5-1-2016

Optimizing Cloud-Service Performance: Efficient Resource Provisioning Via Optimal Workload Allocation

Zhuoyao Wang

Zhuoyao Wang

Candidate

Electrical and Computer Engineering

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Majeed M. Hayat , Chairperson

Wei Wennie Shu

Balu Santhanam

Patrick Bridges

Maria Cristina Pereyra

Optimizing Cloud-Service Performance: Efficient Resource Provisioning Via Optimal Workload Allocation

by

Zhuoyao Wang

Bachelor of Engineering, Electrical Engineering, Jilin University, 2008
Master of Science, Electrical Engineering, University of New Mexico, 2011

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Engineering

The University of New Mexico

Albuquerque, New Mexico

May, 2016

Dedication

This work is dedicated to my beloved family, for their love and sacrifices.

Acknowledgments

I would like to take this opportunity to express my heartfelt to all those who helped me to make my dissertation work a success.

I express my sincere and wholehearted thanks, to my adviser, Prof. Majeed M. Hayat, for his suggestion, guidance, encouragement and support throughout this dissertation work. His enthusiasm in research and teaching has been a perennial source of inspiration to me. Working with him provided me an excellent learning opportunity.

I thank Prof. Nasir Ghani and Prof. Khaled B. Shaban for their support in conducting this research project and participation in my dissertation work. I would also like to thank Prof. Wei Shu, Prof. Patrick Bridges, Prof. Maria Cristina Pereyra and Prof. Balu Santhanam for agreeing to serve on my committee, reading my dissertation and providing useful suggestions during and after the defense exam.

I take this opportunity to thank my former colleagues, especially Dr. Jorge E. Pezoa, Dr. Mahshid Rahnamay-Naeini and Dr. Qi Wang for their help and contribution to the successful completion of this work.

This work was made possible by the NPRP 5-137-2-045 grant from the Qatar National Research Fund (a member of the Qatar Foundation).

Optimizing Cloud-Service Performance: Efficient Resource Provisioning Via Optimal Workload Allocation

by

Zhuoyao Wang

Bachelor of Engineering, Electrical Engineering, Jilin University, 2008
Master of Science, Electrical Engineering, University of New Mexico, 2011

Doctor of Philosophy, Engineering, University of New Mexico, 2016

Abstract

Cloud computing is being widely accepted and utilized in the business world. From the perspective of businesses utilizing the cloud, it is critical to meet their customers' requirements by achieving service-level-objectives. Hence, the ability to accurately characterize and optimize cloud-service performance is of great importance.

In this dissertation, a stochastic multi-tenant framework is proposed to model the service of customer requests in a cloud infrastructure composed of heterogeneous virtual machines (VMs). The proposed framework addresses the critical concepts and characteristics in the cloud, including virtualization, multi-tenancy, heterogeneity of VMs, VM isolation for the purpose of security and/or performance guarantee and the stochastic response time of a customer request. Two cloud-service performance metrics are mathematically characterized, namely the percentile of the stochastic response time and the mean of the stochastic response time of a customer request.

Based upon the proposed multi-tenant framework, a workload-allocation algorithm, termed max-min-cloud algorithm, is then devised to optimize the performance of the cloud service. A rigorous optimality proof of the max-min-cloud algorithm is given when the stochastic response time of a customer request assumed exponentially distributed. Furthermore, extensive Monte-Carlo simulations are conducted to validate the optimality of the max-min-cloud algorithm by comparing with other two workload-allocation algorithms under various scenarios.

Next, the resource provisioning problem in the cloud is studied in light of the max-min-cloud algorithm. In particular, an efficient resource-provisioning strategy, termed the MPC strategy, is proposed for serving dynamically arriving customer requests. The efficacy of the MPC strategy is verified through two practical cases when the arrival of the customer requests is predictable and unpredictable, respectively.

As an extension of the max-min-cloud algorithm, we further devise the max-load-first algorithm to deal with the VM placement problem in the cloud. MC simulation results show that the max-load-first VM-placement algorithm outperforms the other two heuristic algorithms in terms of reducing the mean of stochastic completion time of a group of arbitrary customers' requests. Simulation results also provide insight on how the initial loads of servers affect the performance of the cloud system.

In summary, the findings in this dissertation work can be of great benefit to both service providers (namely business owners) and cloud providers. For business owners, the max-min-cloud workload-allocation algorithm and the MPC resource-provisioning strategy together can be used help them build a better understanding of how much virtual resources in the cloud they may need to meet customers' expectations subject to cost constraints. For cloud providers, the max-load-first VM-placement algorithm can be used to optimize the computational performance of the service by appropriately utilizing the physical machines and efficiently placing the VMs in their cloud infrastructures.

Contents

List of Figures

List of Tables

Glossary

$\mathbf{w}$ A customer request with n applications $\mathbf{w} = \{w_1, \ldots, w_n\}$.

w_i The workload (i.e., number of tasks) of the ith application.

T_{w_i} The execution time of the ith applications in the request $\mathbf{w}$.

λ_{w_i} The execution rate of the ith application, which equals to the reciprocal of the mean of its execution time.

$T_{\mathbf{w}}$ The response time of the request $\mathbf{w}$.

$PT(\mathbf{w}; t)$ The percentile of the response time of the request $\mathbf{w}$.

$F_{T_{\mathbf{w}}}(t)$ The cumulative distribution function of $T_{\mathbf{w}}$.

