a database to run is 30 seconds. The Safari browser uses a 60-second timeout.)

Because in the previous query set TPC-W1, we have already shown ActiveSLA-R can outperform Q-Cop in terms of regression error, in this experiment, we focus on the error rate of different approaches on predicting whether a query can be finished before its deadline. For this purpose, we use the following metrics.

False positive (N_{fp}): The number of queries that (1) were predicted to be meeting deadline but (2) actually miss deadline.

False negative (N_{fn}): The number of queries that (1) were predicted to be missing deadline but (2) actually meet deadline.

Finally, we assume an equal weight between over- and under-prediction and define the prediction *error* as:

$$error = \frac{N_{fp} + N_{fn}}{NT}$$

where NT is the total number of queries in the testing set. For the performance study, we again use 10-fold cross validation.

Figure 14 shows the prediction errors of different approaches with deadlines of 30, 45, and 60 seconds, respectively. Note that in addition to the performance of ActiveSLA, we also report the performance of two intermediate versions of ActiveSLA, i.e., ActiveSLA-R and ActiveSLA-C. We can make the following observations. (1) The

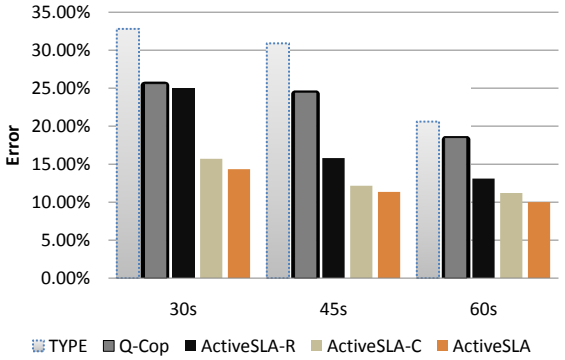


Figure 14: Prediction errors for the TPC-W2 query set by different approaches.

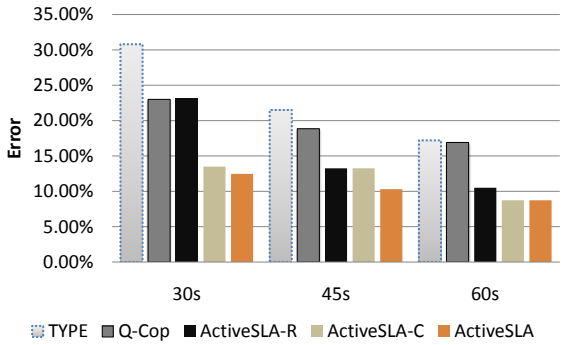


Figure 15: Prediction errors for the TPC-W3 query set by different approaches.

difference in prediction error between TYPE and Q-Cop for the TPC-W2 is not as dramatic as that for the TPC-W1. The main reason is that the execution times of the queries in the TPC-W2 are very similar among different query types, and therefore counting the total number of queries will achieve almost the same effect as counting the number of queries per type. (2) ActiveSLA-R, which uses a non-linear regression model, improves over the linear regression model used by Q-Cop. The improvement is more distinct when the deadline is longer (e.g., 60 seconds vs. 30 seconds). This result suggests that when the query execution time is long (and when the system is likely to be heavily loaded), the benefits of a non-linear model are more apparent. (3) ActiveSLA-C, by using a classification model instead of the regression model used in ActiveSLA-R, consistently outperforms ActiveSLA-R. Note that because both models use the same sets of training and testing data, this performance improvement verifies our claim that a classification model (i.e., the one-step approach) has advantages over a regression model (i.e., the two-step approach) for admission control. (4) ActiveSLA, which takes all the features into consideration and builds a non-linear classification model, has the best performance in terms of minimizing the prediction error under all the deadline settings.

3.5.4 Result for the TPC-W3 Query Set

The database and system settings for the experiments on the TPC-W3 query set are the same as those for the TPC-W2 query set. Because of the updating queries, we set up different isolation levels in the following way. In PostgreSQL, *Read Committed* is the default isolation level. We use strict two phase locking to implement the *Serializable* Isolation. In particular, we use "LOCK TABLE *tables* IN SHARE MODE" for queries with types Q6 through Q9, whereas we use "LOCK TABLE *tables* IN EXCLUSIVE MODE" for queries of type Q10. The variable *tables* in the locking commands represents the tables required by the corresponding queries. For example, for queries of type Q6, *tables* = "item, author, order_line, orders" and for queries of type Q10, *tables* = "item", as shown in Table 4.

Table 4: Query type and the locking types/tables

	Locking(Tables)
Q6	slock(item, author, order_line, orders)
Q7	slock(customer,orders)
Q8	slock(item, order_line, orders)
Q9	slock(order_line, orders, cc_xacts, country)
Q10	xlock(item)

We collect around 6,000 data samples for *Read Committed* and another 6,000 data samples for *Serializable*. In Figure 16 we show the average lock contention delay for queries of each type, to illustrate the interference among queries of different types under *Serializable* Isolation. Q7 and Q9 query types have very short contention delay (less than 2ms) because they only require shared locks on the tables they need and because for these tables, there are no other queries requiring an exclusive lock. In comparison, the situation for Q6, Q8, and Q10 query types are very different. Because Q6 and Q8 query types require shared locks on the "item" table whereas Q10 type requires exclusive locks on the same table, lock contention happens very often among these queries. Figure 15 shows the prediction errors of different approaches on the

TPC-W3 query set. As can be seen, although different isolation levels are used, most conclusions we obtain from the TPC-W2 query set are still valid for the TPC-W3 query set.

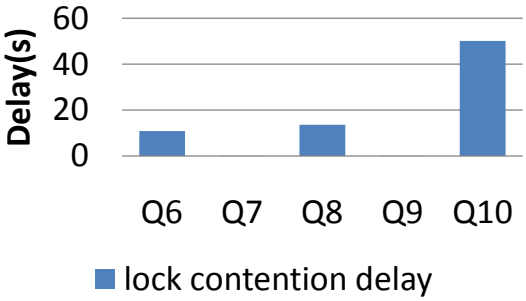


Figure 16: Lock contention delay.

3.5.5 Further Investigation

In this subsection, we provide some additional details about the prediction module of ActiveSLA, including some details on the classification model, an example feature that demonstrates non-linearity, and feature sensitivity as well as overhead analysis.

3.5.5.1 Details on the Machine Learning Model

We zoom into a segment of the learned REP decision tree model for the TPC-W3, as shown in Figure 17, to illustrate how ActiveSLA makes predictions. In the REP tree, a leaf node represents the level to which the model believes that the query will *miss* its deadline (therefore negative values indicate it is very likely the query can be finished on time). The internal branches indicate the criteria used to make the decision. As can be seen from the tree, ActiveSLA first looks at CPU_wai, which is the percentage of time that the CPU is idle because the system had an outstanding disk I/O request. If CPU_wai is lower than 7.5%, the prediction stops and returns a value of -1. If CPU_wai is greater than 7.5%, ActiveSLA next looks at the *type* of the query. If the query is of type Q10, then based on the system isolation level (serializable or not), the model returns a value of 0.33 (very likely to miss deadline)

or -2 (very unlikely). This segment of REP decision tree captures the fact that a query of type Q10 is an update query and so when the isolation level is read commit, the query almost never misses its deadline. On the other hand, if the isolation level is serializable, there will be a lock contention delay and so the query is very likely to miss its deadline.

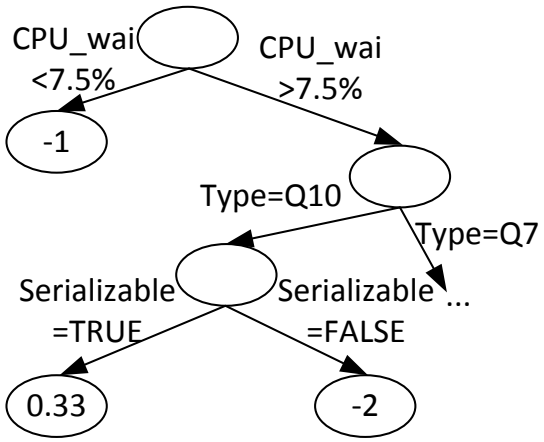


Figure 17: Part of a REP tree learned by ActiveSLA.

3.5.5.2 A Feature That Demonstrates Non-linearity

We use one particular feature to demonstrate the effectiveness and non-linearity of certain database features. The feature that we use is *the estimated number of pages of sequential scan according to the query plan*, i.e., *seq_page*. Figure 18 shows the scatter-plots of the number of pages of sequential scan estimated by the query plan (x-axis) vs. the query execution time (y-axis) for the queries of different types in the TPC-W2 query set.

From the figure we can obtain several observations. First, this particular feature has predictive power for Q7 and Q9 query types, which can be seen from the correlation between x-value and y-value in the corresponding sub-figures. The correlation coefficients are 0.62 and 0.52, respectively. Second, for Q6 type, this feature has almost no predictive power with a correlation coefficient as 0.20. Third, for Q8 type, when the number of pages of sequential scan is low (and therefore the query

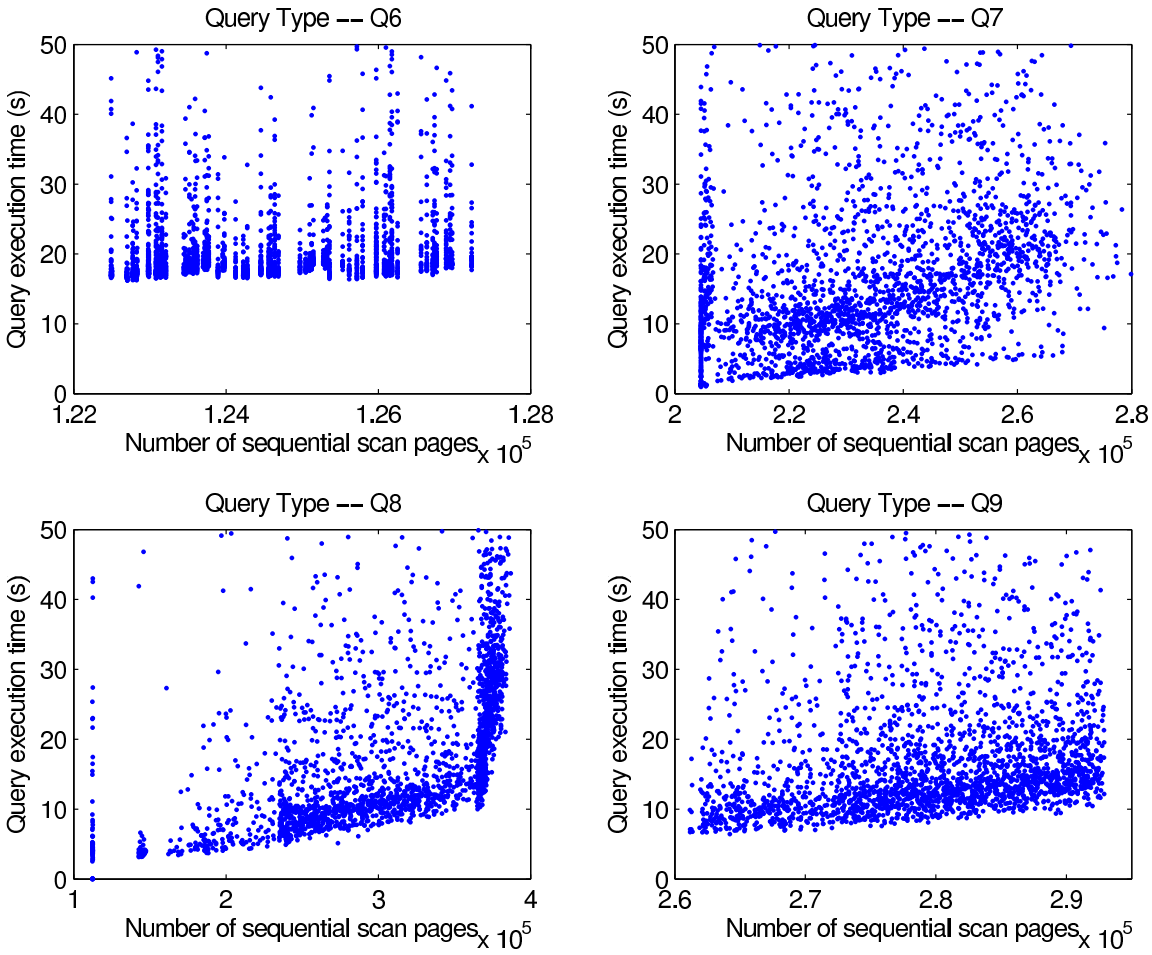


Figure 18: Influence of the number of seq_page scan on query execution time for different query types.

execution time is short), the feature has obvious predictive power with correlation coefficient as 0.78; however, when the number of pages of sequential scan grows larger than 350K, its predictive power disappears with a correlation coefficient as 0.17. This result illustrates the importance of the *non-linear* models used in ActiveSLA.

3.5.5.3 Sensitivity Analysis of Features

ActiveSLA uses three sets of features, namely, *query type/mix*, *query features*, and *database/system conditions*. To study the effectiveness of each feature set, we conduct the following experiment of sensitivity analysis.

For the query sets TPC-W2 and TPC-W3, we compare the performance of (1) ActiveSLA with all three feature sets and (2) ActiveSLA with one feature set removed. Basically, we want to see the performance of ActiveSLA when a particular set of features is not available. The results for the TPC-W2 and the TPC-W3 are similar and summarized as below. (1) Removing any set of features will increase the prediction error, albeit to different degrees. (2) The importance of the feature sets, from the most important to the least important is: *query type/mix* > *query features* > *database/system conditions*, as shown in Figure 19.

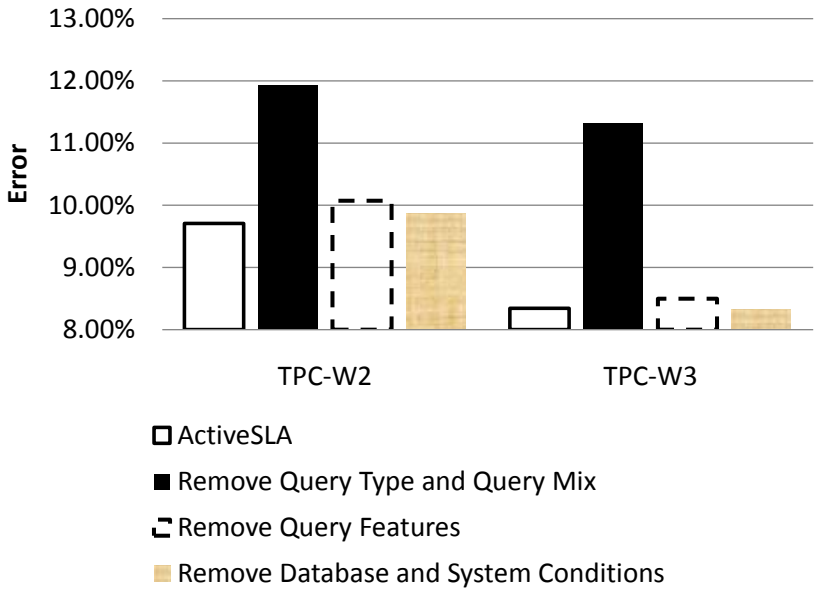


Figure 19: Sensitivity Analysis of Features when the deadline is 60s.

3.5.5.4 Training and Evaluation Overhead

The classifier is first trained and then evaluated. In the training stage, it takes approximately 72ms to build an initial model by using 12,000 samples. The model is rebuilt every time when there are another 100 new samples, which are sent back from the admitted queries. In the evaluation stage, it takes approximately 8ms to predict the probability that a query meets the deadline when a single query arrives at the prediction module. The prediction overhead is negligible compared with the mean query execution time and the deadline (i.e., 30s, 45s, or 60s).

3.6 Decision Module

The decision module of ActiveSLA is responsible for making the final decision on whether a new query should be admitted. In this subsection, we first describe the SLAs that we assume. Next, we describe under a general SLA, how to make profit-oriented admission control decisions by using the standard decision theory. Then we show that under a commonly used SLA form, namely step-function SLA, the decisions can be made in a more efficient way. Finally, we show how our decision module takes into account the interference among clients (queries), who are competing with each other for the shared system resources.

3.6.1 Service-level Agreement (SLA)

As discussed before, a service-level agreement (SLA) is a contract between a service provider and a client that determines the promised service performance and the corresponding gains (or penalties). While SLAs in general depend on certain chosen criteria, we focus on query latency, i.e., query execution time in this chapter. More specifically, we assume that there is an associated SLA for each query q, which determines the gain and the penalty that will be obtained by the service provider under different query execution times for q. An example of such an SLA is shown

in Figure 20(a)(upper sub figure), where the revenue (gain or penalty) is a function $s(t)$ over the query execution time t assuming the query is admitted. In addition, not explicitly shown in the figure is that if the query is rejected up-front, the service provider has to pay a penalty of -r.

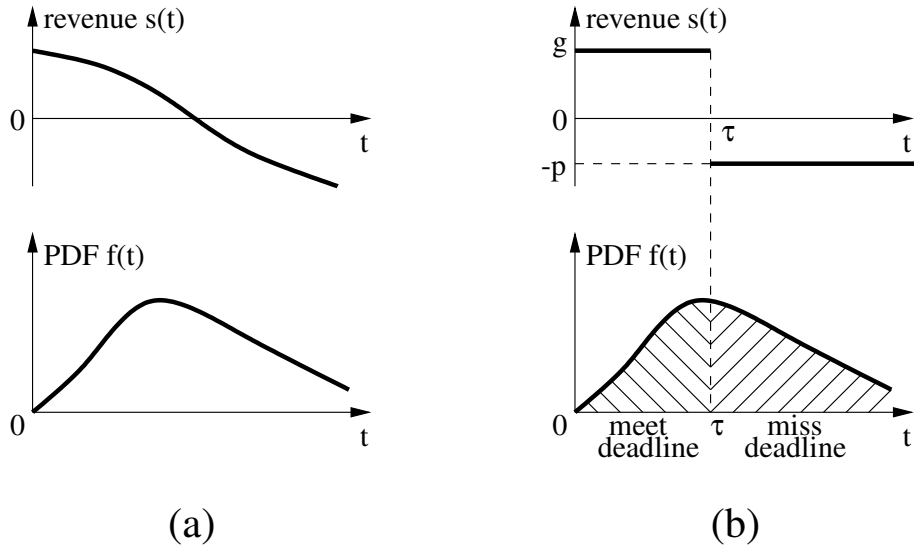


Figure 20: SLA-based admission control decisions for (a) general SLAs and (b) step-function SLAs.

Note that a more commonly used SLA is based on the quantile of the response time of queries from a single client. If this is preferred, there exist techniques (e.g., [52]) that directly map quantile-based SLAs to per-query SLAs. However, based on our extensive interactions with numerous business organizations that provide services to real clients, they desire to be able to manage SLAs at the finest granularity level (i.e., per query) with multiple levels of delivery defined in the SLAs (i.e., piecewise linear functions [36]). The observation is that currently majority of the service providers only give availability SLAs to their clients represented in the quantile form but not other types of SLAs such as latency, throughput, etc. Also, lack of formal models and tooling to enable finer granularity level SLA management is a major inhibitor for businesses to adopt various types of SLAs and also varying levels. Our research aims at advancing the state-of-the art in that area and helping service providers by

working with them and this chapter is a part of that effort.

3.6.2 The Admission Decisions

The main goal of a DaaS provider is to maximize profit by satisfying its client service-level agreements (SLAs). Therefore, we use profit as the final objective in our admission control strategies.

If we assume that the probability density function (PDF) $f(t)$ for the execution time of q is available to the service provider, as shown in Figure 20(a)(lower sub figure), we can compute the expected revenue $E\left[revenue(q)\right]$ for admitting query q as

$$E\left[revenue(q)\right] = \int_{t=0}^{\infty} s(t) \cdot f(t) dt.$$

Then by following the standard decision theory, the admission control decision that maximizes the expected profit for query q should be

$$Decision = \begin{cases} Admit & \text{if } E\left[revenue(q)\right] > -r \\ Reject & \text{otherwise.} \end{cases}$$

The SLA function $s(t)$ usually can be directly obtained from the service contract. However, there are still several other challenges. (1) It is very difficult to obtain the detailed query performance information for query q, i.e., the PDF in Figure 20(a), before the incoming query q is even executed. (2) A different incoming query q may have a different SLA as well as time-varying query performance information, given that the status of database management system and the operating system are constantly changing. (3) Because we assume that queries come in an online fashion and there is no prior knowledge about the future, we also should reserve resources for future "more profitable" queries. In the following, we address these challenges by extending the standard decision theory.

3.6.3 Single Query Decision

Although the SLA on query execution time may take different forms, a step function is commonly used in real contracts because it is easy to describe in natural language. We show such a step-function SLA in Figure 20(b) and Table 5. That is, for a given query q, if the query is admitted and its query execution is finished before the deadline τ, the service provider obtains a gain of g; else if the query misses the deadline τ, the service provider pays a penalty of -p. Otherwise the service provider may reject the query upfront and pay a penalty of -r.

Table 5: Step-function SLA: outcomes and revenues.

	Meet Deadline	Miss Deadline
Admit	g	$-p$
Reject	$-r$	$-r$

Under the step-function SLA, we can simplify the expected revenue for admitting q as (also illustrated in Figure 20(b))

$$\begin{aligned} E\left[revenue(q)\right] &= \int_{t=0}^{\tau} g \cdot f(t)dt + \int_{t=\tau}^{\infty} (-p) \cdot f(t)dt \\ &= g \cdot \int_{t=0}^{\tau} f(t)dt - p \cdot \int_{t=\tau}^{\infty} f(t)dt \end{aligned}$$

The above result reveals that to compute the expected revenue under the step-function SLA, we only need the *area* under $f(t)$ before τ and that after τ, which are actually the probabilities of meeting and missing the deadline. Such probabilities, as have been shown in the previous sections, are provided by the prediction module of ActiveSLA in a real-time fashion. If the probability that the query meets its deadline is c, we can easily see that $\int_{t=0}^{\tau} f(t)dt = c$ and $\int_{t=\tau}^{\infty} f(t)dt = 1 - c$. As a result, we have $E\left[revenue(q)\right] = g \cdot c - p \cdot (1-c)$. Thus the exact PDF for the execution time of q is not necessary for admission decisions anymore. The admission control decision

is made as

$$Decision = \begin{cases} Admit & \text{if } g \cdot c - p \cdot (1 - c) > -r \\ Reject & \text{otherwise.} \end{cases}$$

3.6.4 Multiple Query Decision

The admission decision made in Section 3.6.3 is based on the expected revenue for admitting a single query q. If we want to make the most profit from multiple queries, we have to take into consideration at least two additional hidden costs when we decide whether or not to admit q. (1) Admitting q into the database server may slow down the execution of other queries that are currently running in the server, since query q will consume system resources. Therefore, admitting q will potentially cause other running queries to miss their deadlines, which they were able to meet. This will reduce the total revenue of a DaaS provider. (2) Admitting q will consume system resources and change the system status. This may result in the rejection of the next query, which may otherwise be admitted and bring in a higher revenue. The two additional hidden costs are closely related to the concept of *opportunity cost* in economics [63, 106]. We denote the opportunity cost as o, and we revise the decision module in ActiveSLA according to Table 6, in order to take the opportunity cost into account.

Table 6: Step-function SLA: outcomes and revenues, with opportunity cost.

	Meet Deadline	Miss Deadline
Admit	$g - o$	$-p - o$
Reject	$-r$	$-r$

According to this new table, when $o > 0$, the admission control is relatively more aggressive in rejecting new queries, in order to protect the currently running queries and to reserve system resources for later queries with potentially higher revenues.

With such an extra cost term, the admission control decision becomes

$$Decision = \begin{cases} Admit & \text{if } g \cdot c - p \cdot (1 - c) - o > -r \\ Reject & \text{otherwise.} \end{cases}$$

In practice, the value o for opportunity cost can be either determined by the service provider (e.g., derived from certain business considerations) or learned through the ActiveSLA feedback channel over time.

3.7 Decision Module Evaluation

We use the workload generators to generate different workload traces in an offline fashion. Then we test the workload traces in the real-time system to present the effectiveness of our system. We also compare ActiveSLA's performance with the previous work, Q-Cop. We report the results based on the TPC-W2 query set in this subsection and skip those for other query sets because the results are similar. For each test, unless stated otherwise, we repeat 5 times (with traces generated from different random seeds) and report the average performance.