$F_{T_{w_i}}(t)$ The cumulative distribution function of T_{w_i}.

$\mathbf{v}$ A set of arbitrary m VMs $\mathbf{v} = \{v_1, \ldots, v_m\}$.

$ET(\mathbf{w})$ The mean of the response time of the request $\mathbf{w}$.

$F_X(x)$ The cumulative distribution function of a random variable X.

λ_{v_j} ECU per vCPU of the jth VM.

$PT^{\text{ini}}(\mathbf{w}; t)$ The percentile of the response time of the request $\mathbf{w}$ associated with the initial allocation pattern.

Glossary

$PT^{\text{swap}}(\mathbf{w};t)$ The percentile of the response time of the request $\mathbf{w}$ after the reallocation.

$\mathbf{v}_{\text{mmc}}$ The set of m' fastest VMs out of the m VMs.

$PT^{\text{ini}}(\mathbf{w};t)$ The percentile of the response time of the request $\mathbf{w}$ when the n applications are allocated to the set of VMs $\mathbf{v}_{\text{mmc}}$.

$\lambda_{w_i}^{\text{mmc}}$ The execution rate of the ith application when the n applications are allocated to the set of VMs $\mathbf{v}_{\text{mmc}}$. ECU per vCPU of the jth VM.

$\mathbf{v}_{\text{arb}}$ The set of m'' VMs out of the m VMs that is different than $\mathbf{v}_{\text{mmc}}$.

$PT^{\text{arb}}(\mathbf{w};t)$ The percentile of the response time of the request $\mathbf{w}$ when the n applications are allocated to the set of VMs $\mathbf{v}_{\text{arb}}$.

$\lambda_{w_i}^{\text{arb}}$ The execution rate of the ith application when the n applications are allocated to the set of VMs $\mathbf{v}_{\text{arb}}$.

M The total number of types of VMs that can be chosen by the business to serve a customer request.

k_j The number of vCPUS in the jth type VM.

$price_j$ The usage price of the jth type VM.

$\mathbf{s}$ The set of VMs $\mathbf{v} = \{s_1, \ldots, s_M\}$ to be scheduled to serve a customer request.

$C(\mathbf{s})$ The provisioning cost for utilizing the set of VMs $\mathbf{s}$ in a unit time.

t_D Desired response time of a customer request.

α The confidence factor.

$\text{rev}(\mathbf{w})$ The revenue for serving a customer request.

profit($\mathbf{w}$) The profit for serving a customer request.

N_j The number of available jth type VM in the virtual infrastructure.

λ_c The Poisson arrival rate of the customer requests.

$\mathbf{r}(t)$ The real-time state of the cloud computing system.

$r_j(t)$ The load ratio of the jth physical machine at time t.

$\mathbf{u}$ The requests submitted by a group of arbitrary n customers.

u_i The workload size of the ith customer's request.

S_0 The initial state of the cloud system.

$ET_{S_0}(\mathbf{u})$ The mean of completion time of a group of n customers' request.

T_{u_i} The stochastic execution time of the ith VM.

λ_{u_i} The rate for executing the ith request/VM.

λ_{s_j} The CPU speed of the jth physical machine.

λ_{ij} The execution rate for the ith request/VM when it is mapped to the jth physical machine.

Chapter 1

Introduction

1.1 Motivation

Cloud computing is having a profound effect in today's business world. Many services, including email service, application hosting, data storage, e-commerce and more, have been implemented on cloud infrastructures. Cloud computing aims to provide high-performance but low-cost service to end users while maintaining the service elastic, scalable and resilient [1]. Typically, the performance of the cloud service to be delivered is specified through the service-level agreement (SLA), a contract negotiated and agreed between the service provider and the service user.

When a service provider (also refers to a business owner in this dissertation work) is deploying a cloud-based service, it is essential for it to deliver services that satisfy customers' requirements by having adequate computing resources, namely virtual machines (VMs). Meanwhile, it is also important for the business owner to avoid the cost of having unnecessary VMs (excessive computing power beyond its need). Therefore, an analytical model to accurately predict and optimize cloud-service performance is vital as it would be of great benefit to enhancing the quality

of service while keeping the cost of the business within a budget.

In recent years, numerous works have looked at modeling cloud services and analyzing their performance. Several performance metrics, such as the response time of a task or a batch of tasks, as well as the throughput and power consumption of cloud services have been analytically characterized [2–6]. In most of these existing works, the analysis of the cloud-service performance is based upon queuing theory (a detailed review of the related work will be given in Chapter 2.2.1).

The queuing models have proven their value in studying response time, throughput and stability for cloud services. However, none of these existing models is suitable to simultaneously address and investigate the following five critical concepts, characteristics and concerns that are present in the current cloud-service environments. These are: (i) virtualization and multi-tenancy [7–9], (ii) heterogeneity of VMs in the virtual infrastructure [6], (iii) stochastic response time of a customer request with a general probability distribution [4], (iv) VM isolation for the purpose of security and/or performance guarantee [10–12], and (v) efficient resource provisioning for serving dynamically arriving customer requests [13, 14]. Therefore, a new framework should be developed to complement the existing models to address the issues discussed above.