3.7.1 Result with Stationary Workload

In this experiment, we use a stationary workload with arrival rates ranging from 0.01 request/second to 0.1 request/second. Each test runs 1 hour. We use the numbers for the utility function of q_1 as shown in Figure 10 for the SLA. We can obtain the following observations from the results as shown in Figure 21. (1) When the arrival rate is less than 0.03 request/second, because the system is underloaded, both Q-Cop and ActiveSLA admit most of the queries. (2) When the arrival rate goes beyond 0.03 request/second, load shedding starts to take place more frequently. However, under all the arrival rates, ActiveSLA admits between 10% to 15% more queries than Q-Cop, and among the admitted queries, the number of queries that miss their deadline is comparable between ActiveSLA and Q-Cop. These results show that ActiveSLA

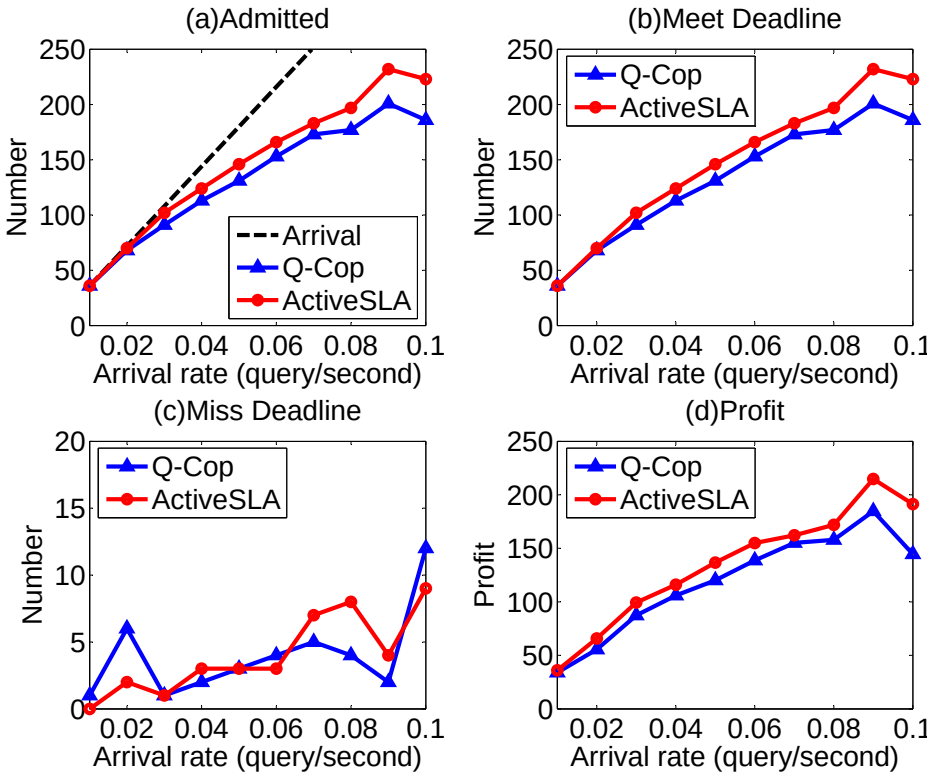


Figure 21: Result with stationary workload: number of queries that (a) are admitted, (b) meet deadline after admitted, (c) miss deadline after admitted, and (d) total profit.

makes more reasonable admission control decisions. (3) The advantage of ActiveSLA's better decisions is reflected in its higher SLA profits compared to Q-Cop.

3.7.2 Result with Non-stationary Workload

For the non-stationary workload, i.e., the World Cup trace from the 1998 World Cup site [23] from 15:00pm to 22:21pm, we study two experiments. In the first experiment, we assume all the queries have the same SLA. In the second experiment, we use two different SLAs to show how ActiveSLA makes profit-oriented decisions. More specifically, in the second experiment we assume half of the queries have one SLA and the other half have another SLA, in order to demonstrate that ActiveSLA is able to provide profit-oriented service differentiation.

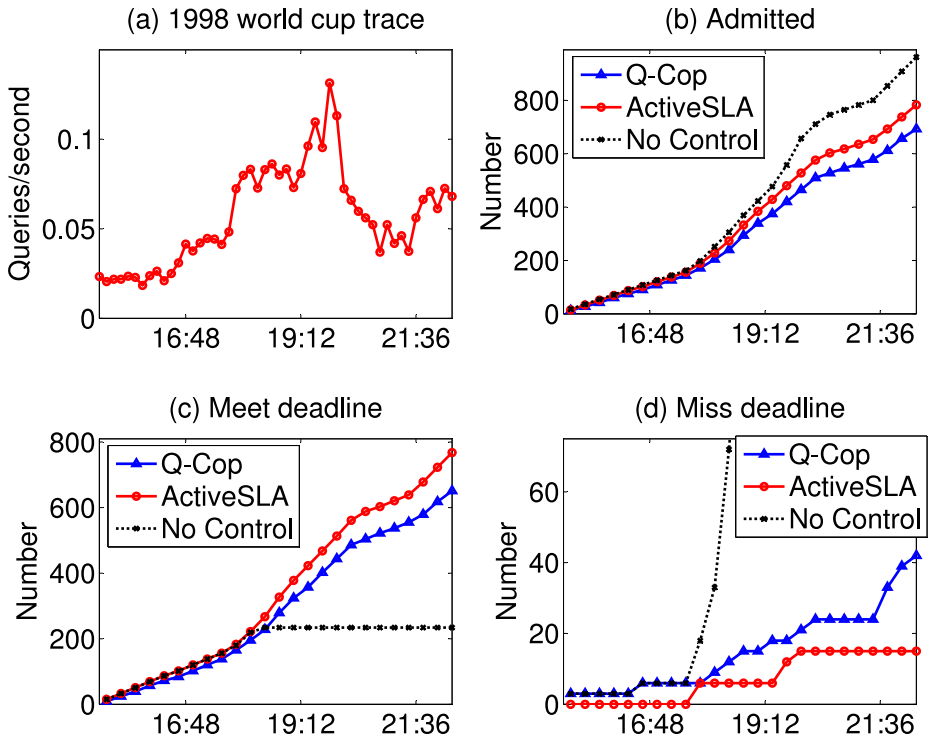


Figure 22: Dynamic workload: over time, (a) the rate of arrived queries, the cumulative numbers of (b) admitted queries, (c) queries that are admitted and meet their deadlines, and (d) queries that are admitted but miss their deadlines.

3.7.2.1 Profit-oriented decisions

We start by again using the SLA described by the first utility function in Figure 10, i.e., $g = 1$, $p = 1$, $r = 0.1$, and $o = 0$. Figure 22 shows the experimental results. From these figures we can see that over time, ActiveSLA admits more queries and fewer queries admitted by ActiveSLA miss their deadlines than those admitted by Q-Cop. In addition, in Table 7, we report the aggregated results over the whole period of the World Cup event. From the table we can see that during this event, ActiveSLA admits 10% more queries than Q-Cop does and compared to Q-Cop, fewer queries admitted by ActiveSLA miss their deadline. Overall, ActiveSLA achieves 20% more profit than Q-Cop.

Table 7: Comparison of SLA profit (with the total number of queries being 963).

	Admitted	Meet Deadline	Miss Deadline	Profit
$g = 1$(gain),$p = 1$(penalty),$r = 0.1$(reject penalty)				
Q-Cop	693	651	42	582
ActiveSLA	783	768	15	735
$g = 1$(gain),$p = 2$(penalty),$r = 0.1$(reject penalty)				
Q-Cop	693	651	42	540
ActiveSLA	744	744	0	722.1

Next, we switch to the SLA described by the second utility function in Figure 10, i.e., $g = 1$, $p = 2$, $r = 0.1$, and $o = 0$. The aggregated results are also shown in Table 7. The fact that ActiveSLA outperforms Q-Cop remains valid. However, because of the higher penalty of missing a deadline, ActiveSLA becomes more conservative in that it admits fewer queries and makes less profit. In this case, the conservative admission control by ActiveSLA is well justified—among the queries admitted into the system by ActiveSLA, none of them misses their deadlines and so the high penalty of missing deadlines is avoided.

3.7.2.2 Profit-oriented service differentiation

In this experiment, we use the same world cup trace as used in the previous part. However, this time we pick half of the queries and assign them SLAs with higher gain. More specifically, we give each query a sequence number. For the queries with odd sequence number (which we refer to as Gold queries) we increase their g values in the SLA to 1.5 while for the queries with even sequence number (which we refer to as Silver queries), we keep their original g values of 1.

Table 8: Comparison of SLA profit with the total number of queries being 963(Gold/Silver).

	Admitted (Gold/Silver)	Meet Deadline (Gold/Silver)	Miss Deadline (Gold/Silver)	Profit
$g = 1.5/1$(gain),$p = 1$(penalty),$r = 0.1$(reject penalty)				
Q-Cop	(365/328)	(341/310)	(24/18)	752.5
ActiveSLA ($o = 0$)	(420/384)	(396/375)	(24/9)	920.1
ActiveSLA ($o = 0.2$)	(438/348)	(432/347)	(6/1)	970.3

In addition, we study two scenarios, one scenario where there is no opportunity cost (i.e., $o = 0$) and the other with opportunity cost (with $o = 0.2$), in the decision module of ActiveSLA. Table 8 reports the results for these scenarios, where we separate the performance of Gold queries(G) and that of Silver ones(S). From the results we can make the following observations. (1) In both scenarios, ActiveSLA admits more queries than Q-Cop and makes more profit. (2) Because the potential revenue gain for Gold queries is higher, ActiveSLA is more aggressively in admitting Gold queries than in admitting Silver ones (and results in more Gold queries missing their deadlines). But such aggressive decisions are rewarded by the higher profit. (3) When the opportunity cost is taken into consideration in the decision module, ActiveSLA admits fewer Silver queries and at the same time, admits more Gold queries (compared to the scenario when there is no opportunity cost). In addition, because of

the protection effect of the opportunity cost, fewer Gold queries miss their deadlines once they are admitted into the system.

It is worth noting that we have only demonstrated that the opportunity cost o impacts the profit the way we expect, and we have not provided a systematic way to set the value of o. In real implementations, o can be either a tuning factor that the service provider needs to set, or it can be automatically adjusted at the runtime through a feedback loop. This is left for future work.

3.8 Summary

In this chapter, ActiveSLA is proposed to enhance a classical admission control approach by leveraging risk assessment based on decision theory to achieve the most profitable service-level compliance for a DaaS provider. The strengths and weaknesses of ActiveSLA compared with a classical admission control approach are summarized as below:

3.8.1 Strengths

Under a step-wise SLA, ActiveSLA is able to derive not only the most probable category(i.e., a step in a step-wise SLA) that the current predicted query execution time belongs to but also the probability of each category that the current predicted query execution time may belong to. However, since a classical admission control approach builds a regression model and returns single point estimation of query execution time, it can only derive the most probable category that the current predicted query execution time belongs to. ActiveSLA also takes into consideration query features as well as database-specific and system-level metrics, which further help to improve the prediction accuracy.

Thus, ActiveSLA will show its strength by obtaining the probability for each category that the current predicted query execution time may belong to under a step-wise SLA because it adopts a classification model. It will also show its strength

in prediction accuracy when query features as well as database-specific and system-level metrics are available.

3.8.2 Weaknesses and future work

ActiveSLA will show its critical limitation under a non-step-wise SLA because a classification model will be very difficult to apply to a non-step-wise SLA. And, since ActiveSLA relies on the query optimizer to obtain query features such as the number of sequential I/O, ActiveSLA will show its serious limitation when the query optimizer returns very inaccurate query features, e.g., due to the incorrect statistics and cardinality estimates of a query execution plan. In the future, we plan to repair the inaccuracy in real-time (e.g., similar to [109, 86]) to make better predictions.

Moreover, since ActiveSLA depends on the opportunity cost o to manage multiple query decisions, an inappropriate value of o in system settings will impose restrictions on its application. The determination of the value of o is an interesting problem by itself as in our future plans.

Finally, the increasing complexity of an admission control-based approach for database management systems in the Cloud presents novel technical challenges that demand enhancement to ActiveSLA itself. For example, in more and more DaaS deployments, different replication levels are provided to overcome the failures that may occur to commodity hardware. In the future, we plan to extend the prediction module of ActiveSLA by including the level of replication as one of the system variables used in non-linear classification. We are also planning to extend our ActiveSLA to deal with different types of database systems to manage data and serve queries, e.g., NoSQL databases.

CHAPTER IV

STATISTICAL IDENTIFICATION OF CRITICAL RESOURCES

In the previous chapter, a classical admission control approach is enhanced to help DaaS providers achieve the most profitable service-level compliance. A job (a query for a DaaS provider) is admitted into the system if accepting this job is expected to make more profit than rejecting it according to decision theory. Besides admission control, resource allocation is also popularly used to help a Cloud service provider make the most profit as shown in Figure 2 in Chapter 1. However, the most important prerequisite for resource allocation is to identify what kind of resource to allocate or what is the critical resource that affects service-level compliance.

In this chapter, a critical resource identification framework called vPerfGuard based on statistical machine learning is proposed to address novel technical challenges, specifically high scalability and adaptability requirements, to effectively identify critical resources for a complex distributed application in the Cloud.

4.1 Background

When an increasing number of jobs are admitted into a system and the service-level compliance becomes worse, more resources could be purchased from IaaS providers to improve the situation. Although IaaS providers offer convenience for dynamic resource allocation by offering different type of resources such as computing power and elastic storage, they charge different amounts for the usage of different types of resources. For example, the "Extra Large" Amazon EC2 High-CPU On-Demand Instances, High-Memory On-Demand Instances and High-I/O On-Demand Instances

cost $0.660, $0.450 and $3.100 per Hour, respectively, for Linux/UNIX Usage in Amazon's US East data center [1]. Therefore, the critical resource that affects the application's performance needs to be carefully identified before resource allocation. Otherwise, blindly adding resources will not only be useless to improve the application's performance but also incurs a large bill of costs.

However, identifying the critical resource that affects service-level compliance is complicated due to high scalability and adaptability requirements for a complex distributed application in a Cloud environment. Although some researchers target their efforts at identifying the critical resources, their approaches heavily rely on human expert experience and domain knowledge, which are not sufficient to deal with the increasing complexity. In this chapter, we propose vPerfGuard, which enhances a classical control-based approach by leveraging statistical machine learning to automatically and adaptively identify critical resources for a complex distributed application in a Cloud environment.

The rest of this chapter is organized as follows. Section 4.2 and Section 4.3 give the problem definition and the solution overview, respectively. The design of vPerfGuard is described in Section 4.4. Section 4.5 introduces testbed setup. The results of experimental studies are presented in Sections 4.6 and 4.7. Section 4.8 describes the visualization of the results before Section 4.9 summarizes the chapter.

4.2 Problem definition

Assume that we are a platform as a service (PaaS) provider. We purchase resources from IaaS providers to service our customer's applications. A customer deploys his Slashdot-like news aggregation site onto our PaaS environment. We refer to this distributed web-based application as *vSlashdot* to distinguish it from the real Slashdot

[1] http://aws.amazon.com/ec2/pricing/

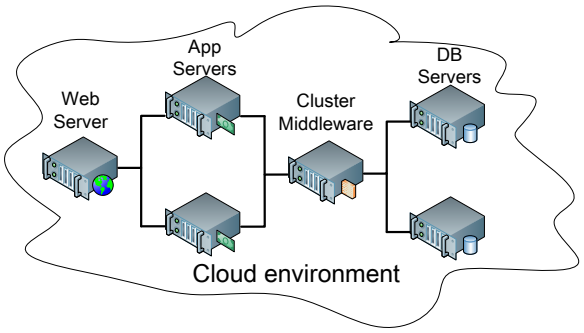


Figure 23: An example deployment for vSlashdot

service. For ease and flexibility of development and deployment, web-based applications are typically architected as a collection of cooperating components spread over multiple logical tiers, with each tier having a single/specific purpose (*n-tier/multi-tier* architecture design). An example multi-tier architecture of the vSlashdot application is shown in Figure 23 – note that each tier may host one or more servers. *Web servers* in the presentation tier interact with *App servers* in the application tier on behalf of clients, which are sending requests to the application. The application tier is where the business logic/rules live, and it in turn interacts with the *DB servers* in the database tier to obtain/store persistent data. Clustering middleware is often used in front of the database tier for better scalability, availability and reliability.

After the vSlashdot service goes online, our customer notices that the performance of the website degrades over time – its throughput decreases and the response times increase. Our customer immediately complains to us about the performance issue and the onest is now on us to quickly *identify the root cause* of the performance degradation by pinpointing the resource bottlenecks in the VMs/hosts and the key components of the application affecting/limiting its performance. After that, we could apply dynamic resource allocation based on the information of critical resource.

A classical critical resource identification approach either relies on human experts with deep technical knowledge to identify performance bottlenecks [64], with the help of performance monitoring tools and system logs, or follows standard procedures in

performance troubleshooting "cookbooks" [77] for problem localization and root cause analysis.

However, a classical critical resource identification approach can no longer meet the high scalability and adaptability demand in a highly-dynamic, large-scale, complex Cloud environment. There are several challenges: (1) a classical critical resource identification approach has highly variable resolution times (from minutes to weeks); (2) a classical critical resource identification approach is not easily *scalable* to analyzing the behavior of many hosts and VMs in consolidated environments and many heterogeneous and distributed applications; (3) performance "cookbooks" are not *adaptive* as they only provide guidelines for problems that were seen before, whereas a dynamic Cloud environment is likely to see emergent behavior or new interactions. For example, the performance of one application may suffer due to demand spikes in other applications (i.e., *noisy neighbors*) sharing the same physical infrastructure.

4.3 Solution approach overview

In order to address the above challenges, we propose a framework, called *vPerfGuard*[2]. vPerfGuard demonstrates that a classical control-based approach enhanced by statistical machine learning identifies critical resources automatically and adaptively. More specifically, vPerfGuard automatically builds a performance model for an application using the system metrics that are most predictive of the application's performance. It then adaptively updates the model when it detects changes in the performance and the potential shifts in the predictive metrics that may accompany such a change. More concretely, the vPerfGuard architecture, as shown in Figure 24, consists of three modules - a sensor module, a model building module, and a model updating module.

Whenever a performance degradation is observed and a critical resource identification request is received, vPerfGuard presents the top predictive system metrics in

[2]vPerfGuard stands for: virtual Performance Guard.

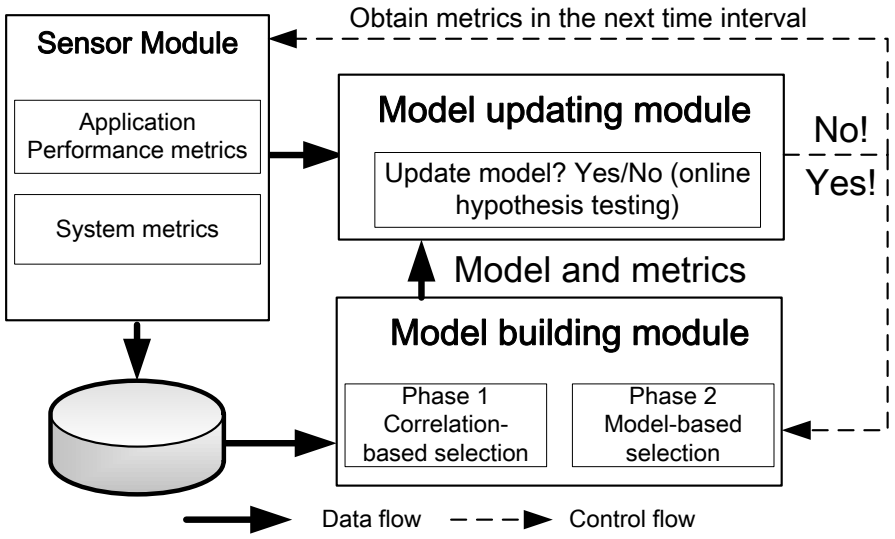


Figure 24: vPerfGuard framework

the current performance model to a Cloud service provider. These metrics can provide hints to a Cloud service provider regarding the potential causes for the observed performance problem, including the critical component within a complex, distributed application as well as the suspected critical resource in the associated host or VM. A Cloud service provider can then use this information to determine the real root cause and take remediation steps such as dynamic resource allocation. Moreover, the ability of these models to predict application's performance using system metrics can enable the development of performance control systems that further automate the process of remediation. The last goal is the focus of the next chapter and will not be discussed further in this chapter.

Through both theoretical reasoning and experimental validation, vPerfGuard achieves *automated*, *scalable*, and *adaptive* critical resource identification by enhancing a classical control-based approach with statistical machine learning in consolidated Cloud environments.

1. We leverage appropriate statistical learning techniques to construct performance models that capture the relationship between application's performance and

system resources. We implement the solution in the model building module of vPerfGuard. The statistical learning techniques (1) filter thousands of system metrics and select those that are most strongly correlated with observed application's performance, eliminating a large number of irrelevant metrics, and (2) further reduce redundancy in the selected metrics and build a performance model using a small set of metrics that give the best prediction accuracy. The automatic model generation process successfully overcomes the *scalability* challenge.

2. We leverage appropriate statistical hypothesis tests to detect the need to update the performance model when it no longer accurately captures the relationship between performance and system resources. We implement the solution in the model updating module of vPerfGuard. The statistical hypothesis tests (1) detect the change-point due to variations in workloads (such as demand spikes) or system conditions (such as resource contention), and (2) trigger the model building module to update the set of predictive metrics and rebuild the model at runtime. The automatic model updating process effectively overcomes the *adaptivity* challenge.

4.4 System design

In this section, we introduce the design of the three modules - the sensor module, the model building module, and the model updating module in vPerfGuard.

4.4.1 Sensor module

The objective of this module is to continuously collect system metrics and application's performance metrics. More specifically, it collects two categories of system metrics - *VM metrics* from the operating systems within individual VMs and *host*

metrics from the physical hosts running the hypervisors. In our experimental evaluations, the sensor can collect thousands of system metrics, which we refer to as *raw metrics*. It also collects the application's performance metrics of interest, e.g., throughput, response times, etc. Note that, although workload metrics such as offered load and transaction mix may also be helpful [108], and our framework does not preclude these metrics, we do not require their inclusion because: (1) we prefer vPerfGuard to be application-agnostic to free Cloud service providers from needing detailed knowledge about the inner operations of their customers' applications, and (2) results from our experimental studies show that they are filtered out and do not appear in any of our final performance models.

4.4.2 Model building module

The objective of this module is to automatically utilize the thousands of *raw metrics* from the sensor module to derive a performance model that captures the relationship between application's performance and system resources.

However, a performance model that is built using all the raw metrics can be computationally expensive to construct and can lead to model over-fitting. First, since the size of the search space with thousands of *raw metrics* is huge, machine learning algorithms operate slowly. Second, many raw metrics are irrelevant or redundant, e.g., a VM's CPU utilization observed from its host is closely related to the CPU utilization observed from within the VM itself. Such dependencies among metrics increase the amount of redundant information in the model and can degrade model quality.

This necessitates the selection of a small number of highly predictive metrics. After removing as many of the irrelevant and redundant metrics as possible, the model accuracy can be improved in some cases while the model can be more easily interpreted in other cases. We leverage two categories of algorithms for feature selection [58] to

our metric selection: *filters* [67] evaluate features according to heuristics based on general characteristics of the data while *wrappers* [67] use the learning algorithm itself to evaluate the usefulness of features.

We achieve the objectives of metric selection and model building using a two-phase algorithm, which is a combination of filters and wrappers: first (in phase 1) selecting a small number of *candidate metrics* that are most strongly correlated with the application's performance from among the raw system metrics, and then (in phase 2) identifying even fewer *predictor metrics* that can give the best prediction accuracy for a specific model from among the candidate metrics.

4.4.2.1 Phase 1: Correlation-based selection

In phase 1 (see Algorithm 1), we aggressively reduce the number of raw metrics considered by filtering out the raw metrics that are not highly correlated with the observed application's performance. We denote the application's performance metric (e.g., mean response time) as $perf$, and the time series of the $perf$ metric ending at time interval t as a vector $\overrightarrow{perf(t)} = [perf(t), perf(t-1), ...]$. We denote a raw system metric (e.g., CPU consumption of a VM) as m, and the set of all the raw metrics as M. We then denote the time series of each metric ending at time interval t as a vector $\overrightarrow{m(t)} = [m(t), m(t-1), ...]$. For each metric $m \in M$, we use r_m to denote the absolute value of the correlation coefficient between $perf$ and m, and p_m to denote the associated p-value for testing the hypothesis of no correlation. Each p-value is the probability of getting a correlation coefficient as large as the observed value by random chance, when the true correlation is zero. If the p-value is small, say less than 0.05, then the observed correlation is significant.

To limit the number of candidate metrics for our model, we select N_{can} top metrics as the candidate metrics for the phase 2 algorithm from all the N_{raw} raw metrics based on the absolute correlation coefficient value and the p-value. We also sort N_{can} metrics

Algorithm 1: Phase 1

1 **procedure** Metrics selection by correlation coefficient
2 Input: performance metric $perf$ and a set of raw metrics M;
3 Output: a set of candidate metrics $M_{can} \subset M$;
4 Tunable Parameter: number of candidate metrics N_{can};
5 $\forall m \in M$, $r_m = |\text{corrcoef}(\overrightarrow{perf}, \overrightarrow{m})|$, $p_m = \text{p-value}(\overrightarrow{perf}, \overrightarrow{m})$;
6 Select top N_{can} metrics with best r_m and p_m;
7 Sort N_{can} metrics in descending r_m and ascending p_m;
8 Return the set of N_{can} metrics, denoted as M_{can}.

for visualization purpose. The time complexity is $O(N_{raw}) + O(N_{can} \log N_{can})$ when the BFPRT algorithm [8] is used for selection and the quicksort algorithm is used for sorting, respectively. N_{can} is a configurable parameter for managing the tradeoff between better model accuracy and lower overhead in the second phase.

4.4.2.2 Phase 2: Model-based selection

In phase 2 (see Algorithm 2), we explore the combination of the candidate metrics generated in phase 1, and choose a combination that gives the best prediction accuracy measured by the average R^2 (coefficient of determination) value [44] of the performance model using a 10-fold cross validation [100]. We evaluate and compare the predictive capability of the following four specific types of performance models — linear regression model [45], k-nearest neighbor (k-NN) [45], regression tree [45], and boosting approach [45].

Although the metric subset space has been reduced from $2^{N_{raw}}$ to $2^{N_{can}}$ after phase 1, the exploration process is still clearly prohibitive for all but a small number of metrics. We use a heuristic, hill climbing [101] search strategy, i.e., given a set of selected metrics, we choose the additional metric from the remaining set that can give the best improvement in the R^2 value. The algorithm ends when the improvement is smaller than a given threshold. For N_{can} candidate metrics, the computation complexity of the phase 2 model-based selection is $O(N_{pred} * N_{can})$, where N_{pred} is the total number of metrics in M_{pred}.

Algorithm 2: Phase 2

procedure Metrics selection by a specific model
Input: a performance metric $perf$ and M_{can} from phase 1;
Output: a set of predictor metrics $M_{pred} \subset M_{can}$ and the associated model $F(M_{pred})$ with learned parameter values;
Tunable Parameter: a type of model F, R^2_{inc} for the minimum incremental R^2 improvement;
$selected = \emptyset$, $left = M_{can}$, $R^2_{old} = 0$, $R^2_{best} = 0$;
while *true* **do**
  for $m \in left$ **do**
    $metrics = selected \cup \{m\}$;
    Use $\overrightarrow{perf}$ and all $\overrightarrow{m}$ in $metrics$, obtain R^2_{new} following 10-fold cross validation;
    if $R^2_{new} > R^2_{best}$ **then**
      $R^2_{best} = R^2_{new}$;
    end
  end
  if $R^2_{best} - R^2_{old} > R^2_{inc}$ **then**
    move m from $left$ to $selected$;
    $R^2_{old} = R^2_{best}$;
  else
    break;
  end
end
Build the final model $F(selected)$ using all the samples;
Return $M_{pred} = selected$ and the model $F(selected)$.

Note that Hall [58] proposes a method to select a subset of metrics based on a heuristic "merit" of the subset. The motivation is that phase 1 (i.e., filters [67]) may pick many metrics, which individually have high correlation with the output metric, but that when combined together in a model do not provide much additional useful information. We implement his method and compare it with our two-phase algorithm. We find that (1) his method has comparable overhead with ours; (2) our phase 2 algorithm can also overcome the limitation of phase 1; (3) the final metrics and models are very similar if the final number of predictor metrics (N_{pred}) is a small number.

4.4.3 Model updating module

The objective of this module is to automatically detect the change-point when the performance model derived from the model building module no longer accurately captures the relationship between application's performance and system resources.

In a highly-dynamic, consolidated Cloud environment, the relationship between application's performance and system resources could be altered due to time-varying workload patterns, aggravated resource contention, different VM-to-host mappings, or other changes. We define such a relationship change as a *change-point*. Note that this is different from detecting changes in performance alone. For example, if a 20% increase in the workload leads to degraded performance, our module should not flag this as a change-point if the relationship between performance and system resources still holds.

We assume that the distribution of the model's prediction errors (residuals) is stationary across adjacent time intervals when there is no change in the environment. Motivated by this, we use an online change-point detection technique [28] to determine whether a change occurs by performing hypothesis testing on the model's prediction errors across adjacent time intervals.

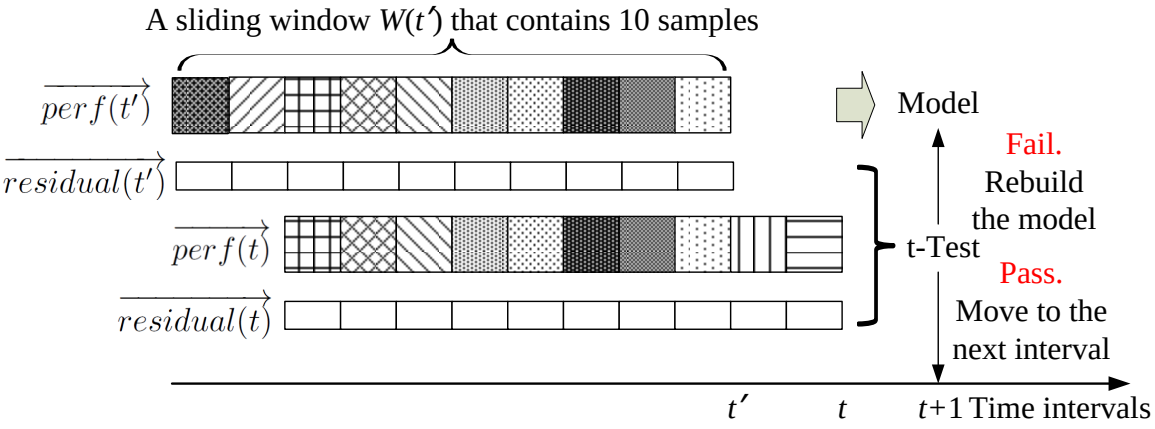


Figure 25: Model updating module

More specifically, given an existing performance model constructed in time window $W(t')$, its prediction errors in $W(t')$ and those in an adjacent time window $W(t)$ as shown in Fig. 25, we adopt an unpaired 2-sample t-test [44] to determine whether the prediction errors observed in $W(t')$ and $W(t)$ come from the same distribution, (could have the same statistical mean), i.e., the null hypothesis is that "there is no significant difference in the statistical mean between $W(t')$ and $W(t)$." If the result

of the hypothesis test suggests that to be the case, then our performance model is likely as (in)accurate as when we accepted it for use in production and, absent other information, we have no reason to discontinue using the model. Note that, a significant difference in the statistical mean is a sufficient but not necessary condition for a significant difference in the distribution and our t-testing does not assume that the error variances in $W(t')$ and $W(t)$ are equal.

4.5 Experimental setup

4.5.1 Hypervisor and sensor module

The vPerfGuard framework is generic and can work with different virtualized platforms and monitoring tools.

For the system metrics, we run VMware ESX 4.1 [17] as the hypervisor on each host. The sensor module of vPerfGuard can collect the host metrics (∼1800 metrics per host) from the "esxtop" [5] interface on ESX systems. Whereas our evaluation is done using VMware's hypervisor and tools, our framework generalizes to other virtualized platforms where similar tools exist to gather system-level metrics, e.g., "xentop" for Xen-based systems [27] and "Hyper-V Monitor Gadget" for Hyper-V-based systems [18]. For the VM metrics, we run "dstat" [1] and "iostat" [6] tools within the guest VMs so that the sensor module of vPerfGuard can collect the VM metrics (48 metrics per VM) from them.

For the application's performance metrics, we collect the throughput and response times per sampling interval directly from the benchmark workload generator. In future work, we plan to leverage monitoring tools that can measure application metrics from the hosting platform. One example of such tools is the VMware vFabric Hyperic [16], which offers out-of-the-box performance monitoring for a suite of Web applications.

4.5.2 Benchmarks and workloads

Although we run various benchmarks [3] on our virtualized testbed, due to space limitations, we focus on the results from the RUBBoS [7] and the TPC-H [15] benchmarks.

In our experiments, we deploy the RUBBoS application with the browsing-only transaction mix in a 4-tier setup, including one Apache server, two Tomcat severs, one CJDBC cluster server and two MySQL servers as shown in Figure 26(a). The sampling interval is 1 minute. We deploy the TPC-H benchmark with a scaling factor 3 using PostgresSQL. The total database size is 4571MB including all the data files and index files. The original benchmark contains 22 queries, i.e., Q1 to Q22. We choose Q6, Q7, Q12 and Q14 for our experiments because these are IO-intensive queries and they can be completed within a sampling interval of 6 minutes. For both benchmarks, we modify the original workload generator to dynamically vary the number of concurrent users in the system.

4.5.3 Testbed setup and configurations

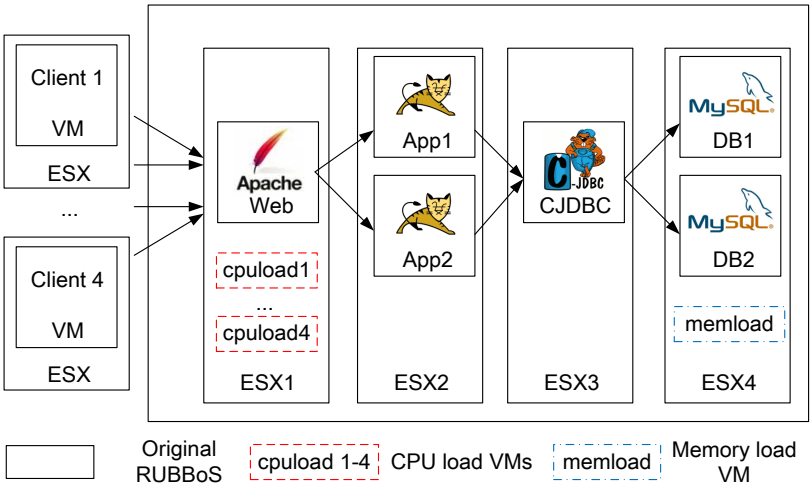
We run the RUBBoS benchmark on eight hosts, as shown in Fig. 26(a). Four hosts, *ESX1* through *ESX4*, are used to run the six VMs hosting the individual application tiers, labeled as *Web, App1, App2, CJDBC, DB1*, and *DB2*, respectively. We also run some *co-hosted* VMs on *ESX1* and *ESX4* to induce resource contention on the respective host. The four client VMs run on the other four hosts.

We run the TPC-H benchmark on three hosts shown in Fig. 26(b). Two virtual machines, $TPCH_F$ (foreground) and $TPCH_B$ (background), are deployed on one host, *ESX5*, running two instances of the TPC-H DB server. The two client VMs run on the other two ESX hosts.

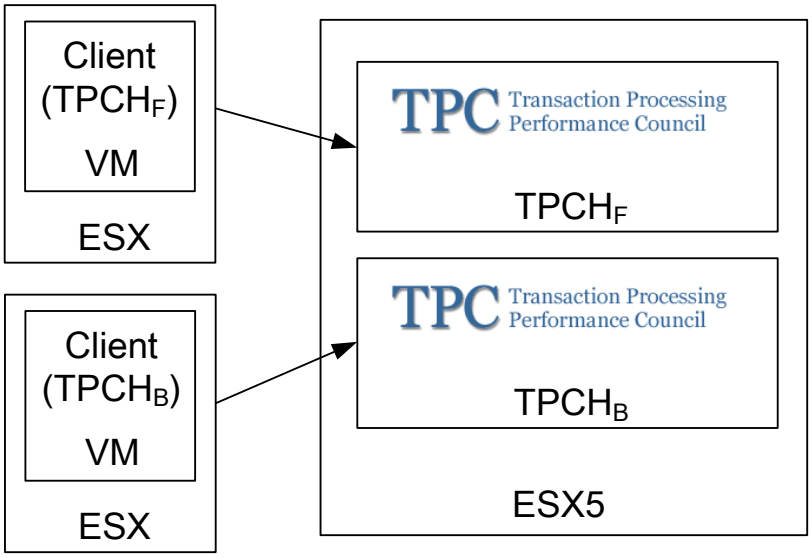
The host and VM configurations are shown in Tables 9 and 10, respectively. All

[3] RUBiS, RUBBoS with the browsing-only and the read-write transaction mixes, TPC-W and TPC-H.

the VMs run Linux kernel 2.6.32. vPerfGuard runs on a separate host.



(a) Test bed for RUBBoS benchmark



(b) Test bed for TPC-H benchmark

Figure 26: Setup of two experimental testbeds

Table 9: Configuration of hosts

Testbed	RUBBoS	TPC-H, vPerfGuard
Model	Dell Power Edge 1950	Dell OptiPlex 780
CPU	2 Intel Xeon E5420 2.5 GHz Quad-Core	1 Intel Core2 Q9650 3.0 GHz Quad-Core
Memory	32 GB	16 GB
Storage	Clariion CX-40 SAN	7200 RPM local disk

Table 10: Configuration of VMs

Testbed	RUBBoS	TPC-H
Application VM vCPU	2	4
Application VM vRAM	1 GB	2 GB
Client VM vCPU	2	4
Client VM vRAM	8 GB	4 GB

4.5.4 Naming convention

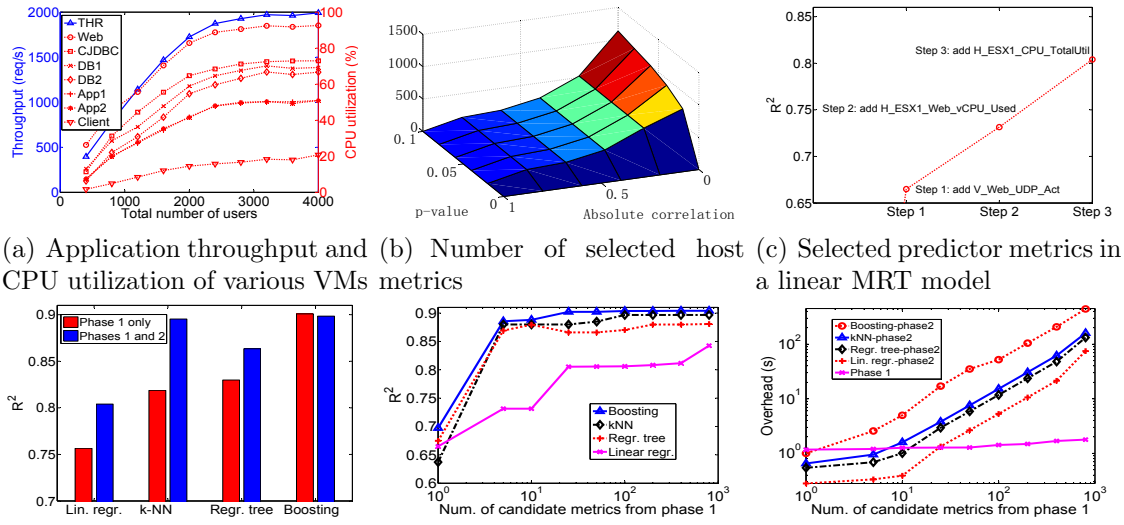
We describe the naming convention for all the metrics we collect in Table 11. For example, THR, MRT and RT_{95p} are application's performance metrics, denoting throughput, mean response time and 95th percentile response time, respectively. The other metrics that begin with H and V are host and VM metrics, respectively. For instance, $H\_ESX1\_Web\_CPU\_System$ represents the CPU "System" counter for the "Web" VM running on the "ESX1" host, and V_CJDBC_Int represents the "Interrupt" counter for the "CJDBC" VM.

Table 11: Metrics naming convention

Application's performance metrics	$THR, MRT, RT_{std}, RT_{50p}, RT_{75p}, RT_{90p}, RT_{95p}, RT_{99p}$
Host metrics	$H\_\{ESX\}\_\{VM\}\_\{Metric\}\_\{Details\}$
VM metrics	$V\_\{VM\}\_\{Metric\}\_\{Details\}$

4.6 Evaluation of model building module

In order to evaluate the model building module and compare the predictive capabilities of different performance models, we use the RUBBoS benchmark in the setup shown in Fig. 26(a), without the co-hosted VMs. We first run a calibration experiment where we vary the number of users from 400 to 4000 with a step size of 400, and observe that the application reaches a performance bottleneck at around 3000 concurrent users. This can be seen in Fig. 27(a) in the saturation of the application throughput and the near 100% CPU utilization of the "Web" VM. We then run a "random" workload for 400 minutes, where the number of users changes randomly



(a) Application throughput and CPU utilization of various VMs

(b) Number of selected host metrics

(c) Selected predictor metrics in a linear MRT model

(d) Improvement by phase 2 selection

(e) R^2 of different models

(f) Two phase algorithm overhead

Figure 27: Evaluation results for 2-phase metric selection and model building algorithm

between 400 to 4000. A sampling interval of 1 minute is used in the sensor module, resulting in 400 measurement samples that are used for the evaluation in this section. Each measurement sample is a high-dimensional vector, consisting of the following 7522 metrics: 8 application's performance metrics as shown in Table 11, 7226 host metrics from the four ESX hosts, and 288 VM metrics from the six VMs.

4.6.1 Evaluation of phase 1

For evaluation purposes, instead of limiting the number of candidate metrics from phase 1 as described in Algorithm 1, we report the number of candidate metrics selected by the phase 1 algorithm as a function of two threshold values — a lower bound, r_{LB}, on the absolute value of the correlation coefficient, and an upper bound, p_{UB}, on the p-value of the observed correlation.

We use throughput as the $perf$ metric and the 7226 ESX host metrics as the raw metrics. The number of selected host metrics is shown in Fig. 27(b). For example, for $r_{LB} = 0.8$ and $p_{UB} = 0.1$, a total of 132 metrics are selected out of the 7226 raw metrics. That means these 132 metrics (or 2% of the raw metrics) are correlated with

the observed throughput with $r_m \geq 0.8$ and $p_m \leq 0.1$. We can also infer that 98% of the raw metrics are not highly correlated with the application throughput. The number of selected metrics is reduced as the minimum correlation level increases or as the maximum p-value decreases. The latter indicates an increased level of confidence in the observed correlation.

We observe similar trends when MRT or RT_{95p} is chosen as the $perf$ metric. We also observe similar trends when we use throughput or MRT as the $perf$ metric and the 288 VM metrics as the raw metrics.

4.6.2 Evaluation of phase 2

To evaluate the phase 2 algorithm, we set the number of candidate metrics from phase 1 to be 100 and the minimum incremental R^2 improvement in phase 2 to be 0.01. For illustration, we provide an example of building a linear regression model for MRT in Figure 27(c), which shows the R^2 value for the model when one, two and three predictor metrics are selected sequentially. The phase 2 algorithm first chooses the network UDP active status of the Web VM on the $ESX1$ host ($V\_Web\_UDP\_Act$), resulting in an R^2 value of 0.668 for the single-metric linear MRT model. The algorithm then adds the second metric, the percentage of CPU Used of the Web VM on the $ESX1$ host ($H\_ESX1\_Web\_vCPU\_Used$), increasing the R^2 value of the model to 0.731. After adding the third metric, the total CPU Used on the $ESX1$ host ($H\_ESX1\_CPU\_TotalUtil$), the algorithm stops searching because the model quality improvement falls below the minimum incremental R^2 improvement threshold (0.01) when a 4th metric is added.

To evaluate the impact of the phase 2 metric selection algorithm, Fig. 27(d) reports the R^2 values for different model types in two scenarios: (1) using the 100 candidate metrics from phase 1 directly as the predictor metrics (without phase 2 selection), and (2) using the smaller number of predictor metrics selected from phase 2. For

the first three model types (linear regression, k-NN, regression tree), the additional metric selection in phase 2 helps improve the accuracy of the final model.

4.6.3 Discussion

The effectiveness of the two-phase algorithm depends on the values of the tunable parameters, including (1) the number of selected candidate metrics from phase 1 (N_{can}), (2) the type of model F chosen in phase 2, and (3) the minimum incremental R^2 improvement in phase 2 (R^2_{inc}). We evaluate the impact of these parameters in two aspects — model accuracy and computation overhead. We assume that MRT is chosen as the $perf$ metric.

First, we fix the minimum incremental R^2 improvement in phase 2 at 0.01, and vary the other two tunable parameters. More specifically, for each of the four types of models, we limit the number of candidate metrics from phase 1 at [5, 10, 25, 50, 100, 200, 400, 800], and run the phase 2 algorithm to build a model for the MRT. In both experiments, a 10-fold cross validation [100] is used to compute the R^2 value for each model type.

Figure 27(e) shows the R^2 value of the final model as a function of the number of candidate metrics from phase 1 (in *log* scale). Different lines represent different model types. We make the following observations: (1) As more metrics are selected in phase 1, the model accuracy from phase 2 is generally improved for all four model types; (2) All the models achieve reasonably good accuracy ($R^2 > 0.8$) with 25 or more candidate metrics from phase 1, although the linear model's R^2 value is slightly lower than those from the nonlinear models.

Figure 27(f) shows the computation time of both phase 1 and phase 2 as a function of the number of candidate metrics from phase 1 (in *log* scale). Different lines for phase 2 represent different model types. We make the following observations: (1) The phase 1 overhead increases slowly with the number of candidate metrics; (2) For

all four model types, the overhead in the phase 2 algorithm grows as we increase the number of candidate metrics from phase 1; (3) The boosting model has the most overhead, and the linear regression model has the least.

For demonstration purposes, we also run the phase 2 algorithm directly on all the raw metrics, without initial metric selection in phase 1. The result shows that, for all the model types, the metric selection in phase 1 helps achieve better accuracy in the final model as well as reducing the overhead in model building in phase 2.

Second, we run similar experiments to evaluate the effect of the minimum incremental R^2 improvement in phase 2 (R^2_{inc}). We do not show the detailed results here due to space limitation. In summary, as this threshold value becomes smaller, we have better model accuracy and more computational overhead in phase 2. We find that the threshold value of 0.01 strikes a good balance between better accuracy and lower overhead.

Finally, we choose the linear regression model as our default model because it has the best *human-interpretability* and lowest overhead with only slightly lower accuracy relative to the nonlinear models. In spite of better accuracy, the regression tree is not a good candidate because (1) if we use shallow trees, the marginal ratio between performance and predictor metric is zero at most points, making it unable to infer which system metric is the bottleneck, and (2) if we use deep trees, the over-fitting issue prevents the model from generalizing faithfully. The k-NN and boosting approaches are also not preferred because they are harder to interpret directly due to model complexity.

4.7 Evaluation of model updating module

To evaluate the model updating module of vPerfGuard, we run the tool against four typical, dynamic workload scenarios a Cloud service provider may experience, including an application's performance bottleneck caused by a surge in the workload

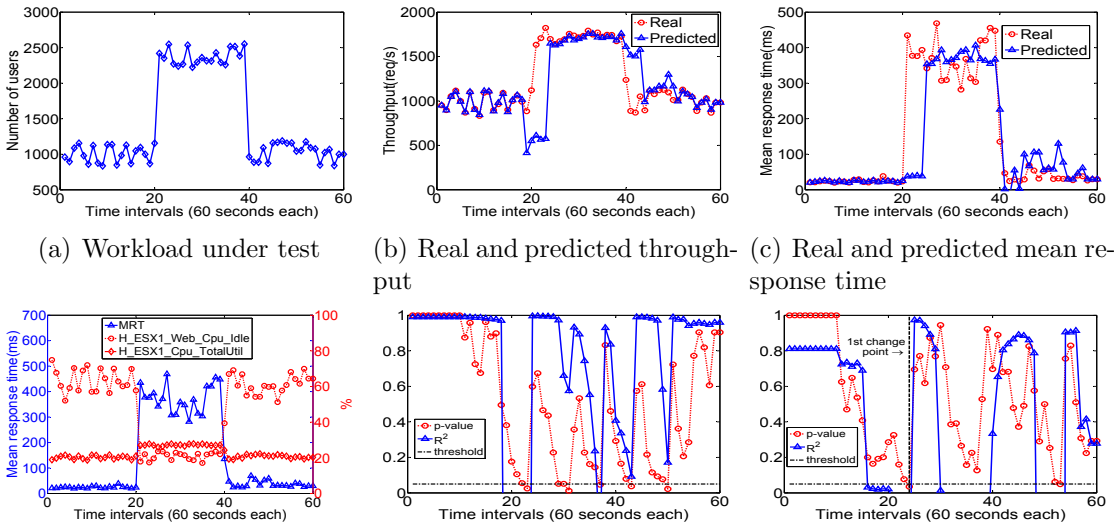
intensity, and performance degradation in one application due to the CPU, memory, or disk I/O contention from co-hosted VMs (*aka. noisy neighbors*).

Our evaluation criteria focuses on three aspects of the performance models generated: (1) **prediction:** whether a model provides an accurate prediction of the application's performance using the selected system metrics; (2) **identification:** whether the selected system metrics point to the correct performance bottlenecks, including the critical application component, the resource under contention, or the host where the contention occurred; (3) **adaptivity:** whether the model is adaptive to the changes in relationship between application's performance and system resources. Note that we do not expect the human analyst to interpret the models directly. We will show a graphical user interface in the next section to illustrate how the top suspicious metrics and the associated coefficients can be presented to the user for inspection.

The following subsections describe the experimental settings of the four scenarios, present the detailed results in Figures 28-31, and summarize the evaluation in Tables 12 and 13. In particular, each figure is organized as follows. Fig. (a) shows the workload(s) used, Fig. (b) and Fig. (c) compare the real and the model-predicted throughput and mean response time (MRT), respectively, Fig. (d) shows the MRT and the top selected metrics in the MRT model, and finally, the model accuracy measures including the p-value and the R^2 value for the throughput and the MRT models are shown in Fig. (e) and Fig. (f), respectively.

4.7.1 Workload surge

A. Experimental settings In this scenario, we use the testbed in Fig. 26(a) with the RUBBoS application only (i.e., no co-hosted VMs). The browsing-only workload mix is run for an hour (60 time intervals), with a surge in the workload intensity that goes from 1000 to 2300 users (with small, random variation) and lasts from the 21st to the 40th intervals (Fig. 28(a)).