Besides the analytical cloud-service performance modeling, we further investigate the workload-allocation problem in the cloud. It is generally known that an efficient workload-allocation algorithm could profoundly improve the performance of cloud services. The purposes of the existing workload-allocation algorithms are including, for example, to minimize the resource provisioning cost, to minimize the data access latency between data node and computation node, and to minimize the bandwidth usage of the cloud service. To the best of our knowledge, however, there is no workload-allocation algorithm devised to optimize the computational performance of cloud services. This is mainly because that the computational performance met-

rics, such as the mean of the response time or the percentile of the response time of customer requests and the throughput of the cloud service, cannot be easily characterized as a linear cost function. Hence, we in this dissertation work aim to develop an optimal but simple-to-implement workload-allocation algorithm to optimize the computational performance of cloud services.

Furthermore, we also focus on the resource provisioning problem. Due to the fact that cloud computing lays a solid foundation for utilizing the *on-demand* resources during runtime. The importance of the resource provisioning problem has been even more emphasized compared to traditional distributed computing platforms. To this end, there is a need for devising a microscopic-level and SLA-based resource-provisioning strategy with low computation complexity, which specifies the amount of resources to be scheduled for serving dynamically arriving customer requests. An efficient provisioning strategy is able to serve more customers while saving on costs. To the best of our knowledge, however, most of the existing work either focused on long-term provisioning for large-scale workflows without considering the SLA requirements or suffered from high computation complexity (especially those based on stochastic programming).

1.2 Contributions of the Dissertation

In this thesis, a probabilistic multi-tenant framework is developed to complement existing models for analyzing and optimizing the computational performance of cloud services. The proposed framework considers a set of heterogeneous VMs that are utilized to construct a cloud-service infrastructure and serve customer requests. The customer requests studied here are complex scientific workloads consisting of many loosely-coupled applications [15]. To speed up the completion of customer requests, a request may be served by multiple VMs. Furthermore, multi-tenancy implies that

each VM may host several applications concurrently. The execution time of applications in the request are assumed stochastic with a general probability distribution. Based on this framework, two cloud-service performance metrics are analytically characterize, namely the percentile of the response time and the mean of the response time of a customer request. These two metrics have been widely used in cloud SLAs and studied in research literature.

With the two service-performance metrics characterized, we proceed to propose a workload-allocation algorithm, termed the max-min-cloud algorithm, to efficiently allocate the applications in an arbitrary customer request to the scheduled set of VMs. The optimality of the max-min-cloud algorithm is rigorously proven in terms of maximizing the percentile of the response time of the request. Extensive Monte-Carlo (MC) simulations are also conducted to examine the optimality of the max-min-cloud algorithm for serving customer requests with various workload patterns, and for the cases when different probability distributions are considered for the execution times of the applications in the requests.

In light of the max-min-cloud algorithm, the resource provisioning problem in the cloud is further investigated. In particular, a minimum-provisioning-cost (MPC) provisioning strategy is devised to efficiently serve dynamically arriving customer requests. The performance of the MPC strategy is compared with two other practical resource-provisioning strategies, which are the greedy-provisioning (GP) strategy and the random-provisioning (RP) strategy. This is done for two practical scenarios when either on-demand VMs or reserved VMs are utilized in the virtual infrastructure. These findings can be used by business owners to build a better understanding of how much virtual resources in the cloud they may need to meet customers' expectations subject to cost constraints.

As an extension of the max-min-cloud algorithm, we devise the max-load-first algorithm to deal with the VM placement problem in the cloud. MC simulation

results show that the max-load-first algorithm outperforms the other two algorithms in terms of reducing the mean of stochastic completion time of a group of customer requests. Simulation results also provide insight on how the initial loads of physical machines affect the performance of the cloud system. The max-load-first algorithm can be used by cloud providers to optimize the computational performance of the service by smartly mapping the VMs to the physical machines in their cloud infrastructures.

To date, our work has resulted in two papers [16, 17].

1.3 Dissertation Overview

The rest of this dissertation is organized as follows. In the next chapter we first present background information of the cloud, including some essential concepts and key characteristics in the current cloud computing environment. Then we review previous work relevant to this dissertation work. In particular, four categories of the related work are discussed, which are analytical performance modeling of cloud-service performance, workload allocation, resource provisioning and VM placement in the cloud.

In Chapter 3.1, we describe the cloud-service environment to be investigated and raise two challenging questions in such a cloud-service environment. The main effort dedicated in this dissertation work is to answer the two questions. A probabilistic multi-tenant framework for cloud services is then developed and two performance metrics are mathematically characterized in Chapter 3.2.

In Chapter 4.1, we apply the proposed analytical model to devise the max-min-cloud workload-allocation algorithm that optimize the performance of cloud services. Given the assumption that the execution times of the applications in the customer

request are exponentially distributed, we rigorously prove in Chapter 4.2 that the max-min-cloud algorithm can achieve the maximum percentile as well as the minimum mean of the response time for serving for an arbitrary customer request. In Chapter 4.3.1, extensive MC simulation results are conducted to illustrate the optimality of the max-min-cloud algorithm by comparing with the MCT-cloud algorithm for the case when the execution times of the applications in the customer request are assumed exponentially distributed. The impact of the different workload patterns of the customer request is investigated and discussed. In Chapter 4.3.2, we further show and explain that for the cases when the execution times the applications are with non-exponential probability distributions, the max-min-cloud algorithm has the greater advantage over the MCT-cloud algorithm. One more inherent value of the proposed analytical model is also shown by studying its computation complexity in Chapter 4.3.3.