(a) Workload under test

(b) Real and predicted throughput

(c) Real and predicted mean response time

(d) MRT and top selected metrics

(e) Model accuracy for throughput

(f) Model accuracy for mean response time

Figure 28: Experimental results for the workload surge scenario

B. Evaluation According to the experimental setting and our previous knowledge from Fig. 5(a), the mean response time increases during the workload surge period due to the CPU resource bottleneck in the Web tier of the RUBBoS application. As shown in Fig. 28(e) and Fig. 28(f), the online module detects a change-point multiple times, resulting in 6 throughput models and 3 MRT models for the duration of the experiment, such that we maintain a high confidence in the learned models (p-value ≥ 0.05). The throughput models starting from the 1st, 24th, 32nd, 38th, 44th and 51st intervals are:

$$THR = 0.19 * V\_CJDBC\_Int - 519.60,$$

$$THR = 0.05 * H\_ESX1\_Web\_vSwitch\_PcksTrans/s + 36.25$$

$$THR = 28.53 * H\_ESX2\_App2\_vSwitch\_MBitsRec/s + 22.53 * V\_DB2\_CPU\_Sys + 358.79$$

$$THR = 16.16 * H\_ESX2\_CPU\_Idle\_Overlap + 1277$$

$$THR = 0.08 * V\_Web\_ContextSwitch - 202.37$$

$$THR = 28.53 * H\_ESX2\_App2\_vSwitch\_MBitsRec/s + 113.75$$

The MRT models for the mean response time starting from the 1st, 25th and 54th intervals are:

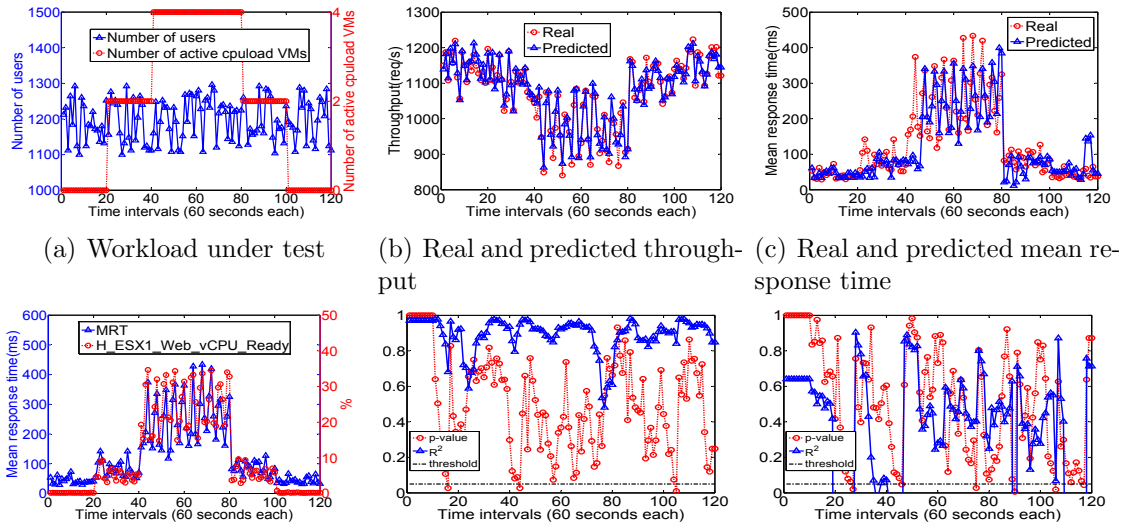
$$MRT = 2.17 * H\_ESX1\_CPU\_TotalUtil - 20.91$$

$$MRT = -8.10 * H\_\mathbf{ESX1\_Web\_CPU\_Idle} + 545.10$$

$$MRT = 0.49 * V\_CJDBC\_UDP\_Act - 162.75$$

We make the following observations. (1) The models have reasonably good prediction capability as shown in Figs. 28(b), 28(c) and Table 12. The signs of the coefficients in each THR or MRT model make sense, e.g., when throughput increases, interrupts, CPU utilization, context switches and network packets transmitted or received also increase. (2) The selected system metrics in all the THR models do not point to the correct cause of the performance degradation. Three of the six models choose network attributes as the top metrics, and two other models choose system interrupts or context switches as the top metrics. The selected system metric $H\_ESX1\_Web\_CPU\_Idle$ in the second MRT model as shown in Fig. 28(d) directly points to not only the bottleneck host (ESX1), but also the bottleneck VM (Web) and the critical resource (CPU).

Following the above observations, we conclude that, (1) both THR and MRT models have good *prediction* capability, and THR models have better prediction accuracy due to its linear relationship with many system metrics, and (2) THR models are not suitable for critical resource identification, and MRT models have good *identification* capability during the periods of performance bottlenecks. This observation is also validated in the next three scenarios and in [51]. When the application experiences a performance bottleneck, the THR remains almost constant, making it harder for our correlation-based selection to identify critical system metrics. At the same time, a small change in a critical system metric may lead to a large change in the MRT, making it easier for our algorithm to identify the correlation. For conciseness, we



(a) Workload under test (b) Real and predicted throughput (c) Real and predicted mean response time

(d) MRT and top selected metric (e) Model accuracy for throughput (f) Model accuracy for mean response time

Figure 29: Experimental results for the CPU contention scenario

do not show the throughput models in the following scenarios, and we show only the MRT models during the periods of performance degradation to illustrate their identification capability.

The MRT models are also *adaptive* to the surge in the workload, with only 3 intervals of delay in response. As Fig. 28(f) shows, after the workload surge at the 21th interval, the application MRT increases dramatically. It takes 3 intervals for the p-value of the first MRT model to drop below the threshold value of 0.05. The vertical line in the figure (24th interval) indicates where the first change-point is detected, after which the Web VM's CPU idle time is chosen as the key metric for learning a new MRT model starting from the next interval.

4.7.2 CPU contention

A. Experimental settings In this experiment, we run the RUBBoS benchmark with a workload intensity randomly varying between 1100 and 1300 users, as shown in Fig. 29(a). To create a CPU contention scenario, we use the four co-hosted VMs, *cpuload*1 to *cpuload*4, as noisy neighbors, to share the CPUs on the ESX1 host

with the RUBBoS Web VM (see Fig. 26(a)). Each *cpuload* VM is configured with 2vCPUs, 1GB vRAM, and runs a "CPU eater" application that consumes the vCPUs at a specified utilization level for a specified time period. We let the CPU utilization of each VM vary periodically between 10% and 40% with a period of 2 minutes. All the four co-hosted VMs are idle for the initial 20 time intervals. We then start the workload in *cpuload*1 and *cpuload*2 at the 21st interval, and start *cpuload*3 and *cpuload*4 at the 41st interval. The workloads in these VMs are idle again at the 81st and 101st intervals (Fig. 29(a)).

B. Evaluation According to the experimental setting, the application's mean response time starts to increase after the 21st interval due to the CPU resource bottleneck on the ESX1 host, caused by the active workloads in the four *cpuload* VMs. The MRT models starting from the 28th, 47th, 76th and 91st intervals are:

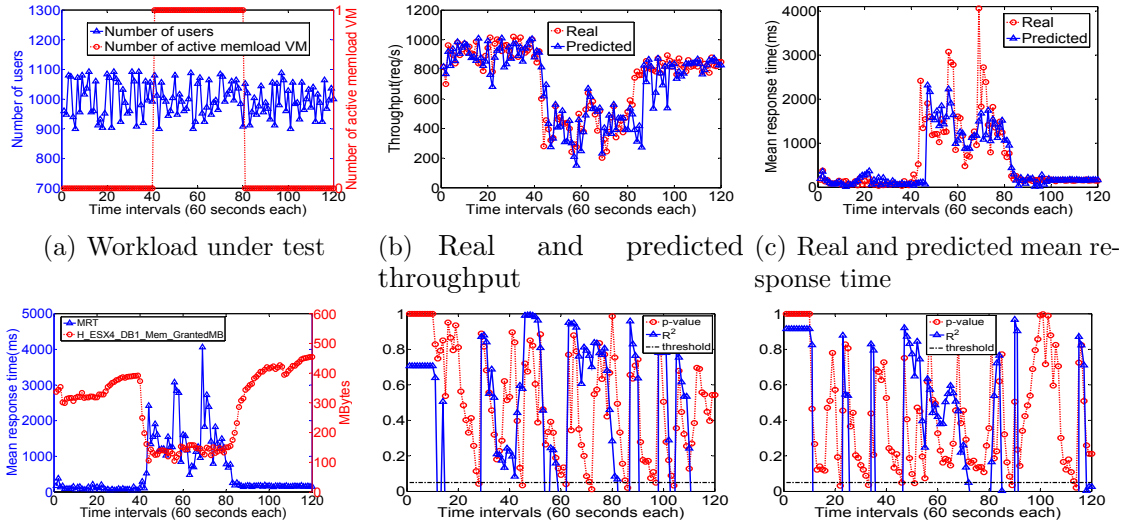
$$MRT = 1.97 * H\_ESX4\_DB1\_Mem\_Active + 1.13 * H\_\mathbf{ESX1\_CPU\_Util} - 89.7$$

$$MRT = 752.76 * H\_\mathbf{ESX1\_CPULoad}\_1MinuteAvg - 562.87$$

$$MRT = 12.48 * H\_\mathbf{ESX1\_Web\_vCPU\_Ready} - 25.01$$

$$MRT = 6.38 * H\_\mathbf{ESX1\_Web\_vCPU\_Ready} + 48.72$$

We make the following observations. (1) These MRT models have good prediction capability as shown in Figs. 29(b), 29(c) and Table 12. (2) These models have good identification capability because they all point to the correct performance bottleneck. For example, one of the top system metrics, $H\_ESX1\_Web\_vCPU\_Ready$, as shown in Fig. 29(d), specifies not only the bottleneck host (ESX1), but also the bottleneck VM (Web) and the critical resource (CPU). (3) The models are adaptive to the increased CPU load from the noisy neighbors as shown in Fig. 29(f). After two of the cpuload VMs become active in the 21th interval, it takes the model updating module 6 intervals to detect the change and build a new model.



(a) Workload under test (b) Real and predicted throughput (c) Real and predicted mean response time

(d) MRT and top selected metric (e) Model accuracy for throughput (f) Model accuracy for mean response time

Figure 30: Experimental results for the memory contention scenario

4.7.3 Memory contention

A. Experimental settings In this experiment, we run the RUBBoS benchmark with a workload intensity randomly varying between 900 and 1100 users, as shown in Fig. 30(a). Because the size of the MySQL database is 498.88MB, the total size of database files on the two database VMs, i.e., $DB1$ and $DB2$, are approximately 1GB. We use a co-hosted VM, *memload* (configured with 4vCPUs, 1GB vRAM), as the noisy neighbor, to run on the $ESX4$ host along with $DB1$ and $DB2$ (see Fig. 26(a)). To create memory contention, we configure the $ESX4$ host with 4GB of physical memory. Since about 3GB of memory is reserved by the hypervisor, only 1GB of memory is available for the three VMs ($DB1$, $DB2$, and *memload*) to share. As a result, the total memory commitment during these 40 intervals is much more than the shared 1GB memory.

For the initial 40 intervals, the *memload* VM remained idle, so the total memory commitment on $ESX4$ is close to 1GB. Between the 41st and the 80th intervals, a four-thread "memory eater" application is started inside the *memload* VM. Each thread in the application allocates 120-180MB memory and randomly touches the

allocated pages to keep them actively used. As a result, the total memory commitment during these 40 intervals is much more than the shared 1GB memory. When the RUBBoS DB servers cannot access enough physical memory, more requests require disk accesses, reducing throughput (Fig. 30(b)) and increasing response times (Fig. 30(c)).

B. Evaluation The MRT models starting from the 47th and 52th intervals are:

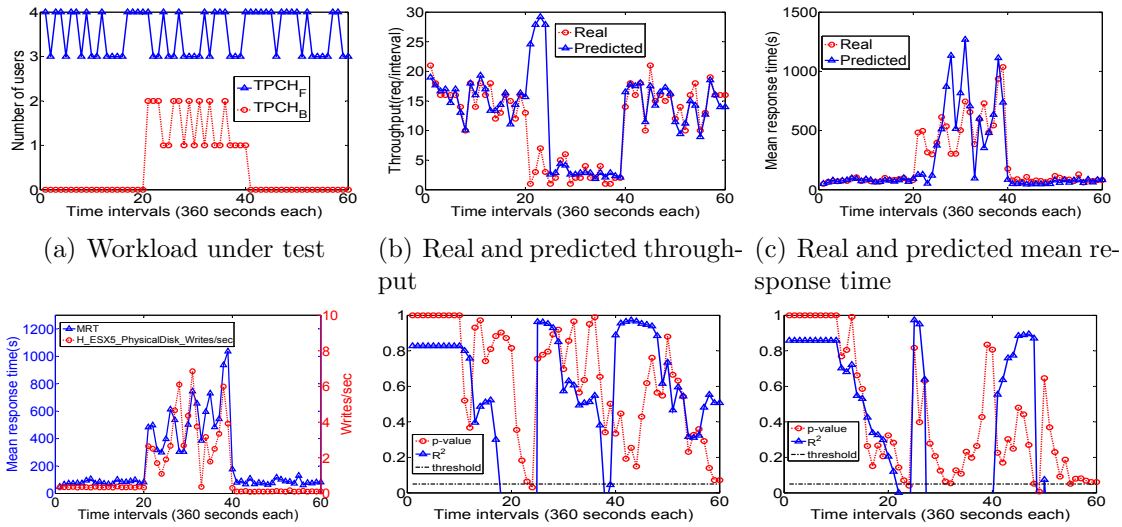
$$MRT = -0.75 * H\_ESX4\_Network\_PksReceived/sec + 3072.86$$

$$MRT = -25.38 * H\_\mathbf{ESX4\_DB1\_Mem\_GrantedMB} + 4876.55$$

We make the following observations. (1) The models have reasonably good prediction capability as shown in Figs. 30(b), 30(c) and Table 12. (2) The second model has good identification capability, since the top system metric, $H\_ESX4\_DB1\_Mem\_GrantedMB$ as shown in Fig. 30(d), indicates not only the bottleneck host (ESX4), but also the critical resource (Memory). (3) The models are adaptive to the increased memory load in the system with 10 intervals of delay as shown in Fig. 30(f).

4.7.4 Disk I/O contention

A. Experimental settings In this experiment, we run two instances of the TPC-H benchmark in parallel. A foreground database VM ($TPCH_F$) and a background database VM ($TPCH_B$) are co-located on the $ESX5$ host, as shown in Fig. 26(b). Since the total database size is 4571MB, which cannot fit into the VM's 2GB vRAM, some of the queries much involve disk I/O. Fig. 31(a) shows the workloads used in both VMs with a 6 minute sampling interval. The workload for $TPCH_F$ has the number of users randomly varying between 3 and 4 throughout the experiment. The background VM, $TPCH_B$, is idle for the initial 20 intervals. Between the 21st and the 40th intervals, the TPC-H workload is activated inside the $TPCH_B$ VM, a noisy neighbor, with the number of users randomly varying between 1 and 2.



(a) Workload under test (b) Real and predicted throughput (c) Real and predicted mean response time

(d) MRT and top selected metric (e) Model accuracy for throughput (f) Model accuracy for mean response time

Figure 31: Experimental results for the disk I/O contention scenario

B. Evaluation The MRT model between the 25th and 49th intervals for the $TPCH_F$ application is:

$$MRT = 180.62 * H\_\textbf{ESX5\_PhysicalDisk\_Writes/s} + 29.24$$

We make the following observations. (1) The models have reasonably good prediction capability as shown in Figs. 31(b), 31(c) and Table 12. (2) The model has good identification capability because the top selected system metric $H\_ESX5\_PhysicalDisk\_Writes/s$ points not only to the bottleneck host (ESX5), but also to the critical resource (disk I/O), as shown in Fig. 31(d). Note that database servers often write temporary files (e.g., sorting files) to the disk when the physical memory is scarce. (3) The models are adaptive to the occurrence of the disk I/O bottleneck with only 3 intervals of delay as shown in Fig. 31(f).

4.7.5 Evaluation summary

For the four dynamic workload scenarios that we test, we summarize the evaluation results for the model prediction and the model identification accuracies of vPerfGuard.

A. Prediction Besides showing the model prediction accuracy in R^2 over a sliding window of samples in the previous figures, we also report in Tab. 12 the statistics (mean, standard deviation, 50th percentile, 90th percentile) of the relative error for the individual THR and MRT samples, i.e., ($|(predicted - real)/real|$). In the last column, we also show the *overall* relative error as (sum of $|predicted - real|$)/(sum of $real$). We can see that vPerfGuard achieves reasonably good prediction accuracy. We also notice that the relative error for THR is much smaller than that for MRT in all four scenarios except the disk I/O contention scenario. This validates our earlier observation in Sec. 4.7.1 that linear models capture the relationship between application throughput and system metrics well in most cases.

Table 12: Relative error for THR and MRT

Scenarios(perf)	Mean(std)	50p	90p	overall
Workload(THR)	0.10(0.20)	0.01	0.53	0.10
Workload(MRT)	0.41(0.64)	0.12	0.96	0.35
CPU(THR)	0.01(0.01)	0.01	0.03	0.01
CPU(MRT)	0.29(0.26)	0.23	0.64	0.27
Memory (THR)	0.12(0.16)	0.06	0.33	0.10
Memory (MRT)	0.51(0.45)	0.25	0.95	0.37
Disk I/O(THR)	0.95(3.32)	0.11	1.58	0.24
Disk I/O(MRT)	0.29(0.39)	0.16	0.70	0.39

B. Identification Table 13 shows the identification accuracy of the MRT models using *precision* and *recall* measures from pattern recognition literature, computed only for the performance bottleneck period. In our context, we define precision to be the fraction of all the selected metrics that are relevant (i.e., point to the correct bottleneck); and if a metric appears in n intervals, it's counted n times. We define recall to be the fraction of all the intervals in which the selected metrics are relevant; and for intervals with multiple selected metrics, that interval is counted using only the fraction of the relevant metrics. We report precision and recall for the detection of bottleneck resource and bottleneck host, separately. We use the CPU contention

scenario as an example where the length of the bottleneck period is 80 intervals. In intervals 21-27, the model contains an irrelevant metric; in intervals 28-46, the models contains two metrics with one being relevant; in the remaining intervals the model contains one relevant metric. Hence, precision=$(19+54)/(7+19\times 2+54) = 74\%$, and recall=$(19/2+54)/80 = 79\%$. In the last column, we also report the delay in change-point detection in number of intervals. We can see that models built by vPerfGuard achieve good identification accuracy in terms of precision and recall, with short delays in model updates.

Table 13: Identification summary

Scenarios	Precision		Recall		Delay
	resource	host	resource	host	
Workload	100%	100%	100%	100%	3
CPU	74%	74%	79%	79%	6
Memory	73%	85%	73%	85%	10
Disk I/O	80%	100%	80%	100%	3

4.7.6 Discussion

A. Adaptation overhead In Table 14, we show the mean and standard deviation of the overhead in running vPerfGuard online for four dynamic scenarios, using 10 consecutive samples for both model building and change-point detection. The average overhead is 67ms (standard deviation = 37ms) for the sensor module to pull the application's performance metrics and the system metrics from the hosts and the VMs. The metric selection and model building module takes an average of 221ms, the longest among all. The average model testing time is 54ms, and the average hypothesis testing time is 5ms.

Table 14: Online model adaptation overhead(mean(std))

Metrics collection(ms)	Building time(ms)	Testing time(ms)	Hypothesis testing(ms)
67(37)	221(148)	54(23)	5(18)

Table 15: Sensitivity analysis

Method(# of samples)	average R^2	# of models (correct)	# of metrics (correct)
p-value (10)	0.62	4 (4)	5(4)
p-value (20)	0.62	3 (2)	4 (2)
p-value (30)	0.76	3 (0)	3 (0)
R^2 (10)	0.74	49 (32)	56 (32)
R^2 (20)	0.73	40 (18)	45 (18)
R^2 (30)	0.80	38 (12)	38 (12)

B. Sensitivity analysis We perform a sensitivity analysis using different change-point detection criteria (p-value < 0.05 or $R^2 < 0.8$) or a different number of samples to explore the tradeoff between prediction accuracy and identification accuracy. In Table 15, we summarize the average positive R^2 value, the number of models generated, the number of models indicating the correct bottleneck, the total number of selected metrics in all the models, and the number of metrics identifying the correct bottleneck for the CPU contention scenario, during the contention period. As we increase the number of samples, the average R^2 increases (as one would expect), but the percentage of correct models or metrics decreases. For change-point detection, if a criterion of $R^2 < 0.8$ is used instead of using hypothesis testing with p-value < 0.05, we may generate too many models (due to over-fitting) for the human analyst to reason about. This result indicates that using hypothesis testing is a more robust method for change-point detection than using the R^2 value directly.

4.8 Visualization of results

We introduce a primitive graphic user interface (GUI) for visualization of the results from vPerfGuard. The GUI consists of four panels: a configuration tab, a real-time tracking tab, a real-time analysis tab, and a real-time diagnosis tab. The configuration tab is used to configure the necessary parameters for vPerfGuard to operate. The real-time tracking tab consists of two windows: one shows the real and the predicted performance metric values for comparison; the other shows the corresponding p-value

and R^2 value. Next we describe the other two tabs using the workload surge scenario as an example.

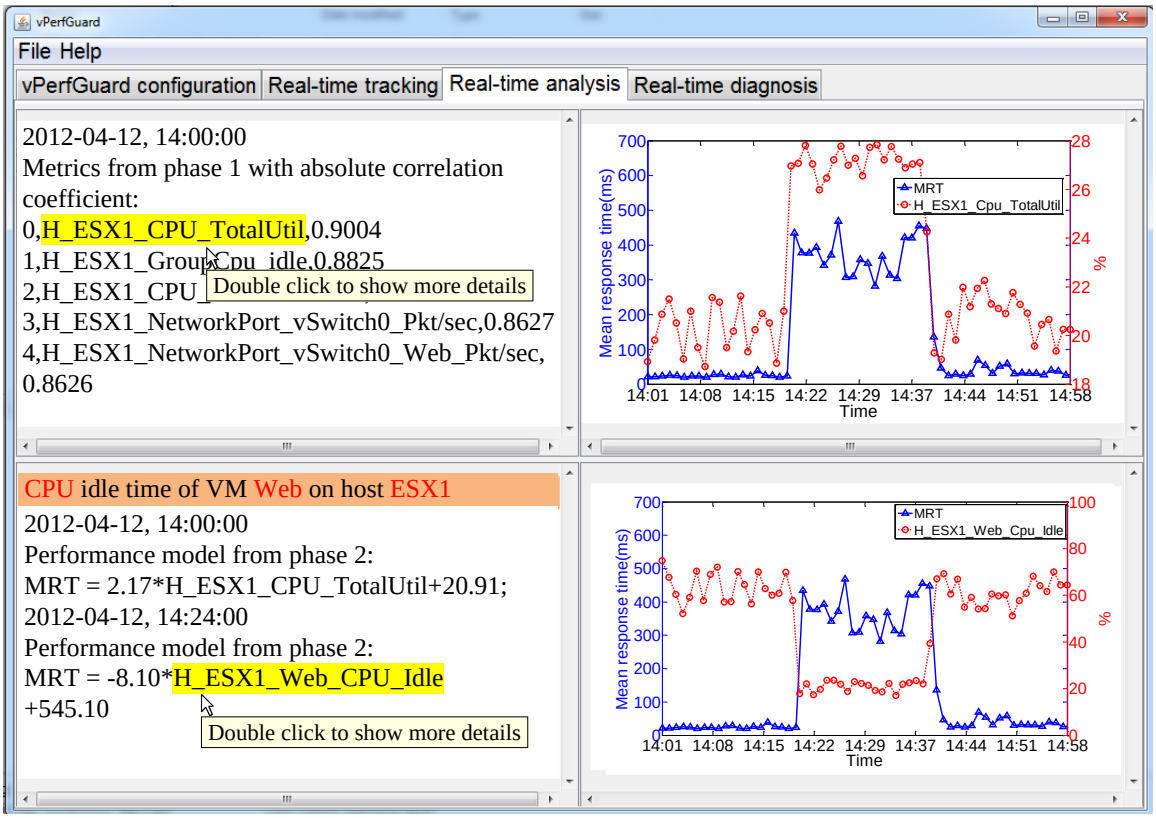


Figure 32: GUI for real-time analysis

The real-time analysis tab (shown in Figure 32) presents the selected metrics and models in real time. The top-left window shows the selected metrics from phase 1 with absolute correlation coefficient in descending order. If a highlighted metric is double-clicked, the time series of the metric and the application's performance metric will be shown in the top-right window. In the same tab, the bottom-left window shows the series of models built in phase 2 with the top system metrics. Because an abstract model may be hard to interpret by a Cloud service provider, when a highlighted metric is double-clicked, the GUI translates the abstract metric name into human readable description at the top of the window. At the same time, the time series of the metric and the application's performance metric are displayed in

the bottom-right window.

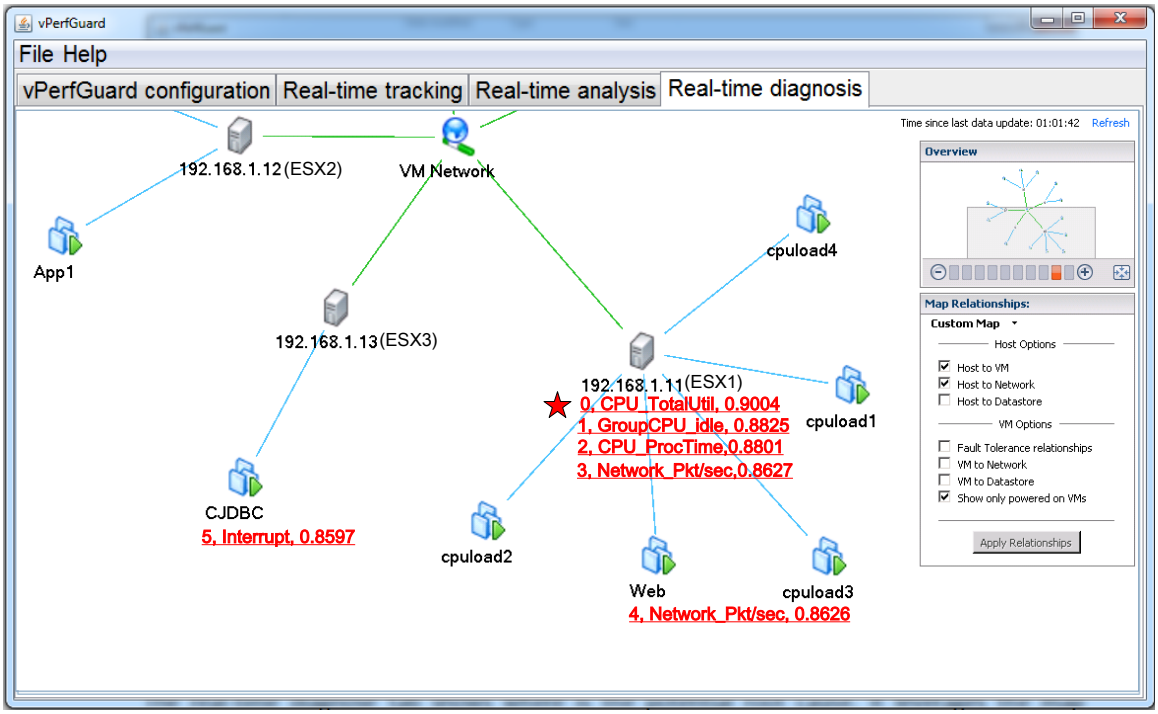


Figure 33: GUI for real-time diagnose

The real-time diagnosis tab (shown in Figure 33) points to possible performance bottleneck locations for critical resource identification purpose. We utilize vCenter map [17], a visual representation of the vCenter Server topology that captures the relationships between the virtual and physical resources managed by the vCenter Server. The selected system metrics are overlayed on top of the associated components in the system. We use red fonts to locate the metrics from phase 1 and red star icon to locate the metrics from phase 2, respectively. As a result, this tab offers a Cloud service provider better visibility into the potential bottlenecks in a complex and distributed environment.

One potentially useful feature we would like to implement is allowing a user to manually add a metric to the model, by right clicking on a specific metric. This offers a Cloud service provider an interface to provide inputs to the identification process by applying domain knowledge. While model-driven approach is useful in highlighting

potential bottlenecks, incorporating domain knowledge from an experienced human analyst may lead to even better results in the timely determination of the real root causes of the performance problems.

4.9 Summary

In this chapter, vPerfGuard is proposed to enhance a classical critical resource identification approach by leveraging statistical machine learning techniques to address the novel technical challenges due to high scalability and adaptability requirements for a complex distributed application in the Cloud. The strengths and weaknesses of vPerfGuard compared with a classical critical resource identification approach are summarized as below:

4.9.1 Strengths

vPerfGuard is able to automatically identify the critical resource from thousands of system raw metrics because the model building module of vPerfGuard will choose tens of candidate metrics through correlation-based selection and then choose several predictor metrics through model-based selection. Thus, vPerfGuard will show its significant strength when the number of raw metrics is huge. However, since a classical critical resource identification approach often needs to manually check the correlation between performance metrics and system raw metrics, thousands of system raw metrics will present scalability challenge for a classical critical resource identification approach.

vPerfGuard is able to adaptively change the performance model because the model updating module of vPerfGuard will trigger the model renewing process based on the results from statistical hypothesis tests. The updated model will be able to identify the updated critical resources. Thus, vPerfGuard will show its strength when the performance model demands updating due to the variation of workload and environment. However, since a classical critical resource identification approach is insensitive to the

variation workload and environment, the dynamic workload and environment will present adaptivity challenge for a classical critical resource identification approach.

4.9.2 Weaknesses and future work

vPerfGuard will show its limitation when it is expected to independently discover the root cause of the application's performance degradation. Note that vPerfGuard could automatically select critical metrics from thousands of system raw metrics. The metrics would help a Cloud service provider such as a PaaS provider to zero in the root cause of the performance degradation based on the correlation. However, since correlation does not imply causation, a PaaS provider still needs to further confirm the root cause of the performance degradation based on the selected critical metrics from vPerfGuard. In the future, we plan to add an interface to vPerfGuard in order to incorporate domain knowledge from an experienced human analyst to determine the root cause.

vPerfGuard may show its limitation in adapting to the change-point when it is applied to identify the critical resource under the scenarios of non-step-wise workload or non-step-wise resource contention. Note that vPerfGuard is only evaluated in workload, CPU, memory and disk I/O contention scenarios under step-wise workload and step-wise resource contention. It is not evaluated in scenarios under non-step-wise workload or non-step-wise resource contention, which will be left for future work.

CHAPTER V

HIERARCHICAL RESOURCE ALLOCATION

In the previous chapter, a critical resource identification framework is proposed to address the high scalability and adaptability challenges to identify critical resources for a complex distributed application in a Cloud environment. After the critical resource that affects the application's performance is identified, a resource allocation controller can be applied.

In this chapter, a classical resource allocation controller is enhanced by leveraging hierarchical resource management to address novel technical challenges due to the heterogenous resource type and amount requirements for components in a multi-component application in the Cloud in order to achieve the highest resource utilization.

5.1 Background

A common usage scenario of dynamic resource allocation is for a platform-as-a-service (PaaS) provider who hosts an application and rents resources from an infrastructure-as-a-service (IaaS) provider. The PaaS provider makes revenues through the delivery of client request service under service-level agreements. And the PaaS provider pays for the bill of renting resources similarly to other commodities. The PaaS provider could make use of a resource allocation controller, which requires/releases critical resources when workload increases/decreases to maintain service-level compliance for an application with the lowest cost.

Due to the convenience of providing a model for developers to create a flexible and reusable application, more and more multi-component applications are being deployed in the Cloud nowadays. A multi-tier web application is a typical multi-component

application. We show an example multi-tier web application called vSlashdot in Figure 23 in the previous chapter which composes of several components such as *Web servers* in the presentation tier, *App servers* in the application tier and the *DB servers* in the database tier.

In this chapter, we propose ERController(Economical and Robust Controller), which enhances a classical resource allocation controller by leveraging hierarchical resource management to help a PaaS provider achieve the lowest resource cost while guaranteeing service-level compliance for multi-component applications such as multi-tier web applications in the Cloud environments.

The rest of this chapter is organized as follows. Section 5.2 and Section 5.3 give the problem definition and the solution overview, respectively. Section 5.4 outlines our experimental setup in this section. Section 5.5 models a multi-tier web application as a tandem queue and proposes an optimal resource partition method, which is evaluated in Section 5.6. Section 5.7 explores the relationship between the total resource and the mean round trip time, and designs an application controller. The performance controller which integrates the application controller and the resource partition method is evaluated in Section 5.8. Section 5.9 summarizes the chapter.

5.2 Problem definition

We provide the problem definition following the previously defined equation where "Profit = Revenue - Cost". We define the problem as how to help a platform-as-a-service (PaaS) provider who hosts a multi-tier web application to achieve the most profit through dynamic resource allocation. There are several assumptions.

We assume the service-level compliance as the mean round trip time of all the requests to the multi-tier web application. Although SLA cost function for the mean round trip time may have various shapes, we believe that a step-wise function is a natural choice used in the real-world contracts as it is easy to describe in natural

language [87]. We use a single step function for SLA in this chapter as a reasonable approximation. We assume that if the round trip time of the request is shorter than Ref (reference time), then the service provider will earn some revenue. Otherwise, the service provider will pay a penalty back to the client. As a result, in order to achieve the most revenue without penalty, the performance controller should keep the response time right below Ref. Note that the step-wise SLA cost function that we use in this chapter is very similar to the SLA cost function in Figure 10(b) in Chapter 3. In both SLA cost functions, the service provider will earn some revenue when Ref (reference time) in this chapter or τ (deadline) in Chapter 3 is met and will pay a penalty cost back when Ref (reference time) in this chapter or τ (deadline) in Chapter 3 is not met. The difference between the two is that, the SLA cost function is corresponding to a single query in Chapter 3 while the SLA cost function is corresponding to a statistical value (mean round trip time) in this chapter. Both types are widely adopted in research and industry and there exist techniques (e.g., [52]) that directly map quantile-based SLAs to per-query SLAs.

According to the previous assumption where the PaaS provider achieves the most revenue because all the requests are admitted and meet their deadline, we need to minimize the cost in order to maximize the profit. The problem becomes how to make multi-level resource allocation decisions that can guarantee service-level compliance with the lowest resource cost.

A classical resource allocation controller often allocates the same resource amount or keeps the same resource utilization for all the components within an application. However, different components within an application have heterogenous requirements for both resource types and amounts. This presents novel technical challenges that demand enhancement to a classical resource allocation controller in order to guarantee service-level compliance with the lowest resource cost. There are two key technical challenges: (1) Based on service-level compliance, what is the total resource budget;

and (2) How to partition the total resource budget to each tier. The two challenges are closely related with each other. The total resource budget affects the partition scheme by setting an upper bound for the resource that could be partitioned. A partition scheme affects the total resource budget through the service-level compliance vice versa.

We assume that the CPU resource is the critical resource for the multi-tier web application's performance and we will use the notations in Table 16 in this section. Specifically, we use S to denote the total resource budget and use RTT to denote mean round trip time. We use "entitlement" (u) and "consumption or usage" (c) to refer to the CPU shares (in percentage of total CPU capacity) allocated to a virtual machine and the CPU share actually used by the virtual machine respectively. We use "utilization" (r) to refer to the ratio between consumption and entitlement, i.e,

$$r = \frac{c}{u}$$

Table 16: Notations

S	total resource budget
RTT	mean round trip time
k	control interval for container level controller
K	control interval for application level controller
N	number of tiers (e.g., Web, App, DB)
Ω	number of transaction types (e.g., Browse, Bid)
T_{cpu}	average resident time on CPU resources
T_{others}	average resident time on non-CPU resources
λ_ω	average arrival rate of transactions type ω
λ	aggregate arrival rate of all transaction types
α_ω	average service time of non-CPU resources of transaction type ω
u_n	CPU entitlement that is allocated to the virtual server at tier n
c_n	CPU consumption of the virtual server at tier n
r_n	CPU utilization of the virtual server at tier n

The problem can be defined as how to design a performance controller with a partition scheme that can guarantee service-level compliance with the least of total

resource budget as shown below.

$$\text{Minimize} \quad S = \sum_{n=1}^{N} u_n$$

$$\text{s.t.} \quad RTT \leq Ref$$

Note that the total resource "consumption" of all the tiers/components is constant since all the requests are admitted and meet their deadline. S in the previous definition could be treated as the total resource "entitlement". Thus, the problem of how to make multi-level resource allocation decisions that can guarantee service-level compliance with the lowest resource cost is equal to the problem of how to make multi-level resource allocation decisions that can guarantee service-level compliance with the highest resource utilization.

5.3 Solution approach overview

In order to solve the above problem and address the novel challenges due to heterogenous resource type and amount requirements for components in a multi-component application in the Cloud, in this chapter, we develop ERController [1] as shown in Figure 34 to guarantee service-level compliance with the least of total resource budget. We use k and K to denote the control intervals of the container level and application level controllers respectively. On the application level, an application controller is used for end-to-end performance guarantee of the whole application through dynamic tuning of the total amount of the resources allocated to the application. The controller works in 90 seconds time interval. On the container level, there is one resource partition controller that is to allocate the total resource to the application tiers or the containers. The controller works in 10 seconds time interval.

The main contributions of this chapter are twofold:

[1] ERController stands for: Economical and Robust Controller.

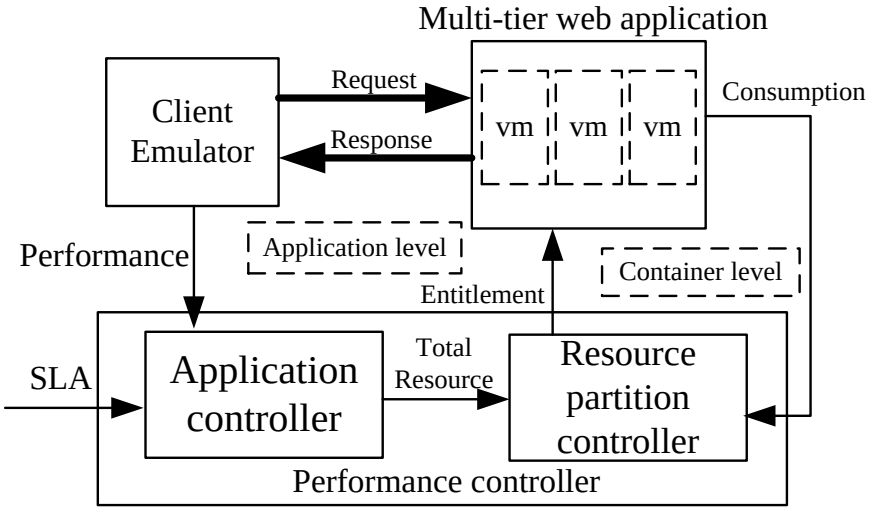


Figure 34: The architecture of our test bed.

1. We model an open multi-tier web application as a tandem queue, which consists of several queueing systems in series. The Round Trip Time (RTT) for each tier is estimated through an M/G/1/PS queue. We show that under a given total CPU budget, the optimal partition which minimizes the RTT, can be calculated based on offline models for open workloads and online measurement. Such a partition scheme can be used by a PaaS provider to economically operate the service. We also test the optimal partition method against closed/semi-open multi-tier web applications, which shows that the optimal partition scheme is robust enough w.r.t. the workload models. Although closed/semi-open multi-tier web application cannot be modeled as M/G/1/PS queue, the optimal partition scheme still outperforms other approaches described in previous work, e.g., "Equal Shares" and "Equal Utilization".

2. We propose a two-level control architecture by leveraging hierarchical resource management for optimal resource allocation for a multi-tier web application. On the application level, an adaptive feedback controller is applied to decide the total resource demands of the application in real time to maintain the RTT

threshold upon varying workload. On the container level, an optimal controller partitions the total resource budget among the multiple tiers that can minimize the end-to-end response time. The two controllers work together to achieve the optimal resource allocation for the application. Through experiments, we show that, the performance controller with the optimal partition scheme can achieve at least the same performance as a couple of other schemes, e.g., 'Equal Shares" and "Equal Utilization", while using up to 20% fewer resources. We also test the two level performance controller with both open and closed multi-tier web applications to show the robustness of our approach.

5.4 Experimental settings

5.4.1 Test Bed

We use RUBiS [14] as the benchmark application. It is an online auction benchmark comprised of a front-end Apache Web server, a Tomcat application server, and a back-end MySQL database server. There are 26 transaction types in RUBiS. The types of the next request generated by the workload generator are defined by a state transition matrix that specifies the probability to go from one transaction to another. In our experiments, we use "Browsing mix" workload that has 10 transaction types, e.g., Home, Browse, ViewItem. These transactions have different resource demands.

We assume that each of the three tiers of the application is hosted in one Xen virtual machine [27]. Our test bed consists of three physical machines as shown in Figure 34. One of them is used for hosting the three VMs, one for the client emulator and the last one for running the controller. Each machine is an Intel Pentium 4 1.80GHz, 1 GB RAM PC with Gigabit Ethernet connected to the switch. All machines run Linux kernel 2.6.16.29. The hosting machine runs Xen 3.0.3. We use Apache v2.0.54 as the web server, Jakarta Tomcat v5.0.28 as the application server and MySQL v3.23.58 as the database server.

5.4.2 Closed, open and semi-open workload generators

A workload generator is needed to generate requests to emulate the behavior of clients. To evaluate the robustness of our approach, we create three types of workload generators that can represent different client behaviors [104].

A workload generator is called closed if new requests are triggered only after previous requests have been completed or timeout. The original RUBiS implementation is a standard closed-loop client emulator. The client generates new request(interaction) after it receives the response of the previous request and waits for an exponentially distributed "think time". Then each client has three possible statuses: (a) waiting in queue; (b) being served by server or (c) "thinking" for some amount of time. The action sequence of each session follows these steps: (a) to (b), (b) to (c) and (c) back to (a). Then the intensity of the workload depends on the number of the clients and the think time. The number of the clients is also called multiprogramming level (MPL). The think time is an exponentially distributed random variable. The mean or expected value of the think time is 3.5s. Therefore, different MPLs represent different intensities of the workload, or request rate.

A workload generator is called open if new requests are generated independently of completion of previous requests. We modify the source code of original RUBiS workload generator to emulate open clients where the number of requests follows the Poisson distribution with a parameter of the arrival rate.

A workload generator is called semi-open if after a client receives a response for the previous requests, it will stay and make a follow up request with some probability p and will leave system with probability $1 - p$. In the extreme cases with very small or large p, the semi-open workload generator resembles an open or a closed one respectively. We also modify the source code of original RUBiS workload generator to be a semi-open client emulator. The intensity of the workload is determined by the arrival rates as well as the probability p. In order to get a balance between closed

and open models, we set the default probability $p = 0.5$ in our experiments.

It is worthwhile to note that, neither the open system model nor the closed system model is entirely realistic [104]. The client behavior in many multi-tier web applications is best represented using an "in-between" system model, i.e., the semi-open model. In the rest of the chapter, we call the RUBiS web application using closed, open and semi-open workload generators as closed, open and semi-open RUBiS web applications respectively.

5.5 Resource Partition Controller

For a given end-to-end performance target such as round trip time of request threshold, there is one optimal resource allocation to the tiers of a multi-tier web application that can minimize the total resource allocation. And, given certain amount of resource available to the application, there exists also one optimal resource allocation to the tiers that can minimize the end-to-end performance, which is studied in this subsection based on queueing theory.

5.5.1 Modeling multi-tier web application with open workload

5.5.1.1 Multi-tier web application

In our model, we consider a multi-tier web application consisting of multiple tiers. We assume that each tier runs on a separate virtual machine. We consider a workload with Ω transaction types. If we define the intensity of the workload for the transaction type ω as λ_ω, then the intensity of the workload can be defined as a vector $(\lambda_1, ..., \lambda_\Omega)$. We also define the aggregate rate of the transaction as $\lambda = \sum_{\omega=1}^{\Omega} \lambda_\omega$.

5.5.1.2 End-to-end performance

The Round Trip Time (RTT) of a multi-tier web application with open workload can be calculated by aggregating the resident times over all resources (e.g., CPU, disk) across all the tiers. The resident time on each tier is composed of two parts, i.e.,

the resident time on CPU resource and that on non-CPU resources. We assume that non-CPU resources are adequately provisioned and hence the effect of contention for these resources on the response time (i.e., the queueing delay) is negligible.

Since processor sharing (PS) approximates round-robin scheduling with small quantum size and negligible overhead, it is representative of scheduling policies in current commodity operating systems [79]. Moreover, a Poisson process is a good approximation of requests for open workload. We model CPU in each tier as an M/G/1/PS queue.

We use T_{cpu} to denote the total resident time on CPU across all the tiers. According to the queueing theory, for M/G/1/PS queue, the CPU resident time in the n-th tier is represented by

$$T_n = \frac{r_n}{\lambda(1 - r_n)}$$

where r_n is the CPU utilization of tier n. Note that CPU utilization in the above equation is the ratio between the virtual machine's CPU consumption and its effective CPU capacity, then we have

$$T_{cpu} = \sum_{n=1}^{N} T_n = \sum_{n=1}^{N} \frac{r_n}{\lambda(1 - r_n)} = \sum_{n=1}^{N} \frac{c_n}{\lambda(u_n - c_n)}$$

We use T_{others} to denote the total resident time spent on all non-CPU resources. We use α_ω to represent service times of transaction type ω on all non-CPU resources of all tiers on the execution path of that transaction type. Then the mean resident time on non-CPU resources can be approximated by the weighted sum of each transaction type's service time.

$$T_{others} = \sum_{\omega=1}^{\Omega} \alpha_\omega \frac{\lambda_\omega}{\lambda}$$

Assume that there is an additive relationship between time spent on CPU and non-CPU resources, by combining T_{cpu} and T_{others}, we have

$$RTT = T_{cpu} + T_{others} = \frac{1}{\lambda}(\sum_{n=1}^{N} \frac{c_n}{u_n - c_n} + \sum_{\omega=1}^{\Omega} \alpha_\omega \lambda_\omega)$$

As stated in Section 5.4, the types of the next request generated by the virtual clients are defined by a state transition matrix. Given a state transition matrix that describes the transition relationship among the 10 transaction types in "Browsing mix" of RUBiS, we can assume that the share of each transaction types is constant when the experiment running time is long enough, i.e., $\frac{\lambda_\omega}{\lambda}$ is constant. For simplicity, we also assume that the average service time of non-CPU resources of each transaction type is constant since the effect of contention for these resources on the response time (i.e., the queueing delay) is negligible, i.e., α_ω is constant. If we can denote $\sum_{\omega=1}^{\Omega} \alpha_\omega \frac{\lambda_\omega}{\lambda} = \beta$ as a constant, then we have

$$RTT = T_{cpu} + T_{others} = \frac{1}{\lambda} \sum_{n=1}^{N} \frac{c_n}{u_n - c_n} + \beta$$

We can see that, the resource entitlement solution for a multi-tier web application is not unique for a given RTT target. Similarly, for a given capacity available to the application, there is an optimal solution for the resource allocation to the multiple tiers that can minimize the RTT.

5.5.2 Optimal resource partition

Assume that the total CPU resource available for the application, or the total CPU shares that the multi-tier web application provider rents, is fixed. Then we have an optimization problem defined as following.

$$\begin{aligned} \text{Minimize} \quad & RTT = \frac{1}{\lambda} \sum_{n=1}^{N} \frac{c_n}{u_n - c_n} + \beta \\ \text{s.t.} \quad & S = \sum_{n=1}^{N} u_n. \end{aligned}$$

There are N independent variables, i.e., u_n where $n = 1, ..., N$. The problem can be denoted as

$$\begin{aligned} \text{Minimize} \quad & f(u_1, u_2, ..., u_N) \\ \text{s.t.} \quad & g(u_1, u_2, ..., u_N) = 0 \end{aligned}$$

where

$$f(u_1, u_2, ..., u_N) = \frac{1}{\lambda} \sum_{n=1}^{N} \frac{c_n}{u_n - c_n} + \beta$$

and

$$g(u_1, u_2, ..., u_N) = \sum_{n=1}^{N} u_n - S = 0$$

Then we have the Lagrange function as

$$\begin{aligned} \Gamma(u_1, u_2, ..., u_N) &= f(u_1, u_2, ..., u_N) + \gamma g(u_1, u_2, ..., u_N) \\ &= \frac{1}{\lambda} \sum_{n=1}^{N} \frac{c_n}{u_n - c_n} + \beta + \gamma(\sum_{n=1}^{N} u_n - S). \end{aligned}$$

where γ is a Lagrange multiplier and the partial derivative equations are

$$\begin{aligned} \frac{\partial \Gamma}{\partial u_n} &= 0 = -\frac{1}{\lambda} \frac{c_n}{(u_n - c_n)^2} + \gamma = 0, n = 1, 2, ..., N \\ \frac{\partial \Gamma}{\partial \gamma} &= 0 = \sum_{n=1}^{N} u_n - S \end{aligned}$$

By solving the above equations, we can get the optimal solution

$$u_n = c_n + \frac{\sqrt{c_n}}{\sum_{n=1}^{N} \sqrt{c_n}} (S - \sum_{n=1}^{N} c_n)$$

Then the controller to implement the optimal solution in every control interval k is

$$u_n(k+1) = c_n(k) + \frac{\sqrt{c_n(k)}}{\sum_{n=1}^{N} \sqrt{c_n(k)}} (S(k) - \sum_{n=1}^{N} c_n(k))$$

From the solution we can see that, the optimal resource budget for each tier is composed of two parts: the first part is equal to the actual resource consumption of that tier, and the second part is a weighted share of the remaining budget (i.e., $S(k) - \sum_{n=1}^{N} c_n(k)$). It is worthwhile to note that, the optimal solution depends on the CPU consumption of the tiers in the last interval, but not on β, the effect of the non-CPU resources. Moreover, the optimal solution does not necessarily result in the equal utilization of the tiers if the resource consumptions are different from each other, as we will show in the next subsection.

5.6 Evaluation of Resource Partition Controller

In this subsection, we evaluate the optimal partition controller through comparison with two other approaches: "Equal Utilization" and "Equal Shares".

5.6.1 Different resource partition schemes and experimental settings

Optimal : With the scheme "Optimal", the resource is allocated to the tiers according to optimal resource partition scheme as described in the previous subsection.

Equal Utilization : With the scheme "Equal Utilization", the resource is allocated to the tiers such that they have the same utilization, i.e., for $n = 1, 2, .., N$, r_n are the same.

Equal Shares : With the naive scheme of "Equal Shares", the resource is shared equally by all the tiers, i.e., for $n = 1, 2, .., N$, u_n are the same.