In Chapter 5, we focus on investigating the resource provisioning problem in the cloud. In light of the max-min-cloud algorithm, we devise the MPC resource-provision strategy in Chapter 5.1. The MPC strategy specifies how to dynamically schedule the amount of computing resources in the virtual cloud-service infrastructure to serve the arriving customer requests. The MPC strategy aims to minimize the provisioning cost for serving a customer request while guarantee the performance quality of the service. After that we investigate the efficacy of the MPC strategy under two practical scenarios when the arrival of customer requests are unpredictable and predictable in Chapter 5.2 and Chapter 5.3, respectively. For the unpredictable case, we assume that on-demand VMs are utilized to serve the customer requests and that the virtual cloud-service infrastructure is elastic. For the predictable case, on the other hand, reserved VMs are utilized and the virtual cloud-service infrastructure is fixed. In both scenarios, the MPC strategy exhibits outstanding performance in terms of generating as much profit or revenue to service providers.

In Chapter 6, we extend the proposed multi-tenant framework and apply the max-min-cloud algorithm to superficially study the VM placement problem in the cloud computing system. The specified problem statement is given in Chapter 6.1. The performance of the cloud computing system is characterized in Chapter 6.2. To capture the performance interference between VMs, we approximately model such behavior by considering the impact of the physical load ratio on the service performance in Chapter 6.2.1. In Chapter 6.3, the max-load-first VM-placement algorithm is then devised, as a byproduct of the max-min-cloud algorithm, to optimize the computational performance of the cloud computing system. The supremacy of the max-load-first algorithm is shown in Chapter 6.4 by means of MC simulations in comparison to other two algorithms.

Our conclusions are given in Chapter 7. We point out that the results presented in this dissertation work are not limited to the case when cloud-service performance is dependent mainly on the ECU (i.e., vCPU speed) of the VMs. Namely, our framework can also be extended to scenarios when the cloud-service performance is determined and affected by other factors, such as network bandwidth, energy consumption, memory capacity and storage space. In those cases, the max-min-cloud algorithm and the MPC strategy are also useful for optimizing the cloud-service performance by considering the utilization of network bandwidth and/or the consumption of energy in the VMs.

Chapter 2

Cloud Computing and Relevant Research Challenges: An Overview

This Chapter begins with a brief description of cloud computing and some of the key concepts and characteristics that are widely-used in cloud computing, which is followed by discussion of some of the research challenges in the cloud relevant to our work.

2.1 Cloud Computing and its Key Characteristics

There exist various definition of cloud computing in literature. As an example, the work in [18] compared over 20 different definitions. However, the most adopted and cited version by researchers is provided by The National Institute of Standards and Technology (NIST) [1] in the following.

NIST definition of cloud computing *Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing*

resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

To deploy a cloud service, typically the *service providers* (refers also to business owners later in the dissertation) rent virtual resources from one or many *cloud (infrastructure) providers* to establish the infrastructure and serve the end users (refers to customers in the dissertation). The cloud providers lease resources according to a pay-per-use pricing model. During recent years, large technique companies such as Amazon, Google and Microsoft strive to provide more powerful, reliable and cost-efficient cloud platforms, such as Amazon EC2 [19], Google Compute Engine [20] and Microsoft Azure [21].

Similarly to the many existing computing platforms such as cluster computing, grid computing and utility computing, cloud computing, too, is a distributed computing paradigm. However, cloud computing evolves and adopts to the emerging technologies to make it highly attractive to business owners. Here we list some of the key concepts and characteristics that are commonly emphasized in modern cloud services.

2.1.1 Virtualization and Multi-tenancy

In general, cloud computing has profoundly accelerated as a result of the popularity and adoption of virtualization. Virtualization is a technology that abstracts away the details of physical hardware and provides virtual resources for high-level applications [8]. Specifically, virtualization software (known as a hypervisor) is used to run multiple VMs on a single physical server to provide the same functions as multiple physical machines [7].

By taking advantage of virtualization, the virtual resources in a cloud infrastruc-

ture can be easily pooled to serve multiple users and/or applications [7], which is termed as *multi-tenancy*. With the multi-tenancy characteristic, the cloud infrastructure is able to serve all the customer requests concurrently [9]. As a result, service providers maximize the utilization of resources, so as to decrease the operating costs.

2.1.2 Elasticity and Pay-Per-Use Pricing Model

When deploying services based on cloud, it is easy to expand and reduce resources according to specific service requirement, which is one of the most important advantages of cloud computing. Generally, service providers can unilaterally provision the virtual resources, i.e., expand and reduce the resources depending on their demands [1, 7]. For example, a business owner may pay and ask more *on-demand* resources from cloud providers for the peak hours in order to keep the same quality of service, and then release the extra resources after the peak hours or the completion of the customer requests.

The typical pricing model in cloud computing is known as *per-per-use*. Business owners pay cloud providers only when using the resources, either for a short term or for a longer duration. Such pricing lowers service operating cost as it charges customers on a per-use basis. However, it also introduces complexities in controlling the operating cost [8].

2.1.3 Performance Interference and Isolation

Virtualization and multi-tenancy allows different tenants/applications to run in the same VM, however, there is a common issue that extensive resource sharing can easily cause performance interference. In particular, when multiple applications are executed in the same VM there may exist the potential bad behaviors of one tenant

to adversely affect the performance of others in an unpredictable manner and the unbalanced situation where some tenants achieve very high performance at the cost of others,

Generally, the behavior of performance interference within/between VMs is fairly complicated. Some preliminary work have been achieved by conducting experimental or measurement studies to understand and analyze the performance interference effects in virtual environments [22–24].