In our experiments, we fix the total CPU share for partition at 0.5 CPU. We then vary the workload rate from 15req/s to 50req/s. For each workload style, each workload rate and each partition scheme, we run one experiment for 900s. For open workload generator, we change the arrival rate between 15req/s and 50req/s. For closed one, we change the MPL between 52 and 175 such that the average request rate is varied between 15req/s and 50req/s [2] with the response time of the requests much less than the think time [121]. For semi-open one, we change the arrival rate between 7.5req/s and 25req/s. Since the default probability $p = 0.5$, the average request rate is 15req/s to 50req/s [3] since the response times of the requests are much less than the think time [121]. Although the optimal solution is derived based on an open queueing model, it would be interesting to evaluate how well it works for closed

[2] When the response time of the requests are much less than the think time and the system is not saturated, the request rate can be simply calculated as $\frac{MPL}{ThinkTime}$. For example, given $MPL = 52$ and $ThinkTime = 3.5s$, the request rate is around 15req/s according to queueing theory.

[3] When the response time of the requests are much less than the think time and the system is not saturated, the request rate can be simply calculated as $\frac{\lambda}{1-p}$. For example, given $\lambda = 7.5$ and $p = 0.5$, the request rate is around 15req/s according to queueing theory.

or semi-open workloads.

5.6.2 Experimental results

We denote "Optimal", "Equal Utilization" and "Equal Shares" as "Opt", "Util" and "Shares", respectively. Figures 35-37 show the mean round trip time resulted from the experiments for the three approaches, for the three workloads, "closed", "open", or "semi-open", when the rate can vary between 15 and 50req/sec.

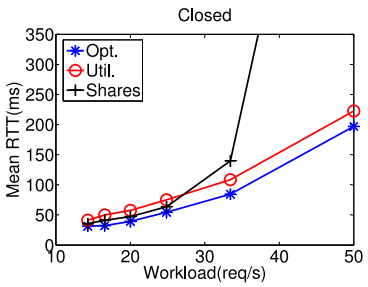


Figure 35: Mean RTT for Closed workload

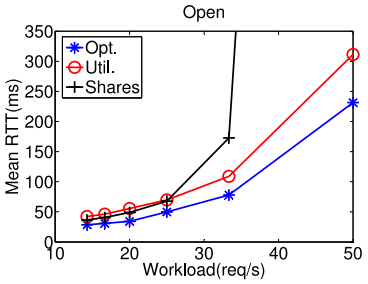


Figure 36: Mean RTT for Open workload

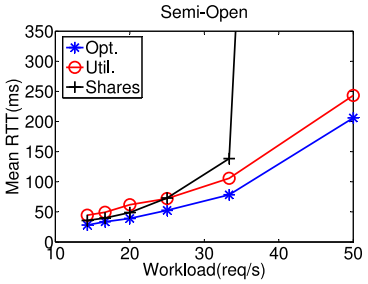


Figure 37: Mean RTT for Semi-Open workload

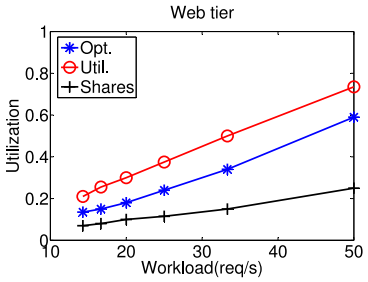


Figure 38: Web tier with Open workload

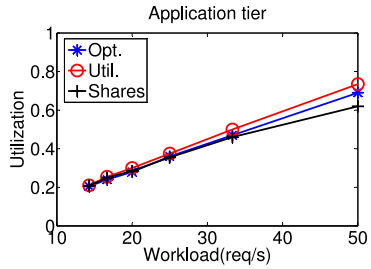


Figure 39: Application tier with Open workload

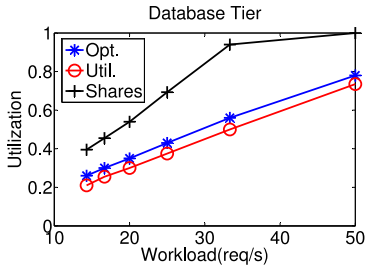


Figure 40: Database tier with Open workload

We have several observations. First, the "Opt" approach outperforms the other two approaches. On average, with the same total CPU shares, our method "Opt" can achieve about 20% shorter round trip time than the other two methods. Note that, this better performance can be achieved under different workload intensities (from 15 req/s to 50 req/s) as well as under different workload type (open, close, semi-open). This proves the robustness of our optimal partition scheme. Second, as we can see, the performance of Semi-open is in between that of Open and Closed. This validates the

previous results that different workload type will demonstrate different performance as reported in [104] and [121]. Finally, Figures 35-37 show the relationship between the response time and the workload, or equivalently the relationship between the response time and the resource utilization as the total CPU allocation is the same, and the CPU consumption is almost the same with the different workload models. The response time is a nonlinear, but monotonically increasing function of CPU utilization. Note that the change of utilization can result from both workload variance and dynamic resource entitlement.

We zoom in the application with open workload generator. Figures 38-40 show the resource utilization levels of the three tiers of the application with open workload generator, when the resource is under control of the three approaches respectively. We can also have several observations. (1)As we expect, the utilization of the three tiers for the "Util" partition scheme is always the same when the workload varies. This is because the "Util" partition scheme tries to keep the same utilization level for all the three tiers. (2) When the application is under control of the "Opt" controller, the utilization of the web tier is overall lower than that resulted from "Util" controller, while that of the DB tier is overall higher. This implies that, compared with "Util" controller, relatively less CPU is allocated to the DB tier, while more CPU is provided to the Web tier. (3) The CPU resource is equally shared by the three tiers when the application is under control of "Shares" controller. It results at further lower utilization at Web tier, the higher utilization at the DB tier than those from the other two controllers. When workload is high, e.g., 50req/s, "Equal Shares" has very long round trip time, because of the extremely high CPU utilization of the DB tier as shown in Figure 40.

5.6.3 Discussion

The solution that is proposed in the previous two sections can be generally used to address the optimal partition problem for the multi-tier web application provider. It is worthwhile to note that the optimal solution does not depend (directly) on the workload parameters such as the workload intensities, the workload transaction types, and the service times that are usually challenging to be derived or measured. Instead, it depends on the CPU consumption of the individual tiers, which are readily available in standard systems with non-intrusive measurement. Moreover, the solution is independent of the service time on all non-CPU resources (with the general assumption that they are not bottleneck). In the future, we will use more complicated model, e.g., layered queueing model [69], to consider other system resources (e.g., disk and network).

From the experiments, although M/G/1/PS model is a good approximation for the average behavior of the open RUBiS web application, it is interesting to find out that the "Opt" partition scheme still outperforms the other methods for closed and semi-open workload styles as well. As a piece of future work, we are to explore further on how well this approximation can be to improve the resource efficiency.

5.7 Application Controller Design

We describe how to design an application level feedback controller in this subsection. For the multi-tier web application, the application controller design objective can be formalized as a constraint optimization problem, i.e., to minimize the total resource consumption with constraints on the reference response time Ref.

$$\text{Minimize} \quad S = \sum_{n=1}^{N} u_n$$

$$\text{s.t.} \quad RTT \leq Ref$$

However, there are several issues that make it challenging to solve the optimization problem directly. First, a relationship model between S and RTT needs to be built. Second, there is always in-accuracy with the model. And third, the workload can vary along the time.

In our approach, we solve the problem in two steps, or two levels. In the first step, we use system identification method to build the relationship model. In the second step, a feedback controller is applied to find the minimal total amount of CPU resource that can meet the end-to-end response time threshold. We call it the application controller. Due to the feedback property, the feedback controller is able to overcome the model in-accuracy as well as the time-varying workload.

5.7.1 System Identification

As stated in related work [118], the relationship between the resource entitlement and response time is nonlinear and depends on the variation of workloads. However, we can still assume that the relationship can be estimated by a linear function in the neighborhood of an operating point. We adopt autoregressive-moving-average (ARMA) model [97] to represent this relationship.

We run system identification experiment to determine the relationship between the mean RTT and the total CPU allocation to all the three tiers. In each experiment with workload rate λ, the total CPU is randomly varied in [0.20CPU, 0.80CPU]. The mean RTT sampling interval is fixed at 90 seconds. Each experiment runs 100 intervals, i.e., 9000 seconds. The workload rate λ varies from 10 to 25 requests per second. The experiment is repeated for each rate. We use the first 50 samples to train the model and then use the second 50 samples to evaluate the model.

We find that, if we take the mean RTT as output and CPU shares as input in the ARMA model, there is no good fit model. However, if we take the reverse of mean RTT as output and CPU share as input in the ARMA model, there exist good

fit models. Assume that the mean RTT at the K-th interval is $RTT(K)$, we define $y(K) = 1/RTT(K)$. We define the operating point of $y(K)$ as y_0 and the CPU share as S_0, define $\Delta y(K) = y(K) - y_0$, and $\Delta S(K) = S(K) - S_0$. We choose the following ARMA model to represent the dynamic relation between $\Delta y(K)$ and $\Delta S(K)$.

$$\Delta y(K) = \sum_{i=1}^{n} a_i \Delta y(K-i) + \sum_{j=1}^{m} b_j \Delta S(K-j).$$

where the parameters a_i, b_j, the orders n and m characterize the dynamic behavior of the system. For convenience, we refer to such a model as "ARXnm" in the following discussion. The model above is estimated offline using least-squares based methods in the Matlab System ID Toolbox [9] to fit the input-output data collected from the experiments. The models are evaluated using the R^2 metric defined in Matlab as a goodness-of-fit measure. In general, the R^2 value indicates the percentage of variation in the output captured by the model.

From the data in Table 17, we can find that a simple model ARX01 can fit the input-output data well enough, although the ARX11 model has marginally better fitting numbers. This is reasonable, given that, the modeling and control interval is much longer than the queueing time, and the queueing process is the main resource of the dynamics in the system. Figure 41 demonstrates how the model works when rate is 20req/s. However, we also find that, the parameters of the models vary along the workload, which implies that a controller with fixed parameters may not work in the whole range of operation conditions.

Table 17: R^2 values for ARX models

Rate(req/s)	10	15	20	25
R^2 (ARX01)	0.9370	0.9123	0.9015	0.8687
R^2 (ARX11)	0.9410	0.9282	0.9265	0.8935

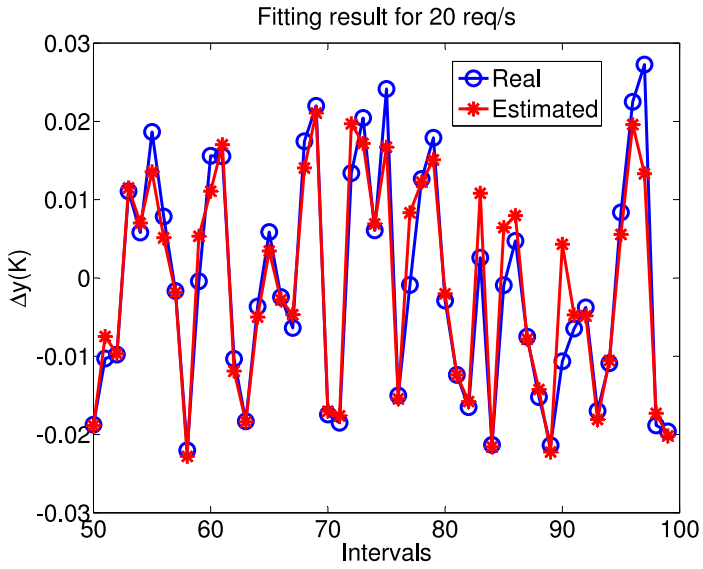


Figure 41: Fitting result for 20req/s

5.7.2 Controller Design

In our experiments, we choose ARX01 as our model, which implies that

$$\frac{\Delta y(z)}{\Delta S(z)} = az^{-1}$$

where $y(z)$ and $S(z)$ are the z-transform of y and S. We use an proportional-integral (PI) controller whose transfer function can be described as

$$\frac{\Delta S(z)}{e(z)} = k_p + \frac{k_i z}{z-1}$$

where k_p and k_i are the proportional and integral gains of the controller, respectively and $e(z)$ is the z-transform of the error. Then we have the closed model transfer function:

$$C(z) = \frac{G(z)}{1+G(z)} = \frac{(k_i+k_p)az - k_p a}{z^2 + ((k_i+k_p)a - 1)z - k_p a}$$

During our experiments, the parameter a of the model above is identified online through a recursive least square method to fit the nonlinear and time-varying behavior of the system. Then we use the Root Locus [61] method to design the controller parameters k_p and k_i so that the setting time of the controller is within three steps and the overshoot of the controller is within 10% for a step response.

5.8 Performance guarantee through adaptive PI control

We evaluate the two-level performance controller that integrates application controller with different resource partition schemes in this subsection. We first show how the different resource partition schemes can be formulated as different constraint optimization problems. Then we describe the time-varying workload that we use for evaluation in detail. Finally, we present the evaluation results using different SLAs.

5.8.1 Comparison of performance controller based on different resource partition schemes

the application's performance controller with "Optimal" partition scheme solves the constraint optimization problem as shown in (1), (2) and (3).

$$\text{Minimize} \quad S = \sum_{n=1}^{N} u_n \tag{1}$$

$$\text{s.t.} \quad RTT \leq Ref \tag{2}$$

$$RTT = \frac{1}{\lambda} \sum_{n=1}^{N} \frac{c_n}{u_n - c_n} + \beta \tag{3}$$

According to the definitions, the application's performance controller with "Equal Utilization" solves the following constraint optimization problem:

$$\text{Minimize} \quad S = \sum_{n=1}^{N} u_n \tag{4}$$

$$\text{s.t.} \quad RTT \leq Ref \tag{5}$$

$$r = \frac{c_1}{u_1} = \frac{c_2}{u_2} = ... = \frac{c_N}{u_N} \tag{6}$$

Similarly, the application's performance controller with "Equal Shares" solves the following constraint optimization problem:

$$\text{Minimize} \quad S = \sum_{n=1}^{N} u_n \tag{7}$$

$$\text{s.t.} \quad RTT \leq Ref \tag{8}$$

$$u = u_1 = u_2 = ... = u_N \tag{9}$$

Comparing the different constraint optimization problems, we can see that, all the three problems share the same objective and one constraint $RTT \leq Ref$. However, "Optimal" leverages a queueing model (3). "Equal Utilization" appends the constraint (6) while "Equal Shares" has the constraint (9). "Equal Utilization" has been used in [12] for the benefits of simple communication between an application controller and container controllers. Although it does differentiate the resource partition to the three tiers using the relative metrics, e.g., the resource utilization, it is still not optimal as there is no guarantee that constraint (6) will still hold in the optimal solution.

5.8.2 Time-varying workload for evaluation

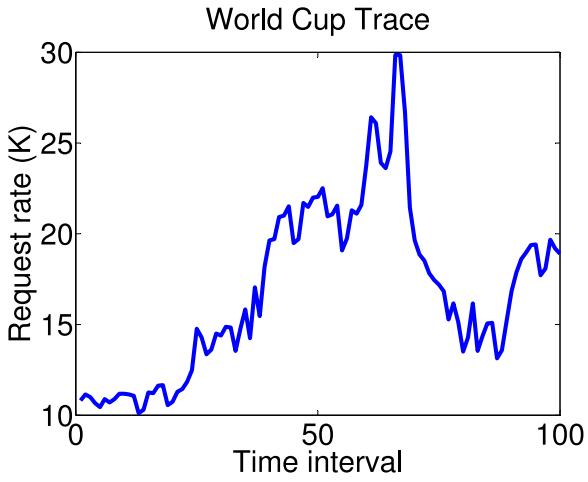


Figure 42: World Cup Trace

We use a real workload trace as shown in Figure 42 to evaluate the application's performance under the application controller with different resource partition schemes. The workload trace is generated based on the Web traces from the 1998 World Cup site [23]. We extract the "request rate" metric from the Web trace and scale it down to fit our experiment environment. For example, we use 10req/s to mimic 10k request rate and use 20req/s to mimic 20k request rate. The initial CPU shares are set to 50. Each experiment runs 9000 seconds. To evaluate the controllers,

we run a set of experiments with different styles of workloads (open or closed), and different resource partition schemes ("Opt" or "Util") on the container level. Thus, there will be four cases in a set of experiments. In our experiments, we try two SLAs where the threshold for the mean round trip time is 35ms and 200ms, respectively.

5.8.3 Setting point for the mean round trip time is 35ms

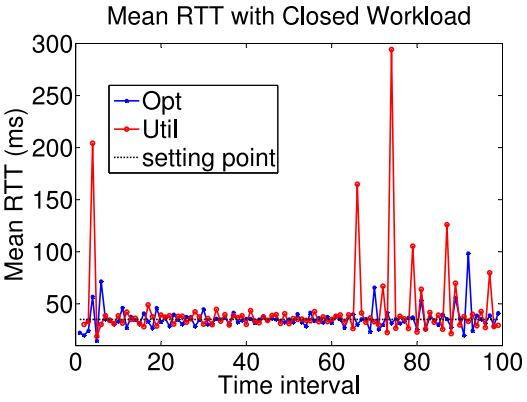


Figure 43: Mean *RTT* with Closed Workload

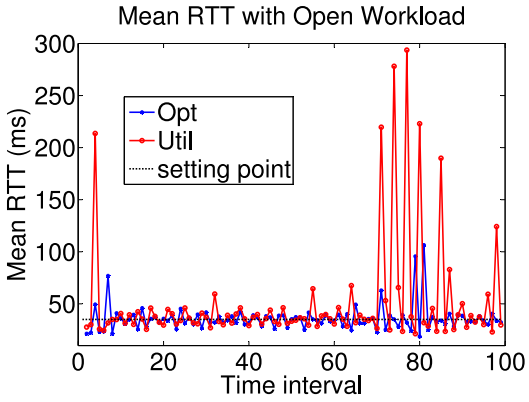


Figure 44: Mean *RTT* with Open Workload

Figures 43 and 44 show the experimental results when the setting point for the mean round trip time is 35ms. Table 18 shows more statistics of the four cases: the mean, the standard deviation, the 50/90/95 percentiles of the response times of the individual requests, the throughput (req/sec), the total CPU entitlement and the total CPU consumption. The samples for the statistics are between the 10th interval and the 70th interval, where there are no obvious overshoots of the response times and the mean *RTT* is maintained much closer to the reference values. We make the following observations: (1) up to 20% less amount of CPU resource is provisioned to the application in the cases with "Opt" controller than the application in the cases with "Util" controller. This validates the economical property of our controller. (2) The cases with "Opt" controller have lower percentile response times, compared with those with "Util" controller. This validates the robust property of our controller.

Table 18: Steady-state performance when setting RTT=35ms

	Response Time (ms)				Thr	CPU Resource	
	Mean (std)	50p	90p	95p		Ent	Con
Opt (close)	36 (52)	12	103	139	18.8	0.54	0.16
Util (close)	36 (56)	12	115	153	18.9	0.64	0.16
Opt (open)	37(53)	12	104	142	19.1	0.57	0.16
Util (open)	37 (58)	12	115	154	19.1	0.67	0.16

5.8.4 Setting point for the mean round trip time is 200ms

The application with closed workload is a "self-tuning" system. According to Little's law, we can derive that the relation between the throughput and RTT for the application with closed workload can be approximated as

$$Throughput = \frac{MPL}{RTT + ThinkTime}$$

The above model implies that the throughput should vary along with the RTT. However, as shown in Table 18, the throughput of the applications with the open or closed workloads is almost the same. This is because that, in the above experiments, the default think time has a mean of 3.5s and the setting point of RTT=35ms, that is, $RTT << ThinkTime$. So the throughput is almost not affected by the RTT, and the closed system works very similar as the open one.

To see the difference of the closed and open systems, we reduce the think time for the closed system to a mean of 350ms and increase the setting point of RTT to 200ms. We run the same set of experiments. The RTT during each interval and the steady state performance are shown in Figures 45-46 and Table 19. Table 19 shows that the throughput can be affected by the response time for the closed system.

We have similar observations: (1) less amount of CPU resource is provisioned to the application in the cases with "Opt" controller than the application in the cases

with "Util" controller. (2) The cases with "Opt" controller have lower percentile response times, compared with that with "Util" controller. The above observations validate the economical and robust property of our controller.

Moreover, from Figure 45, we can see that the controller works well with closed workload and the performance is well tracked as shown in Table 19. However, the performance seems out of control for the open system as shown in Figure 46. Upon sharp changes of the workload, there are very long transient processes before the response time converges. This can be due to the high utilization of the applications, when the response time is very sensitive to changes of the resources. It implies that, the parameters of the adaptive PI controllers have to be carefully tuned for the open system when the utilization is pushed high. As one more piece of work, we are working with a more robust adaptive controller to fit different types of workloads in a wider operation region.

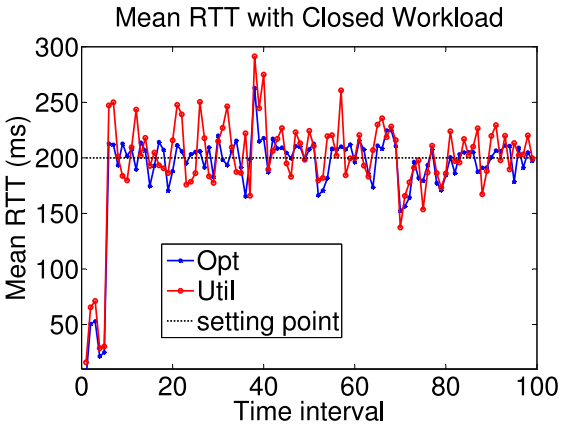


Figure 45: Mean *RTT* with Closed Workload

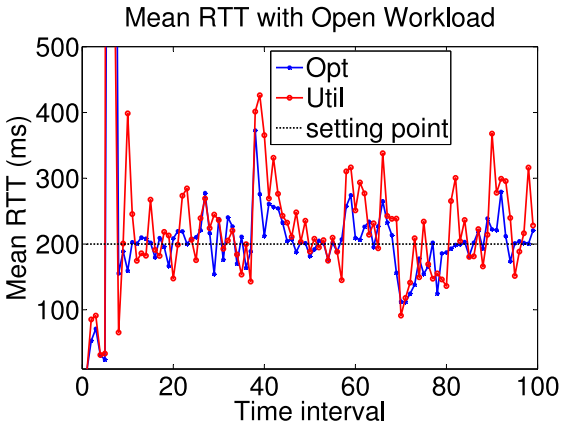


Figure 46: Mean *RTT* with Open Workload

5.9 Summary

In this chapter, ERController is proposed to enhance a classical resource allocation controller by leveraging globally-optimizing multi-level resource allocation based on hierarchical resource management to achieve the highest resource utilization for a

Table 19: Steady-state performance when setting RTT=200ms

	Response Time (ms)				Thr	CPU Resource	
	Mean (std)	50p	90p	95p		Ent	Con
Opt (close)	205(214)	140	461	608	28	0.26	0.213
Util (close)	205(229)	142	472	617	28	0.29	0.214
Opt (open)	225(363)	136	475	661	42	0.43	0.301
Util (open)	230(399)	136	481	686	42	0.47	0.313

PaaS provider. The strengths and weaknesses of ERController compared with a classical resource allocation controller are summarized as below:

5.9.1 Strengths

Under a step-wise SLA, ERController is able to achieve the highest resource utilization for a PaaS provider when a multi-tier web application could be modeled as a tandem queue. However, since a classical resource allocation controller assumes "Equal Utilization" or "Equal Shares", which is not optimal for resource utilization, ERController will show its strength to obtain the highest resource utilization so that the profit for a PaaS provider could be maximized.

5.9.2 Weaknesses and future work

ERController will show its limitation under a non-step-wise SLA because it would be very difficult to determine the setting point for the application controller under a non-step-wise SLA. In the future, we need to look into more general SLA cost function and explore how it can affect the application of the derived results.

ERController will show its limitation when the SLA is affected by more than one resources and there is no additive relationship between time spent in CPU resource and non-CPU resources because we assume that there is an additive relationship between time spent in CPU resource and non-CPU resources when we derive the

optimal resource partition for the resource partition controller. However, if we treat networking as a non-CPU resource (e.g., data transfer between two tiers), then data transfer time can overlap with CPU time. Hence, we need a better model in the future for the complicated dependencies among different resources.

CHAPTER VI

RELATED WORK

6.1 Overview

Monitoring, modeling and management of performance and resources for applications in the Cloud is always one of the hottest topics in Cloud computing research area. In this chapter, we summarize and compare the work related to our thesis according to three aspects, i.e., admission control, critical resource identification and resource allocation. Specifically, we demonstrate the advantages after we enhance the classical control-based approaches by leveraging decision theory, statistical machine learning and hierarchical resource management.

6.2 Admission control

6.2.1 Classical control-based approaches

We survey previous work in admission control for application's performance in two categories: general admission control and DBMS admission control.

General admission control Most of the classical techniques are based on rejecting incoming work to a service by refusing to accept new requests. For example, Schroeder *et al.* [103] dynamically adjust the lowest MPL that corresponds to the best application's performance. Welsh and Culler propose an adaptive approach to overload control in the context of the SEDA Web server [120] to control the 90th-percentile response time of requests. Popovici and Wilkes [98] use simulation to develop scheduling policies to make profits in the uncertain resource environment. Kamra [71] designs a self-tuning admission controller for 3-tier web sites. Karlsson *et al.* [72, 73] develop a self-tuning adaptive controller for admission control in storage

systems based on online estimation of the relationship between the admitted load and the achieved performance. The admission control mechanisms in the above work are general admission control mechanisms, which can be not only used in database management systems, but also used in general applications or systems.