To prevent performance interference, performance isolation is crucial and has been commonly utilized in the multi-tenancy environment. A simple and practical method is to limit the number of applications that are executed simultaneously in a VM [17]. Another method is that if a customer request is identified as an aggressive request, such request may be rejected [11].

2.1.4 Heterogeneity of Virtual Machines

Similar to other modern computing platforms, such as cluster computing [25] and grid computing [26], cloud computing environments also exhibit substantial heterogeneity [6, 27, 28]. Typically cloud providers, offer a wide selection of different types of VMs (namely instances) to business owners. Each instance comprises various combinations of CPU, memory, storage, and networking capacity. According to the purpose of the cloud service to be provided, business owners have the flexibility to choose the appropriate mix of VMs to fit their need. Although service providers can start deploying their virtual cloud-service infrastructure with near-homogeneous instances, the facility will become more heterogeneous over time due to the upgrade and replacement of VMs featuring the latest technologies.

2.2 Related Work

In this section, a survey of four categories of previous work will be presented. The first category is the work on the analytical performance modeling of cloud services. The second category of the related work is on workload allocation (or task scheduling) in heterogeneous computing and cloud computing environments. The third is on the resource provisioning problem and the last category is on the VM placement problem in the cloud

2.2.1 Analytical Performance Modeling of Cloud Services

Analytical performance modeling for distributed systems under parallel and grid computing environments has been the focus of attention for a long time. To the best of our knowledge, however, there are only a small portion of works that have addressed cloud environments.

In general, the times for serving customer requests in the cloud are considered stochastic. This is due to the fact that VMs may exhibit a varying and unstable performance especially when multiple tenants are concurrently running on the same VM. For the ease of deriving an exact (rather than approximation) analysis on the cloud-service performance especially in queuing models [2, 3, 5], the service times are typically assumed exponentially distributed. To make the results more practical, however, the service times must be modeled by a general probability distribution [4].

One of the pioneering works was proposed by Xiong *et al.* [2], where the cloud service environment is modeled as an M/M/1 queuing network. In their model, the arrival and response times of customer requests are assumed to be exponentially distributed. The probability distribution of the response time was characterized by

using the Laplace transform. The relationship among the maximum number of tasks, minimum service resources and highest level of service was then determined.

Subsequently, many queuing models were proposed to relax the assumptions in [2] and to consider additional stochastic factors that are inherent in the cloud. In [3], Yang *et al.* used an M/M/m/m+r queue to model the service environment of the cloud. The service response time of a customer request is assumed to be composed of submission, waiting, service and execution times. The probability density function and the mean of the customer's response time were derived. Khazaei *et al.* [4] assumed a general execution time for customer requests as well as a large number of servers in the cloud environment. The authors then modeled the cloud service based upon a M/G/m queuing system, and proposed an analytical technique for performance evaluation based on an approximate Markov-chain model.

Despite their many advantages, the queuing models have a common flaw in that they are unable to address virtualization and multi-tenancy in the cloud. In particular, the customers' requests to be served in the queuing models are considered as identical computational jobs rather than VMs that may consist of different number of jobs. In the queuing models, each server is assumed to execute only one job at one time. As the current virtualization technology is getting mature, however, the support of multi-tenancy becomes essential for serving customers' requests in cloud computing, where the requests from different customers (or tenants) could be served concurrently on the same physical server by sharing the hardware resources. Furthermore, it is also difficult for queuing models to capture the impact of hardware resources on the service performance of the cloud system. In fact, the service performance will be highly affected by the hardware resources of the system, including the number of servers, as well as the CPU speed of the servers, memory size and storage space.

Besides the above works using queuing models, Yeo and Lee [6] considered the

heterogeneity of the servers in the cloud. Namely, the authors assumed that the CPU speed of the servers in the cloud are uniformly-distributed random variables, and as such, the response times for executing a customer request also followed a uniform distribution. They then derived statistics (including the mean and variance) of the execution time for a given number of requests. The authors also applied regression methods to estimate the relationship between power and performance over time, and further performed energy-consumption analysis.

2.2.2 Workload Allocation

As for the second category of the related work, we shall discuss the workload-allocation (namely VM/task-scheduling) problem in cloud computing environments, which is one of the most critical factors that optimizes the cloud-service performance.

Many heuristic allocation/scheduling algorithms based upon queuing models have been proposed for various schemes that are used. For example, Braun *et al.* [29] provided a comparison of eleven task-scheduling algorithms through experiments. The experimental results indicated that a genetic algorithm leads to the best performance for all the cases. The min-min algorithm reported in [29] was the second best algorithm but with significantly less computational cost than that of the genetic algorithm. In [30] Wu *et al.* proposed the segmented min-min algorithm which improves the Min-min algorithm by scheduling large tasks first. The segmented min-min algorithm balances the load well and demonstrates even better performance in both makespan and running time. Later, Ritchie and Levine embedded a local-search procedure into the min-min algorithm [31]; they proposed the min-min+LS algorithm that significantly improves the performance of the min-min algorithm but maintains a similar computational cost to that for the min-min algorithm.