DBMS admission control Contrast to general admission control mechanisms which are oblivious to query types and query mixes, Q-Cop [112], QShuffler [21] and Gatekeeper [46] take into consideration the different requirement for different type of queries when admission control decisions are made in database management systems. For example, Q-Cop is a prototype system for improving admission control decisions that considers a combination of the load on the database management system, the number of concurrent queries being executed, the actual mix of queries being executed, and the expected time a user may wait for a reply before they or their browser give up (i.e., time out). Compared with the query type and query mix oblivious methods, Q-Cop makes more informed decisions about which queries to reject and as a result significantly reduces the number of requests that time out by 47% [112].

6.2.2 Our approach ActiveSLA

ActiveSLA has the advantage of [103] where the decision module dynamically tunes the best MPL as there are different optimal MPLs for different workloads. ActiveSLA also has the advantage of [112, 21] where the query type and query mix are taken into consideration.

However, ActiveSLA distinguishes itself from the above work by leveraging decision theory in two major aspects. (1) It estimates the probability for a new query to meet/miss service-level agreements before it is admitted. ActiveSLA builds a non-linear classification model to predict this probability rather than a linear regression model that is used in Q-Cop [112]. Moreover, besides query type and query mix that

are used in existing work, ActiveSLA also takes into consideration query features as well as the database-specific and system-level metrics, which further help to improve the prediction accuracy. (2) The admission control decisions made by ActiveSLA are steered by the expected profits, which are derived by the probability for a new query to meet/miss service-level agreements and the profit/penalty specified in service-level agreements. Therefore, differentiated services, which are very important in the Cloud databases, are also provided. The experimental results demonstrate that ActiveSLA is able to make admission control decisions that are both more accurate and more profit-effective (at least 20% better) than several classical methods.

6.3 Critical resource identification

6.3.1 Classical control-based approaches

We survey previous work in critical resource identification for application's performance in two categories: model construction and model management.

Model construction A performance model is crucial for critical resource identification because it connects the application's performance and the system resource metrics. Tremendous amounts of human effort has gone into application's performance modeling. A performance model may be a mental model, which is unnecessarily restrictive based on human expert experience and domain knowledge or the desire to produce an analytical model that is both reasonably accurate and computationally scalable.

Several projects propose a performance model based on low-overhead end-to-end tracing (e.g., [20, 26, 35, 102, 47]), which captures the flow (i.e., path and timing) of individual requests within and across the components of a distributed system. For example, Aguilera *et al.* [20] develop two different algorithms, i.e., RPC messages based and signal-processing based ones for inferring the dominant causal paths through a distributed system. Magpie [26] extracts the resource usage and control

path of individual requests in a distributed system and tags incoming requests with a unique identifier and associating resource usage throughout the system with that identifier. Chen *et al.* [35] describe Pinpoint, a system for locating the components in a distributed system most likely to be the cause of a fault. Sambasivan *et al.* [102] compares request flows from two executions (e.g., of two system versions or time periods) to diagnose performance changes in a distributed storage service caused by code changes, configuration modifications, and component degradations.

Some researchers propose a performance model based on collected metrics rather than communication patterns among components. For example, Urgaonkar *et al.* [115] present an analytical performance model for multi-tier Internet services based on a traditional queueing network model. Stewart *et al.* [108] leverage the nonstationarity in an application's transaction mix to build a model for estimating the mean response time. Padala *et al.* [97] propose an auto-regressive moving average model to represent the relationship between application's performance and its CPU and disk I/O resource allocations, where the model parameters are updated online using a recursive least squares method. Heo *et al.* [62] build a prototype of a joint resource control system for allocating both CPU and memory resources to co-located VMs in real time. Lu *et al.* [84] dynamically adjust the cache size for multiple request classes. Kundu *et al.* [78] build application's performance models for virtualized environments based on artificial neural networks, using several pre-selected system-level metrics.

Other researchers take into consideration of the consolidation influence for a performance model. For instance, Q-Clouds [91] ensures that the performance experienced by applications is the same as they would have achieved if there were no performance interference. Govindan *et al.* [54] develop Cuanta to manage resource contention and performance degradation. Their methods actively predict the degradation that will be observed after applications are consolidated. Then they develop methods that select the optimal workload placements to make desirable performance

and energy trade-offs. Koh *et al.* [75] study the effects of performance interference by looking at system-level workload characteristics. Mei *et al.* [89] study the effects of performance interference of network I/O applications. The above work assumes that they know the application's critical resources from domain knowledge or experience. Then they can build a static model, e.g., the linear model in [75] based on the critical resources and benchmark performance.

Model management Model management or adaptive model is also important because different applications' performance can be affected by different system-level resource metrics at different times in a virtualized environment. Moreover, the Cloud service provider's actions such as VM consolidation and migration [74] can affect the expected resource-performance relationships. Cohen *et al.* [38] use a data-driven approach to build a tree-augmented naive (TAN) Bayesian network model to learn the probabilistic relationship between the SLO state and system metrics. However, their models are built offline after an SLO violation to identify performance bottlenecks. Bodík *et al.* [30] use models to map application workload-levels to the resources (number of virtual machines) needed to satisfy SLOs for applications running in a public Cloud. Then they use hypothesis testing of prediction errors to identify degradation in the accuracy of the models.

6.3.2 Our approach vPerfGuard

vPerfGuard distinguishes itself from the above work by leveraging statistical machine learning in two major aspects.

Model construction Although some model construction approaches [108, 97, 62, 84, 78] have the potential to provide performance predictions for a single application, these models rely on a small number of metrics or control knobs that are predetermined through human expert experience and domain knowledge.

Leveraging statistical machine learning techniques, our vPerfGuard to model construction is *data-driven* (or "black-box") – learning from the rich telemetry collected from the application, VMs and the hypervisors. We argue that this approach complements the use of expert-built models (or "white-box"). Note that Bodík *et al.* [29] present a methodology for automating the identification of performance crises using a data center fingerprint, which reflects the data center state. However, their approach depends on the previous fingerprint. The more data center states they have collected, the more accurate their identification is. Compared with their approach, vPerfGuard leverages statistical learning techniques to be totally open to the new Cloud environment states that have never been seen before due to statistical learning techniques' data-driven characteristic. Our work complements theirs by helping identify the root-cause using models.

Model management Classical methods based on low-overhead end-to-end tracing (e.g., [20, 26, 35, 102, 47]) could not be applied directly into the consolidated Cloud environment due to two reasons: (1) Although their method pinpoints the critical component within an application, it is not able to pinpoint the critical resources. (2) Their method is not suitable to the Cloud environment when different applications' performance can be affected by different system-level metrics at different times in a virtualized environment.

Compared with their method, vPerfGuard leverages statistical machine learning techniques for the virtual machine and the hypervisor metrics. The metrics which come from our adaptive models point to not only the critical component within an application but also the critical resources. Moreover, due to the enhancement by leveraging statistical hypothesis tests, vPerfGuard automatically detects the need to update the performance model when it no longer accurately captures the relationship

between performance and system resources. Note that the adaptive model management method in [30] is limited to the resources of the number of virtual machines. Thus, the metrics that are considered in their model management method is only a subset of the thousands of virtual machine and host metrics that are considered in our vPerfGuard.

6.4 Resource allocation

6.4.1 Classical control-based approaches

We survey previous work in resource allocation in two categories: single-component application and multi-component application.

Single-component application Resource allocation controller can be used to achieve the lowest resource cost while guaranteeing service-level compliance. For a single-component application, a feedback controller can be used to allocate resources [42, 80, 119] based on control theory which offers a principled way for designing feedback loops to deal with unpredictable changes, uncertainties, and disturbances in systems. The input of the controller is the difference between the measured performance metric and the reference performance metric according to service-level agreements while the output of the controller is the critical resource that affects service-level compliance. For example, if the service-level agreement is a function of round trip time and CPU is the critical resource that affects service-level compliance, then a feedback controller can be built to achieve minimal-cost rental of CPU resource (e.g., from an IaaS provider) while maintaining a sufficiently low round trip time level under the time-varying workload.

Multi-component application Compared with a single-component application, many applications are deployed in the Cloud in a totally distributed way, which makes

the dynamic and adaptive resource allocation even more challenging. For a multi-component distributed web application [81], there are also lots of special intrinsic parameters that can be tuned to improve resource utilization. By carefully exploiting those potential parameters, we are able to use the resources more efficiently. For example, some previous work implements an utilization controller for each component inside the application [118, 97]. The utilization settings are the same for all the utilization controllers in their work [118, 97]. A few workload management products such as HP global workload manager [12] maintain the utilization at a default utilization target, e.g., 75%.

Moreover, when a multi-level controller is used, the robustness and stability under different workload type and intensity become a major concern. However, most of the current work adopts either closed workload or open workload and pays little attention to whether a workload generator is closed or open. For example, Pradeep *et al.* develop an adaptive resource control system that dynamically adjusts the resource shares to applications in order to meet application-level QoS goals while achieving high resource utilization in the data center [97]. Lu *et al.* dynamically adjust the cache size for multiple request classes [84]. Krasic *et al.* [76] propose an approach called cooperative polling to ensure that all applications fairly share resources. Lu *et al.* propose scheduling scheme to satisfy the requirements of different QoS requests for access to the networked storage system [85]. The workload types used in the above work are either open or closed ones. However, as illustrated in [104], there is a vast difference in behavior between open and closed models in real-world settings.

6.4.2 Our approach ERController

ERController distinguishes itself from the above work by leveraging hierarchical resource management to build an outer-level application controller and an inner-level resource partition controller.

Similar to the feedback controller that is applied to a single-component application, ERController also leverages a feedback controller on the outer-level to guarantee service-level compliance by maintaining the difference between the measured performance metric and the reference performance metric at zero. However, compared with the above work where simple "equal utilization" or "equal share" are used, the feedback controller on the outer-level coordinates the total resources through a resource partition controller on the inner-level. On the inner-level, the resource partition controller leverages queueing theory to help allocate resources for a multi-tier distributed web application. After modeling each tier with an M/G/1/PS queue, we formulate the problem as an optimization problem and also derive a solution for the problem. The experiment shows that the resource partition controller outperforms "equal utilization" or "equal share" by achieving a shorter response time with the same resource. As a result, when an outer-level application controller and an inner-level resource partition controller work collaboratively, ERController can save around 20% resource cost than classical methods but achieve the same service-level compliance.

Furthermore, compared with the existing work which uses either open or closed workload type, ERController not only exhibits how control policies are impacted by different workload types but also explains the differences in service-level compliance. These results have never been reported before in related literatures. For example, we show that there is more fluctuation for open workload type than closed workload type when the same controller is applied. We also show that the standard deviation for the response time under open workload type is larger than that for the response time under closed workload type. These useful results will be very helpful for the system management where a controller is designed and applied. Moreover, besides open workload type, our method still outperforms others with different workload styles such as closed and semi-open, which proves the robustness and stability of our controller.

CHAPTER VII

CONCLUSION AND FUTURE WORK

Although Cloud computing has seen explosive growth in recent years, dynamically monitoring, modeling and management of performance and resources is still a hard problem due to the increasing complexity of automated performance and resource management for applications in Cloud computing. This thesis leverages decision theory, statistical machine learning and hierarchical resource management to improve classical control-based approaches to automated performance and resource management. The major objective is to help Cloud service providers achieve the most profit. To that end, a set of enhanced control-based approaches are designed and implemented to address the increasing complexity of automated performance and resource management for applications in Cloud computing

7.1 Summary of thesis contributions

The thesis has three major contributions:

1. Based on decision theory, it enhances classical admission control by leveraging risk assessment to achieve the most profitable service-level compliance as shown by ActiveSLA in Chapter 3;

2. Based on statistical machine learning, it enhances a classical critical resource identification approach by leveraging statistical filtering to identify critical resources as shown by vPerfGuard in Chapter 4;

3. Based on hierarchical resource management, it enhances a classical resource allocation controller by leveraging globally-optimizing multi-level resource allocation to achieve the highest resource utilization as shown by ERController in

Chapter 5.

7.1.1 ActiveSLA: automatic control featuring risk assessment

The main contribution of ActiveSLA is to enhance a classical control-based approach by leveraging decision theory, which is concerned with identifying the profit and penalty values, uncertainties and other issues relevant in an admission decision. More specifically,

1. In order to derive the probabilities for different outcomes, such as the probabilities that a query meets or misses deadline, which are the prerequisite for decision theory, ActiveSLA leverages machine learning techniques. The machine learning model takes into consideration of many query related features as well as database and system related features and provides detailed probabilities for different outcomes.

2. In order to make the most profitable admission decision, ActiveSLA leverages the decision theory. The decision theory takes into consideration of not only probabilities for different outcomes, such as meet or miss deadline, but also the profit consequences of alternative actions and outcomes. Moreover, the potential impact of admitted query on the currently running queries as well as on the future queries are also incorporated into the decision theory.

ActiveSLA is implemented as two modules. First, a prediction module is built to estimate the probability for a new query to finish the execution before its deadline. Second, based on the predicted probability, a decision module is built to determine whether or not to admit the given query into the database system. The decision is made with the profit optimization objective, where the expected profit is derived from service-level agreements between a service provider and its clients.

ActiveSLA is evaluated by extensive real system experiments with standard database benchmark TPC-W, under different traffic patterns such as static and dynamic traffic

patterns, different DBMS settings such as read commit and serialization, and different SLAs such as gold and silver SLAs. The evaluation results demonstrate that ActiveSLA can make a more precise prediction of whether the query will meet or miss the deadline and make more profit by obtaining better service-level compliance.

For example, Figure 14 shows that the prediction error for one of the classical methods (Q-Cop) is around 25% when the deadline is 30s. However, the prediction error for our ActiveSLA is around 13%, which cuts the prediction error almost by half. For another example, Table 8 shows that the total SLA profit for one of the classical methods (Q-Cop) is 752.5. However, the total SLA profit for our ActiveSLA is 970.3, almost 29% increase of the profit compared with 752.5.

7.1.2 vPerfGuard: automatic control featuring statistical filtering

The main contribution of vPerfGuard is to enhance a classical control-based approach by leveraging statistical machine learning that describes how one or more random variables (resource metrics) are related to one or more random variables (application's performance metrics) to identify critical resources automatically and adaptively. More specifically,

1. vPerfGuard leverages statistical filtering, i.e., a two-phase metric filtering algorithm, to automatically identify the system metrics that are the most critical to the application's performance metrics. That is, first (in phase 1) selecting a small number of candidate metrics that are most strongly correlated with the application's performance; and then (in phase 2) identifying even fewer predictor metrics that can give the best prediction accuracy for a specific model from among the candidate metrics.

2. vPerfGuard leverages online change-point detection techniques to generate a performance model using the predictor metrics adaptively. When the application workload or the execution environment changes significantly, vPerfGuard

automatically updates the set of critical resource metrics it uses in its performance model and rebuilds the model at runtime by online monitoring of model prediction accuracy.

vPerfGuard is implemented as three modules - a sensor module, a model building module, and a model updating module. Once an application is running, vPerfGuard's sensor module collects two categories of system metrics - *VM metrics* from the operating systems within individual VMs and *host metrics* from the physical hosts running the hypervisors and the virtual machines. The sensor module also collects the application's performance metrics. These metrics are processed through the model building module, which will output a model with an appropriate set of predictor metrics. The model updating module will identify when the model's predictions have significantly diverged from the observed performance via hypothesis testing over the residuals of the model. If the model passes the hypothesis testing, this shows that it still accurately captures the relationship between the system resources and application's performance. However, if the model fails the hypothesis testing, it is considered unsuitable for the current situation and a new model will be constructed.

vPerfGuard is evaluated through real system experiments using a set of common benchmarks such as RUBBoS, RUBiS, TPC-W and TPC-H, under different traffic patterns such as constant and dynamic traffic patterns, in a number of common usage scenarios such as VM colocation and consolidation, and different types of resource contention such as workload, CPU, memory and disk I/O contentions in the Cloud environments. The evaluation results demonstrate that vPerfGuard makes improvement over the classical methods because vPerfGuard can (1) automatically select the critical system resource that affects service-level compliance; and (2) adaptively update the critical system resource when the application environment changes.

For example, Figure 28 shows that vPerfGuard is able to automatically and adaptively identify that the CPU resource on the ESX1 host, which hosts the web server

virtual machine is the critical system resource that affects service-level compliance from thousands of raw metrics. The example shows that, vPerfGuard will liberate a PaaS provider from the sheer amount of telemetry and the requirement for expert experience and domain knowledge, and thus save a lot of manpower. Moreover, the identified critical resource and critical component provide valuable information for constructing automatic resource allocation control systems. Based on the information, a correct control system will allocate more CPU resource to the web server virtual machine rather than the other resources or the other components. Thus, vPerfGuard prevents a control system from blindly adding useless resource, which will incur a large bill of cost.

7.1.3 ERController: automatic control featuring hierarchical resource management

The main contribution of ERController is to enhance a classical control-based approach by leveraging hierarchical resource management to help a Cloud service provider achieve the highest utilization for multi-component applications while guaranteeing service-level compliance.

Based on hierarchical control, ERController arranges resource allocation in a hierarchical tree. Each element of the hierarchy is a linked node in the tree. The total resource budget flows down the tree from superior nodes to subordinate nodes, whereas the result response time of each tier flows up the tree from subordinate to superior nodes. More specifically,

1. The higher level operates with a longer interval of planning and execution time than its immediately lower level. It is responsible to decide the total resource budget for the critical resources in order to meet service-level compliance.

2. The lower level operates with a shorter interval of planning and execution time than its immediately higher level. It is responsible to decide how to partition

the resource budget to each tier to achieve the highest resource utilization.

Our ERController is implemented as a two-level controller. On the application level, an adaptive feedback controller is applied to decide the total resource demands of an application in real time to maintain the mean round trip time (RTT) at the desired value specified in service-level agreements upon varying workload. The control model is built based on an ARMA model and system identification method. The proportional-integral (PI) controller is designed based on the control model and the Root Locus [61] method. On the container level, an optimal resource partition controller partitions the total resource budget among the multiple tiers that can minimize RTT. The controller is built using Lagrange multiplier method based on M/G/1/PS model. The two controllers work together in a hierarchical way to accomplish the optimal resource allocation that can guarantee service-level compliance and achieve the highest resource utilization.

ERController is evaluated through real system experiments using a real benchmark RUBiS, under static and dynamic traffic patterns, with three different workload models—open, closed, and semi-open in the Cloud environments. The evaluation results indicate two major advantages of ERController in comparison to previous approaches.

First, ERController is more economical than classical method such as "Equal Utilization" for maintaining a specific service-level compliance. According to the results in Table 18, the average CPU shares that are used by ERController and "Equal Utilization" are 54 and 64, respectively when we set RTT as 35ms with closed workload. This shows that ERController can save around 19% CPU resource. According to the results in Table 19, the average CPU shares that are used by ERController and "Equal Utilization" are 26 and 29, respectively when we set RTT as 200ms with closed workload. This shows that ERController can save around 12% CPU resource.

Second, ERController is more robust than classical method such as "Equal Utilization". According to the results in Table 18, the standard deviation of response time when we use ERController and "Equal Utilization" are 52ms and 56ms, respectively when we set RTT as 35ms with closed workload. According to the results in Table 19, the standard deviation of response time when we use ERController and "Equal Utilization" are 214ms and 229ms when we set RTT as 200ms with closed workload, respectively. The smaller value of the standard deviation shows that ERController is more robust than the classical methods.

7.2 Limitations of the thesis and short term future work

Although we show that classical control-based approaches to automated performance and resource management for applications in Cloud computing can be enhanced by leveraging decision theory, statistical machine learning and hierarchical resource management, there are still a number of limitations, which are left for immediate future work.

7.2.1 Improving global optimal decisions

In ActiveSLA, we have leveraged decision theory to make admission decisions for each query. In order to make global optimal decision rather than local optimal one, we include the opportunity cost concept as a way of managing multiple query decisions and shows its effectiveness through experimental studies. The determination of the exact value of the opportunity cost is an interesting problem. One direction of the extension of the thesis would be to develop new techniques or new models such as leveraging certain business considerations or workload characteristics through feedback to resolve this value.

7.2.2 Improving human readability

In vPerfGuard, we have leveraged statistical machine learning to identify critical resources. The critical resources are demonstrated by the names of the variables used in the online adaptive models. In order to help a Cloud service provider who is not familiar with the naming convention to understand and locate the metrics such as $H\_ESX1\_Web\_CPU\_Idle$, we develop GUI and also leverage the vCenter map. One direction of the extension of the thesis would be to develop new techniques or new models such as leveraging techniques from semantic web or natural language processing to develop automated reasoning systems to improve human readability of the results.

7.2.3 Modeling complicated dependencies

In ERController, we have assumed that we can identify the dependencies that can be modeled explicitly. For example, we assume that in a multi-tier web application, the dependency chain goes from web server to application server, and then goes from application server to database server. However, this may not always be apparent for components in an application. For example, it would be difficult to use a tandem queue to model the complicated dependencies between MySQL server node and MySQL data node [87, 88] in a MySQL cluster. One direction of the extension of the thesis would be to develop new techniques or new models to manage performance and resources for applications with complicated dependencies.

7.3 Long term future work

My long term future research will consistently follow the direction of enhancing classical control-based approaches to automated system management by leveraging other techniques. The potential targets would be the new challenges that arise with the advent of new hardware technology, new software framework and new computing

paradigms. More specifically, the following new application contexts sound interesting for the long-term future work.

7.3.1 Big data

Because big data [2, 55] are not suitable to work with using on-hand database management tools, the technologies being applied to them mainly include massively parallel processing (MPP). However, there are two main problems closely related to the MPP performance: automatic configuration of the parameters; and load balance of all the working nodes. A control-based approach is promising to solve these two problems in the dynamic parallel processing environments.

Timely and cost-effective processing of large datasets has become a critical ingredient for the success of many academic, government, and industrial organizations. The combination of MapReduce frameworks and Cloud computing is an attractive proposition for these organizations. However, even to run a single program in a MapReduce framework, a number of tuning parameters have to be set by users or system administrators. Users often run into performance problems because they don't know how to set these parameters, or because they don't even know that these parameters exist. With MapReduce being a relatively new technology, it is not easy to find qualified administrators.

Classical control-based approaches which are enhanced by leveraging a statistical method could be used to automate the setting of tuning parameters for MapReduce programs. A statistical method could build a what-if engine for "what is the expected performance if this set of parameters are applied?". A controller could take correct actions to configure the parameters based on the what-if engine. The whole system can go a long way towards improving the productivity of users who lack the skills to optimize programs themselves due to lack of familiarity with MapReduce or with the data being processed.

7.3.2 Parallel and distributed databases

The performance of a traditional database can be improved through parallelization of various operations, such as loading data, building indexes and evaluating queries or through the distribution of storage devices [94, 95, 96, 59, 19]. A control-based approach shows the potential to coordinate all the resources, e.g., CPU, memory and disk to work together to achieve the best performance.

Classical control-based approaches to automated quality-of-service management for parallel and distributed databases can be enhanced by leveraging hierarchical resource management, especially global load balancers. It is a challenging task to provide quality-of-service guarantees for data services in a parallel and distributed environment. The transaction workloads may have time-varying intensities and skewed patterns. And, the transaction workloads in distributed databases may not be balanced although data replication is used.

Following a hierarchical resource management scheme, a hierarchical controller which is composed of an admission controller on the outer-level and a load balancer on the inner-level could be designed. The outer-level admission controller controls the admission process of incoming transactions. The control objective is to meet the quality-of-service guarantee, e.g., the mean response time for the transactions. The inner-level load balancer collects the performance data from other nodes and balances the system-wide workload to each data replica. The outer and the inner-level controllers work in a collaborative way that the quality-of-service for a parallel and distributed database could be guaranteed.

7.3.3 Green computing

As current data centers are consuming an extraordinary high energy demand, reducing their energy consumption is a primary focus for proponents of green computing [4]. A control-based approach is not only potential for routing data to data centers where

electricity is less expensive, but also potential for combining several physical systems into virtual machines on one single, powerful system, thereby unplugging the original hardware and reducing power and cooling consumption.

Classical control-based approaches to automated power management for data centers can be enhanced by leveraging cost-sensitive adaptation. Generally, classical control-based approaches to automated power management involve two ways, i.e., migrate/route incoming workloads to data centers where electricity is less expensive and migrate/route servers to data centers where power and cooling consumption are reduced. Classical control-based approaches do not consider the cost or the power consumption during the adaptation and it is easy for them to fall into the pitfall of "consuming power to save power".

Enhanced by cost-sensitive adaptation, a classical control-based approach could take much wiser actions when it performs automated power management. For example, when it considers migrating a virtual machine which hosts a database server to another host in order to save power, it should take into the consideration of the adaptation cost due to data migration, load balance and buffer pool warmup. This prevents the controller from the scenario such as "the data migration cost is much higher than the power saving". For another example, when it considers migrating a virtual machine to another host which makes use of renewable energy such as wind turbines and solar panels, it should also take into the consideration of the adaptation cost due to the intermittent nature of renewable energy sources. This prevents the controller from the scenario such as "the cost that enables a server to use renewable energy sources is more expensive than brown energy that is produced with conventional fossil-based fuel".