In [32], Maguluri *et al.* defined a stochastic model for a cloud service where tasks

are assumed to arrive according to a stochastic process and are subsequently queued. The authors focused on studying the stability of the cloud service and developed frame-based non-preemptive VM configuration algorithms. These algorithms can be made nearly throughput-optimal by choosing sufficiently long frame durations.

2.2.3 Virtual Machine Placement

Next, we shall discuss the VM placement problem in the cloud as the third category. As virtualization becomes a core technology of cloud computing, the problem of VM placement becomes crucial. Note that other researchers may call it as the VM-mapping, load-balancing or workload/server consolidation problem. In fact, the VM placement problem is similar to the workload allocation problem based on their essence. However, they are distinguished mainly by the assumptions, limitations and service environments associated with the specific problems.

A great volume of VM-placement algorithms has been proposed in the literature to achieve different goals, such as increasing quality of service (QoS), throughput and reliability of the cloud system. In [33], techniques of VM placement and consolidation which leverage min-max and shares features provided by hypervisors were explored. In [34], a dynamic consolidation mechanism based on constraint programming was developed. This consolidation mechanism was originally designed for homogeneous clusters. However, heterogeneity which is common in a multiple cloud provider environment was ignored.

Some of the existing VM-placement algorithms are based on the concept of load balancing. For example, Hu *et al.* [35] proposed a algorithm based on genetic algorithms. The authors used historical data and state of the system to achieve best system resource utilization so as to reduce the cost of dynamic VM migration. In [36] Liu *et al.* proposed the LBLV algorithm to balance the virtual storage in the cloud

storage system. The algorithm enhanced flexibility and robustness of the system and their results provided suggestions for designing the storage architecture of large-scale net data storage [36]. Meanwhile, Bhadani and Chaudhary [37] proposed the CLBVM algorithm, which used global state information to balance the load evenly in the cloud system. The algorithm aimed to decrease response time and increase system throughput, and therefore the overall performance of the system was improved.

Recently Xu and Fortes [38] further focused on multi-objective optimization for the VM placement problem. Their goal is to simultaneously minimize total resource wastage, power consumption and thermal dissipation costs. An genetic algorithm with fuzzy multi-objective evaluation is proposed for efficiently searching the large solution space and conveniently combining possibly conflicting objectives. A simulation-based evaluation using power-consumption and thermal-dissipation models, demonstrates the good performance, scalability and robustness of our proposed approach.

Survey papers such as [39] and [40] offer further details and addition examples for recently proposed algorithms.

2.2.4 Resource Provisioning in the Cloud

Finally, we shall discuss the resource provisioning problem in the cloud. Due to the elasticity of the cloud, the resource provisioning in the cloud may be more dynamic compared to the traditional distributed computing models, i.e., service providers could acquire resources based on the current demand, which can considerably lower the operating cost.

A large portion of the existing work are based on the macroscopic level [14,41–44], i.e., long-term provisioning for large-scale workflows. However, these work do not consider the notion of SLA requirements, which is one of the most important business

concepts in cloud computing. For example, Yang *et al.* [42] proposed a profile-based approach for developing just-in-time scalability for cloud applications. As a result, on-demand resources in cloud can be efficiently provisioned. In [43] and [14], Chaisiri *et al.* proposed the OVMP algorithm and its refinement, the OCRP algorithm, to minimize the total cost of resource provisioning. The authors applied stochastic integer programming with two-stage recourse to solve the resource provisioning problem. The uncertainty of the demand and provisioning cost is considered in their cloud-service model. The OVMP and OCRP algorithms can optimally adjust the tradeoff between reservation of resources and allocation of on-demand resources.

Recently, some of the work addressed the SLA requirements and the elasticity of cloud computing in the design of resource-provisioning algorithms. For example, Li and Guo [45] proposed a model for optimization of SLA-based resource scheduling in cloud computing based on stochastic integer programming technique. The authors applied Gröbner bases theory for solving the stochastic integer programming. However, the algorithm still suffers from the fact that the size of Gröbner basis grows exponentially. Feng *et al.* [46] relied on the performance-aware pricing model in SLAs for maximizing providers' revenues through Lagrange multiplier method based resource allocation among the customers. Zhao *et al.* [47] presented an end-to-end framework that facilitates adaptive and dynamic provisioning in the database tier by applying consumer-centric policies to satisfy the SLA requirements and control the monetary cost. Hwang *et al.* [48] presented a two-phase algorithm, long-term reservation and on-demand subscription, for service operators to minimize the cost of provisioning their service, but QoS has not been considered well. Ran *et al.* [49] focus on the dynamic resource provisioning for compute-intensive tasks with cost optimization and QoS guaranteeing. An online resource provisioning strategy based on the large deviation principle is proposed, which avoids requiring any priori knowledge of the workload and triggers a prompt response when the overload probability violates the desired QoS.

2.2.5 Summary

To date, the development of cloud computing is still at an early stage. There are still many challenges for researchers to make contributions and, in turn, bring significant progress to the industry. We, in this dissertation, focus on optimizing the computational performance of cloud services via efficient resource provisioning and optimal workload allocation.

For more details and research challenges of cloud computing, we point interested readers to the following survey papers [7–9, 50, 51].

Chapter 3

Probabilistic multi-tenant framework

In this chapter, we first describe the cloud-service environment to be investigated. The response time of an arbitrary customer request is then analytically characterized. In general, the response time is one of the most commonly adopted performance metrics specified in the cloud SLAs [52–54]; it is also considered extensively in the research literature [2–4, 15].

3.1 Cloud Service Environment

Suppose that a business deploys a virtual cloud-service infrastructure by utilizing on-demand or reserved VMs, purchased from cloud providers, to serve its customers. Here, these VMs are heterogeneous and they may belong to different categories. Namely, each category of VMs has several sub-types that are comprised of varying combinations of attributes, such as elastic computing unit (ECU), number of virtual CPUs (vCPUs), memory, storage, and networking capacity. As an example, Table 1

Table 3.1: Details of VMs provided by Amazon EC2

Category	Type	ECU	vCPUs	Memory(GB)	Price(per hour)[1]
General Purpose	m1.small	1	1	1.7	$0.022
	m1.medium	2	1	4	$0.044
	m3.large	6.5	2	7.5	$0.133
	m4.xlarge	13	4	16	$0.239
Compute Optimized	c1.medium	5	2	1.7	$0.075
	c3.large	7	2	3.75	$0.105
	c4.xlarge	16	4	7.5	$0.209
	c4.2xlarge	31	8	15	$0.419
Memory Optimized	r3.xlarge	13	4	30.5	$0.333
Storage Optimized	i2.xlarge	14	4	30.5	$0.853

lists four categories of VMs provided by Amazon EC2. Depending upon the purpose of a service, the business may choose the corresponding category of VMs or mixed categories of VMs. Since this work focuses on analyzing and optimizing the computational performance of cloud services, we assume that the virtual cloud-service infrastructure investigated in this study is comprised of only *Compute Optimized and General Purpose* [19] VMs, as listed in Table 1.

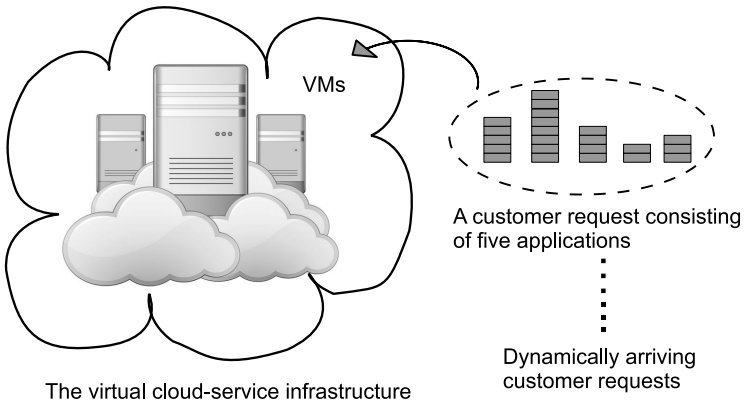
We assume that a customer request consists of a group of loosely-coupled and heterogeneous applications. Here, each application is considered as a workload that consists of a certain number of tasks. A task is assumed to be the smallest unit processed by a VM. For example, we could refer to a task here as 1,000 basic mathematical operations. We assume that the applications are not embarrassingly parallel [55]. That is, the tasks in an application are dependent and should not be parallelized to

[1]The listed prices for m1.small, m1.medium and c1.medium VMs have been adjusted to eliminate the unfairness in terms of ECU/price ratio for these three types of VMs due to the generation upgrade in Amazon EC2.

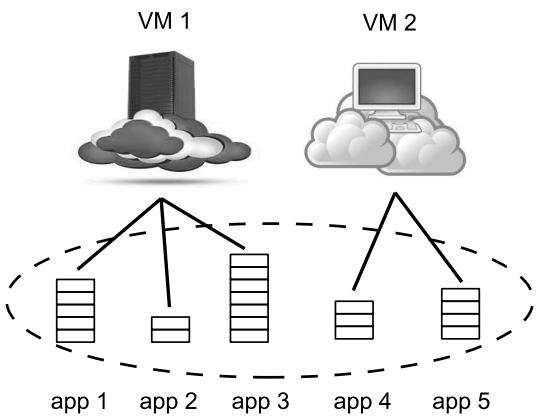
more than one thread since the overhead from synchronization and communication would dominate the execution time of the application. To fulfill a request, all the applications in the request must be executed. Due to the multi-tenancy characteristic of the cloud, it is also assumed that one VM is able to serve multiple applications simultaneously. To this end, the heterogeneous applications in a customer request are allocated to and executed concurrently by multiple VMs in the virtual infrastructure. Moreover, all the tasks in an application are served within the same VM.

To guarantee the security and performance of the cloud service, we further assume that an isolation mechanism is utilized. As typically employed in the cloud, a VM can only serve those applications that belong to the same customer. When a request is completed, the set of VMs that execute the request will be released and become available for serving subsequent customer requests. In addition, each vCPU in a VM serves only one application at a time. For example, a VM with two vCPUs may serve at most two applications concurrently. By doing so, we assume that the performance interference between the applications executed in the same VM can be ignored. As such, the execution of the applications in a customer request can be considered as mutually-independent.

For convenience, we give a simple example to illustrate how a request is served in the proposed cloud-service environment, which is also shown in Fig. 3.1. Suppose that a customer submits his/her request consisting of five heterogeneous applications for execution. Suppose also that two VMs in the virtual infrastructure, say VM 1 with four vCPUs and VM 2 with two vCPUs, are scheduled to serve this request. One scenario is that three of the applications in the request are hosted on VM 1 and the other two of the applications are hosted on VM 2, as shown in Fig. 3.1(b). In fact, there are $\binom{5}{1} + \binom{5}{2} = 15$ ways in total to allocate the five applications to the two VMs.