APPENDIX A

QUERY FEATURES USED IN ACTIVESLA

We use "explain" command in PostgreSQL and MySQL to provide the necessary support to obtain query specific features. For example, the query cost in PostgreSQL depends mainly on 5 parameters, i.e., the number of sequential I/O (*seq_page*), the number of non-sequential I/O(*random_page*), the number of CPU tuple operations (*cpu_tuple*), the number of CPU index operations (*cpu_index*), and the number of CPU operator operations (*cpu_operator*). Each operation is assigned a unit cost by PostgreSQL, e.g., by default these unit costs are set to 1.0, 4.0, 0.01, 0.001, and 0.0025, respectively. The total estimated cost for a query plan is

$$cost = 1.0 \times seq\_page + 4.0 \times random\_page + 0.01 \times cpu\_tuple$$
$$+ 0.005 \times cpu\_index + 0.0025 \times cpu\_operator$$

With this background information, we can either directly look into the detailed query cost, or indirectly infer the values of the 5 parameters of a query in the following way. For example, in order to get the query feature *seq_page*, we call the "explain" command twice, with 1.0 and (1.0+Δ) as the unit cost for *seq_page*. Assuming the results of the two are *cost* and *cost*$'$, with

$$cost' = (1.0 + \Delta) \times seq\_page + 4.0 \times random\_page + 0.01$$
$$\times cpu\_tuple + 0.005 \times cpu\_index + 0.0025 \times cpu\_operator,$$

If Δ is a very small and the best query plan does not change, then have $seq\_page = (cost' - cost)/\Delta$.

Similarly, the "explain" command in MySQL outputs a table, in which two important columns are 'type' and 'rows'. Column 'type' shows what kind of scan to be

used, e.g., 'ALL' means a sequential scan of the whole table. Column 'rows' shows the estimation on how many rows will be returned.

APPENDIX B

DATABASE AND SYSTEM CONDITIONS USED IN ACTIVESLA

Here are database and system features that we collected.

Transaction isolation: The SQL standard defines four levels of transaction isolation, i.e., Read Uncommitted, Read Committed, Repeatable Read, and Serializable. In PostgreSQL, for example, Read Uncommitted is treated as Read Committed, while Repeatable Read is treated as Serializable. As a feature, we use a nominal variable FALSE,TRUE to denote whether Read Committed or Serializable is chosen.

Buffer cache: Each cache entry in buffer cache points to an 8KB block (sometimes called a page) of data. When a process requests a buffer, it calls BufferAlloc with what file/block it needs. If the block is already in the cache, it gets pinned and then returned. Otherwise, a new buffer must be found to hold this data. Therefore, for example, if a query is going to do a sequential scan of a table whose size is 100 pages and there are 50 pages in the buffer cache, we use 50 as the value for the feature *DB_buffer*. In order to obtain such information, for the buffer cache, we create a view called "pg_buffercache" to collect the number of pages of a table in DB buffer and we query this view to get the feature value.

System cache: Databases are designed to rely heavily on the operating system cache. The DB buffer and system cache usually work as follows: Backends that need to access tables first look for needed blocks in DB buffer. If they are already there, they are fetched right away. If not, an operating system request is made to load the requested blocks. The blocks are loaded either from the kernel disk buffer cache, or from disk. Therefore, for example, if a query is going to do a sequential

scan of a table whose size is 100 pages among which 10 pages are in the system cache, then we use 10 as the value for the feature *Sys_cache*. We describe how we obtain it PostgreSQL and MySQL. Similar methods can be used to monitor system cache content for other OS and RDBMS. It mainly contains two steps. (1) Obtain the data file location. PostgreSQL uses a directory to store all the data in all the databases. The default location is "/usr/local/pgsql/data/base". An object in PostgreSQL has its unique oid. Assume that the database and the table that we are interested in have oid d_{oid} and t_{oid}, respectively, where t_{oid} and d_{oid} can be obtained from pg_database table. Then the filename to store the data of this table turns out to be "/usr/local/pgsql/data/base/d_{oid}/t_{oid}" [1]. The default data directory for MySQL is "/var/lib/mysql". Assume that the database and the table name that we are interested in are d_{name} and t_{name}, respectively. Then the file to store the data of this table is "/var/lib/mysql/d_{name}/t_{name}.MYD" [2]. (2) We wrote a Perl script to return the portion of the files in the system cache, from which we get the number of pages of a table in system cache.

Besides cache, we also collect some other general system metrics, such as CPU stats, memory stats and disk stats. These metrics are obtained by running dstat [3] and iostat. [4]. We summarize all the features that we use in ActiveSLA and the methods by which we obtained these features in Table 20.

[1] If the table is larger than 1GB, there will be several files
[2] MyISAM storage engine
[3] http://dag.wieers.com/home-made/dstat/
[4] http://linuxcommand.org/man_pages/iostat1.html

Table 20: Features, description, and obtain methods.

Features	Description	Obtain methods
Query type and mix		
type	query type	
num_i,avg_i	number and average running time of queries of type i	Server
Query features		
seq_page	sequential I/O	PostgreSQL
rand_page	non-sequential I/O	PostgreSQL
cpu_tuple	CPU tuple operations	PostgreSQL
cpu_index	CPU index operations	PostgreSQL
cpu_operator	CPU operator operations	PostgreSQL
System features		
Transaction isolation	Read Commit(FALSE), Serializable(TRUE)	Server
DB_buffer	pages in DB buffer	PostgreSQL
Sys_cache	pages in System cache	Perl
CPU	CPU_usr,CPU_sys,CPU_idl, CPU_wai,CPU_hiq,CPU_siq	dstat
MEM	MEM_used,MEM_free MEM_buff,MEM_cach	dstat
DISK	DISK_rrqm/s, DISK_wrqm/s, DISK_r/s, DISK_w/s DISK_rsec/s, DISK_wsec/s, DISK_rq, DISK_qu,DISK_await DISK_svctm, DISK_%util	iostat

REFERENCES

[1] "dstat." http://dag.wieers.com/home-made/dstat.

[2] "http://en.wikipedia.org/wiki/big_data."

[3] "http://en.wikipedia.org/wiki/Cloud_computing."

[4] "http://en.wikipedia.org/wiki/green_computing."

[5] "Interpreting esxtop Statistics."
http://communities.vmware.com/docs/DOC-9279.

[6] "iostat.
http://linux.die.net/man/1/iostat.".

[7] "RUBBoS." http://jmob.ow2.org/rubbos.html.

[8] "Selection algorithm: en.wikipedia.org/wiki/Selection_algorithm."

[9] "System identification." http://www.mathworks.com/products/sysid/.

[10] "Control theory: http://en.wikipedia.org/wiki/control_theory."

[11] "ELBA: http://www.cc.gatech.edu/systems/projects/elba/."

[12] "HP Global Workload Manager (gWLM)."
http://mslweb.rsn.hp.com/gwlm/index.html.

[13] "IBM News: http://www-03.ibm.com/press/us/en/pressrelease/34334.wss."

[14] "RUBiS: http://rubis.ow2.org/."

[15] "TPC-H." http://www.tpc.org/tpch.

[16] "VMware vFabric Hyperic." http://www.vmware.com/products/application-platform/vfabric-hyperic/.

[17] "VMware vSphere www.vmware.com/products/vsphere/overview.html.".

[18] "Windows Hyper-V Server." www.microsoft.com/hyper-v-server/.

[19] Achyutuni, K. J., Omiecinski, E., and Navathe, S. B., "Two techniques for on-line index modification in shared nothing parallel databases," in *Proc. of SIGMOD*, 1996.

[20] Aguilera, M. K., Mogul, J. C., Wiener, J. L., Reynolds, P., and Muthitacharoen, A., "Performance debugging for distributed systems of black boxes," in *Proc. of SOSP*, 2003.

[21] Ahmad, M., Aboulanaga, A., and Babu, S., "Modeling and exploiting query interactions in database systems," in *Proc. of CIKM*, 2010.

[22] Antsaklis, P. J. and Passino, K. M., eds., *An introduction to intelligent and autonomous control*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[23] Arlitt, M. and Jin, T., "Workload characterization of the 1998 World Cup web site," *HP Tech. Rep.*, 1999.

[24] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M., "Above the clouds: A berkeley view of cloud computing," Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[25] Astrom, K. J. and Wittenmark, B., *Adaptive Control*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 1994.

[26] Barham, P., Donnelly, A., Isaacs, R., and Mortier, R., "Using magpie for request extraction and workload modelling," in *Proc. of OSDI*, 2004.

[27] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A., "Xen and the art of virtualization," in *Proc. of SOSP*, 2003.

[28] Basseville, M. and Nikiforov, I. V., *Detection of abrupt changes: theory and application*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[29] Bodik, P., Goldszmidt, M., Fox, A., Woodard, D. B., and Andersen, H., "Fingerprinting the datacenter: automated classification of performance crises," in *Proc. of EuroSys*, 2010.

[30] Bodík, P., Griffith, R., Sutton, C., Fox, A., Jordan, M., and Patterson, D., "Statistical machine learning makes automatic control practical for internet datacenters," in *Proc. of HotCloud*, 2009.

[31] Box, G. E. P. and Draper, N. R., *Empirical model-building and response surface*. New York, NY, USA: John Wiley & Sons, Inc., 1986.

[32] Box, G. E. P. and Jenkins, G., *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[33] Brogan, W. L., *Modern control theory (3rd ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1991.

[34] Carey, M. J., Jauhari, R., and Livny, M., "Priority in dbms resource scheduling," in *Proc. of VLDB*, 1989.

[35] Chen, M. Y., Accardi, A., Kiciman, E., Lloyd, J., Patterson, D., Fox, A., and Brewer, E., "Path-based faliure and evolution management," in *Proc. of NSDI*, 2004.

[36] Chi, Y., Moon, H. J., and Hacigumus, H., "iCBS: Incremental cost-based scheduling under piecewise linear SLAs," *PVLDB*, 2011.

[37] Chou, H.-T. and DeWitt, D. J., "An evaluation of buffer management strategies for relational database systems," in *Proc. of VLDB*, 1985.

[38] Cohen, I., Chase, J. S., Goldszmidt, M., Kelly, T., and Symons, J., "Correlating instrumentation data to system states: A building block for automated diagnosis and control," in *Proc. of OSDI*, 2004.

[39] Cooper, B., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.-A., Puz, N., Weaver, D., and Yernenin, R., "Pnuts: Yahoo!'s hosted data serving platform," in *Proc. of VLDB*, 2008.

[40] Cooper, B., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R., "Benchmarking cloud serving systems with ycsb," in *Proc. of SoCC*, 2010.

[41] Curino, C., Zhang, Y., Jones, E. P. C., and Madden, S., "Schism: a workload-driven approach to database replication and partitioning," *PVLDB*, 2010.

[42] Diao, Y., Gandhi, N., Hellerstein, J. L., Parekh, S., and Tilbury, D. M., "Using mimo feedback control to enforce policies for interrelated metrics with application to the apache web server," in *Proc. of NOMS*, 2002.

[43] Doyle, J., Francis, B., and Tannenbaum, A., *Feedback Control Theory*. Dover Publications, 1990.

[44] Draper, N. R. and Smith, H., *Applied regression analysis.* Wiley-Interscience, 1998.

[45] Duda, R. O., Hart, P. E., and Stork, D. G., *Pattern Classification.* New York: Wiley, 2nd ed., 2001.

[46] Elnikety, S., Nahum, E., Tracey, J., and Zwaenepoel, W., "A method for transparent admission control and request scheduling in e-commerce web sites," in *Proc. of WWW*, 2004.

[47] Erlingsson, U., Peinado, M., Peter, S., and Budiu, M., "Fay: extensible distributed tracing from kernels to clusters," in *Proc. of SOSP*, 2011.

[48] Friedman, J., Hastie, T., and Tibshirani, R., "Additive logistic regression: a statistical view of boosting," *Annals of Statistics*, vol. 28, p. 2000, 1998.

[49] Fukunaga, K., *Introduction to statistical pattern recognition (2nd ed.).* San Diego, CA, USA: Academic Press Professional, Inc., 1990.

[50] Ganapathi, A., Kuno, H., Dayal, U., Wiener, J. L., Fox, O., and Jordan, M., "Predicting multiple metrics for queries: Better decisions enabled by machine learning," in *Proc. of ICDE*, 2009.

[51] Georges, A. and Eeckhout, L., "Performance metrics for consolidated servers," in *Proc. of HPCVirt*, 2010.

[52] Gmach, D., Krompass, S., Scholz, A., Wimmer, M., and Kemper, A., "Adaptive quality of service management for enterprise services," *ACM Trans. Web*, 2008.

[53] Goodwin, P. and Wright, G., *Decision Analysis for Management Judgment [Paperback].* Chichester, Wiley, 2004.

[54] Govindan, S., Liu, J., Kansal, A., and Sivasubramaniam, A., "Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines," in *Proc. of SoCC*, 2011.

[55] Gruenheid, A., Omiecinski, E., and Mark, L., "Query optimization using column statistics in hive," in *Proc. of IDEAS*, 2011.

[56] Hacigumus, H., Iyer, B., and Mehrotra, S., "Providing database as a service," in *Proc. of ICDE*, 2002.

[57] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H., "The WEKA data mining software: An update," *SIGKDD Explorations*, vol. 11, no. 1, 2009.

[58] Hall, M. A., "Feature selection for discrete and numeric class machine learning," pp. 359–366, Morgan Kaufmann, 1999.

[59] Han, B., Omiecinski, E., Mark, L., and Liu, L., "Otpm: Failure handling in data-intensive analytical processing," in *Proc. of CollaborateCom*, 2011.

[60] Hastie, T., Tibshirani, R., and Friedman, J., *The Elements of Statistical Learning*. Springer Series in Statistics, New York, NY, USA: Springer New York Inc., 2001.

[61] Hellerstein, J., Diao, Y., Parekh, S., and D.Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[62] Heo, J., Zhu, X., Padala, P., and Wang, Z., "Memory overbooking and dynamic control of xen virtual machines in consolidated environments," in *Proc. of IM*, 2009.

[63] Irwin, D. E., Grit, L. E., and Chase, J. S., "Balancing risk and reward in a market-based task service," in *Proc. of HPDC*, 2004.

[64] Jain, R., *The Art of Computer Systems Performance Analysis*. New York: Wiley-Interscience, 1991.

[65] Jayasinghe, D., Malkowski, S., Wang, Q., Li, J., Xiong, P., and Pu, C., "Variations in performance and scalability when migrating n-tier applications to different clouds," in *Proc. of IEEE CLOUD*, 2011.

[66] Jiang, D., Ooi, B. C., Shi, L., and Wu, S., "The performance of mapreduce: an in-depth study," *Proc. VLDB Endow.*, 2010.

[67] John, G. H., Kohavi, R., and Pfleger, K., "Irrelevant features and the subset selection problem," in *Proc. of ICML*, 1994.

[68] Jones, A. T. and Mclean, C. R., "A proposed hierarchical control model for automated manufacturing systems," *Journal of Manufacturing Systems*, vol. 5, pp. 15–25, 1986.

[69] Jung, G., Joshi, K., Hiltunen, M., Schlichting, R., and Pu, C., "A cost-sensitive adaptation engine for server consolidation of multi-tier applications," in *Proc. of Middleware*, 2009.

[70] Jung, G., Swint, G., Parekh, J., Pu, C., and Sahai, A., "Detecting bottleneck in n-tier it applications through analysis," in *Proc. of DSOM*, 2006.

[71] Kamra, A., "Yaksha: A self-tuning controller for managing the performance of 3-tiered web sites," in *Proc. of IWQoS*, 2004.

[72] Karlsson, M., Karamanolis, C., and Zhu, X., "Triage: Performance isolation and differentiation for storage systems," in *Proc. of IWQoS*, 2004.

[73] Karlsson, M., Karamanolis, C., and Zhu, X., "Triage: Performance differentiation for storage systems using adaptive control," *Trans. Storage*, vol. 1, pp. 457–480, Nov. 2005.

[74] Keir, C. C., Clark, C., Fraser, K., H, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A., "Live migration of virtual machines," in *Proc. of NSDI*, 2005.

[75] Koh, Y., Knauerhase, R., Brett, P., Bowman, M., Wen, Z., and Pu, C., "An analysis of performance interference effects in virtual environments," in *Proc. of ISPASS*, 2007.

[76] Krasic, C., Saubhasik, M., Sinha, A., and Goel, A., "Fair and timely scheduling via cooperative polling," in *Proc. of Eurosys*, 2009.

[77] Kumar, C. and Rosenberg, H., "Troubleshooting Performance Related Problems in vSphere 4.1 Environments." `communities.vmware.com/docs/DOC-14905`.

[78] Kundu, S., Rangaswami, R., Dutta, K., and Zhao, M., "Application performance modeling in a virtualized environment," in *HPCA*, 2010.

[79] Lazowska, E., Zahorjan, J., Graham, G., and Sevcik, K., *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. NJ: Prentice-Hall, Inc., 1984.

[80] Liu, X., Zhu, X., Singhal, S., and Arlitt, M. F., "Adaptive entitlement control of resource containers on shared servers," in *Proc. of IM*, 2005.

[81] Liu, X., Heo, J., and Sha, L., "Adaptive control of multi-tiered web application using queueing predictor," in *Proc. of NOMS*, 2006.

[82] Liu, Z., Squillante, M. S., and Wolf, J. L., "On maximizing service-level-agreement profits," *SIGMETRICS Perform. Eval. Rev.*, vol. 29, no. 3, pp. 43–44, 2001.

[83] Ljung, L., *System identification: theory for the user*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986.

[84] Lu, Y., Abdelzaher, T., and Saxena, A., "Design, implementation, and evaluation of differentiated caching services," *IEEE Trans. on Parallel and Distributed Systems*, 2004.

[85] Lu, Y., Du, D. H.-C., Liu, C., and Zhang, X., "Qos scheduling for networked storage system," in *Proc. of ICDCS*, 2008.

[86] Luo, G., Naughton, J. F., Ellmann, C. J., and Watzke, M. W., "Toward a progress indicator for database queries," in *Proc. of SIGMOD*, 2004.

[87] Malkowski, S., Hedwig, M., Jayasinghe, D., Pu, C., and Neumann, D., "Cloudxplor: A tool for configuration planning in clouds based on empirical data," in *Proc. of SAC*, 2010.

[88] Malkowski, S., Jayasinghe, D., Hedwig, M., Park, J., Kanemasa, Y., and Pu, C., "Empirical analysis of database server scalability using an n-tier benchmark with read-intensive workload," in *Proc. of SAC*, 2010.

[89] Mei, Y., Liu, L., Pu, X., and Sivathanu, S., "Performance measurements and analysis of network i/o applications in virtualized cloud," in *Proc. of CLOUD*, 2010.

[90] Mitchell, T. M., *Machine Learning*. New York, NY, USA: McGraw-Hill, Inc., 1 ed., 1997.

[91] Nathuji, R., Kansal, A., and Ghaffarkhah, A., "Q-Clouds: managing performance interference effects for QoS-aware clouds," in *Proc. of EuroSys*, 2010.

[92] Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V., *Algorithmic Game Theory*. New York, NY, USA: Cambridge University Press, 2007.

[93] NIST, "http://csrc.nist.gov/publications/nistpubs/800-145/sp800-145.pdf."

[94] Omiecinski, E., "Parallel relational database systems," in *Modern Database Systems*, pp. 494–512, 1995.

[95] Omiecinski, E. and Shonkwiler, R., "Parallel join processing using non-clustered indexes for a shared memory multiprocessor," in *Proc. of SPDP*, 1990.

[96] Özsu, M. T. and Valduriez, P., *Principles of distributed database systems (2nd ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1999.

[97] Padala, P., Hou, K., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., and Shin, K., "Automated control of multiple virtualized resources," in *Proc. of EuroSys*, 2009.

[98] Popovici, F. I. and Wilkes, J., "Profitable services in an uncertain world," in *Proc. of SC*, 2005.

[99] Pu, C. and Swint, G. S., "Dsl weaving for distributed information flow systems," in *Proc. of APWeb*, 2005.

[100] Refaeilzadeh, P., Tang, L., and Liun, H., "Cross validation," *Encyclopedia of Database Systems*, 2009.

[101] Rich, E. and Knight, K., *Artificial Intelligence*. McGraw-Hill Higher Education, 2nd ed., 1990.

[102] Sambasivan, R. R., Zheng, A. X., De Rosa, M., Krevat, E., Whitman, S., Stroucken, M., Wang, W., Xu, L., and Ganger, G. R., "Diagnosing performance changes by comparing request flows," in *Proc. of NSDI*, 2011.

[103] Schroeder, B., Harchol-Balter, M., Iyengar, A., Nahum, E. M., and Wierman, A., "How to determine a good multi-programming level for external scheduling," in *Proc. of ICDE*, 2006.

[104] Schroeder, B., Wierman, A., and Harchol-Balter, M., "Open versus closed: a cautionary tale," in *Proc. of NSDI*, 2006.

[105] Shivam, P., Demberel, A., Gunda, P., Irwin, D., Grit, L., Yumerefendi, A., Babu, S., and Chase, J., "Automated and on-demand provisioning of virtual machines for database applications," in *Proc. of SIGMOD*, 2007.

[106] Silver, E. A., Pyke, D. F., and Peterson, R., *Inventory Management and Production Planning and Scheduling*. Wiley, third ed., 1998.

[107] Stengel, R. F., *Optimal Control and Estimation*. Courier Dover Publications, 1994.

[108] Stewart, C., Kelly, T., and Zhang, A., "Exploiting nonstationarity for performance prediction," in *Proc. of EuroSys*, 2007.

[109] Stillger, M., Lohman, G. M., Markl, V., and Kandil, M., "LEO - DB2's learning optimizer," in *VLDB*, 2001.

[110] Swint, G. S., Jung, G., Pu, C., and Sahai, A., "Automated staging for built-to-order application systems," in *Proc. of NOMS*, 2006.

[111] Swint, G. S., Pu, C., Jung, G., Yan, W., Koh, Y., Wu, Q., Consel, C., Sahai, A., and Moriyama, K., "Clearwater: extensible, flexible, modular code generation," in *Proc. of ASE*, 2005.

[112] Tozer, S., Brecht, T., and Aboulnaga, A., "Q-cop: Avoiding bad query mixes to minimize client timeouts under heavy loads," in *Proc. of ICDE*, 2010.

[113] Transaction Processing Performance Council, "TPC benchmark W (web commerce)," February 2002.

[114] Urgaonkar, B., Shenoy, P., and Roscoe, T., "Resource overbooking and application profiling in shared hosting platforms," in *Proc. of OSDI*, 2002.

[115] Urgaonkar, B. and et al., "An analytical model for multi-tier internet services and its applications," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, pp. 291–302, June 2005.

[116] Voas, J. and Zhang, J., "Cloud computing: New wine or just a new bottle?," *IT Professional*, vol. 11, pp. 15–17, Mar. 2009.

[117] Wang, Q., Malkowski, S., Kanemasa, Y., Jayasinghe, D., Xiong, P., Pu, C., Kawaba, M., and Harada, L., "The impact of soft resource allocation on n-tier application scalability," in *Proc. of IPDPS*, 2011.

[118] Wang, Z., Chen, Y., Gmach, D., Singhal, S., Watson, B., Rivera, W., Zhu, X., and Hyser, C., "Appraise: Application-level performance management in virtualized server environments," *IEEE Trans. on network and service management*, vol. 6, no. 4, 2009.

[119] Wang, Z., Zhu, X., and Singhal, S., "Utilization and slo-based control for dynamic sizing of resource partitions," in *Proc. of DSOM*, 2005.

[120] Welsh, M. and Culler, D., "Adaptive overload control for busy internet servers," in *Proc. of USITS*, 2003.

[121] Xiong, P., Wang, Z., Jung, G., and Pu, C., "Study on perform-ance management and application behavior in virtualized environment," in *Proc. of NOMS*, 2010.

[122] Xiong, P., "Dynamic management of resources and workloads for rdbms in cloud: a control-theoretic approach," in *Proc. of SIGMOD PhD Symposium*, 2012.

[123] Xiong, P., Chi, Y., Moon, H. J., Pu, C., and Hacigumus, H., "Smart-SLA: intelligent management of virtualized resources for database systems in cloud environment," *IEEE Transactions on Parallel and Distributed Systems*. submitted.

[124] Xiong, P., Chi, Y., Moon, H. J., Pu, C., and Hacigumus, H., "Intelligent management of virtualized resources for database systems in cloud environment," in *Proc. of ICDE*, 2011.

[125] Xiong, P., Chi, Y., Zhu, S., Tatemura, J., Pu, C., and Hacigumus, H., "ActiveSLA: A profit-oriented admission control framework for database-as-a-service providers," in *Proc. of SOCC*, 2011.

[126] Xiong, P., Wang, Z., Malkowski, S., Wang, Q., Jayasinghe, D., and Pu, C., "Economical and robust provisioning of n-tier cloud workloads: A multi-level control approach," in *Proc. of ICDCS*, 2011.

[127] Xiong, P., Zhu, X., Griffith, R., Kambo, R., and Pu, C., "vPerfGuard: an automated tool for building adaptive performance models for applications in cloud environments," in *Proc. of SOCC*, 2012. submitted.

[128] Zhang, J., Chung, J.-Y., and Zhang, Z., "A router model for qos-based multimedia web services," in *Proc. of ICME*, 2003.

[129] Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Padala, P., and Shin, K., "What does control theory bring to systems research?," *SIGOPS Oper. Syst. Rev.*, vol. 43, pp. 62–69, Jan. 2009.

VITA

Pengcheng Xiong is a PhD candidate at School of Computer Science, Georgia Institute of Technology. His research interests lie in the fields of database management systems, distributed systems, operating systems, and autonomic computing. He holds a PhD and MS in Control Science and Engineering from Tsinghua University and a BS in Control Science and Engineering from Huazhong University of Science and Technology.