(a)



(b)

Figure 3.1: High-level cloud-service environment

3.1.1 Two Challenging Questions

Given the multi-tenant cloud-service environment described above, two challenging questions are raised.

- Suppose that VM 1 and VM 2 are scheduled to serve the request as illustrated in Fig. 3.1(b), then which allocation pattern of the 15 possibilities will result in the best performance of the request?

- Which VMs (including number and type) in the virtual infrastructure are the most appropriate amount of computing resource to serve the request?

We will specifically answer the above two questions in Chapter 4 and Chapter 5.

3.2 Cloud-Service Performance Evaluation

3.2.1 Characterization of the Stochastic Response Time

From a customer's point of view, the response time of his/her request is one of the most important concerns when using the cloud service [56]. We therefore focus on analyzing and minimizing the response times of customer requests as detailed next.

Let $\mathbf{w} = \{w_1, ..., w_n\}$ be a set representing a customer request. Such a request may consist of n arbitrary applications, where w_i is the workload size of the ith application. Based on the cloud-service environment described above, the n applications are executed by n vCPUs concurrently. Let T_{w_i}, for $i = 1, \ldots, n$, represent the execution times of the n applications in the request $\mathbf{w}$. In this thesis, we assume that the execution times of the n applications dominate the response time of such a request. As such, the response time of the request $\mathbf{w}$ is determined by the application that is completed last, and we can write

$$T_{\mathbf{w}} = \max(T_{w_1}, \ldots, T_{w_n}). \tag{3.1}$$

The execution time of the ith application, T_{w_i} for $i = 1, \ldots, n$, is assumed to be stochastic. Namely, the probability distribution of T_{w_i} is considered to be a general distribution with the execution rate λ_{w_i} that equals the reciprocal of the mean of its execution time.

3.2.2 Percentile of the Response Time

Next, we analytically characterize two statistics of the stochastic response time of the customer request as key performance metrics to evaluate the cloud service. The first metric is the percentile of the response time of the customer request before time t, which is defined as the cumulative distribution function (CDF) of the response time of the request [2]. In particular, the percentile of the response time gives the probability that request $\mathbf{w}$ can be completed before time t. This performance metric has also been considered by other researchers in prior works [13, 57, 58]. Namely, let $PT(\mathbf{w};t)$ represent the percentile of the response time of the request $\mathbf{w}$. By definition, we have

$$PT(\mathbf{w};t) \triangleq F_{T_{\mathbf{w}}}(t), \tag{3.2}$$

where $F_{T_{\mathbf{w}}}(t)$ is the CDF of $T(\mathbf{w})$. With the performance isolation mechanism applied to the cloud service, the execution of the n applications is assumed to be mutually-independent. To this end, we can further write the percentile of the response time as

$$PT(\mathbf{w};t) = \prod_{i=1}^{n} F_{T_{w_i}}(t), \tag{3.3}$$

where $F_{T_{w_i}}(t)$ is the CDF of the execution time of the ith application. Note that when a request $\mathbf{w} = \{w_1, ..., w_n\}$ is served by two different sets of VMs, $\mathbf{v}$ and $\mathbf{v}'$, for which the vCPUs in $\mathbf{v}$ are pair-wise faster than for those in $\mathbf{v}'$ (i.e., $\lambda_{w_i} \geq \lambda'_{w_i}$ for $i = 1, \ldots, n$), then

$$PT(\mathbf{w};t) \geq PT'(\mathbf{w};t) \text{ for all } t > 0. \tag{3.4}$$

3.2.3 Mean of the Response Time

The second performance metric to be characterized is the mean of the response time of the request, which is another widely-adopted performance metrics in cloud SLAs

and prior works [54, 59–61]. From equation (3.1), we can write

$$ET(\mathbf{w}) \triangleq \mathsf{E}[T_{\mathbf{w}}] = \mathsf{E}[\max\{T_{w_1}, ..., T_{w_n}\}]. \tag{3.5}$$

as the mean of the response time of the request $\mathbf{w}$. Note that for any non-negative random variable X,

$$\mathsf{E}[X] = \big(\int_0^\infty 1 - F_X(x)\big)dx,$$

where $F_X(x)$ is the CDF of random variable X. Hence, $ET(\mathbf{w})$ can be further written as

$$\begin{aligned} ET(\mathbf{w}) &= \int_0^\infty \big(1 - F_{T_{\mathbf{w}}}(t)\big)dt \\ &= \int_0^\infty \big(1 - \prod_{i=1}^{n} F_{T_{w_i}}(t)\big)dt. \end{aligned} \tag{3.6}$$

Equation (3.6) can be numerically evaluated given the knowledge of $F_{T_{w_i}}(t)$ for $i = 1, \ldots, n$.

It is important to note that there is a connection between $ET(\mathbf{w})$ and $PT(\mathbf{w}; t)$, that is

$$ET(\mathbf{w}) = \int_0^\infty \big(1 - PT(\mathbf{w}; t)\big)dt. \tag{3.7}$$

3.2.4 Execution Rate of the Workloads

To completely characterize the two performance metrics, it remains to model the CDFs, $F_{T_{w_i}}(t)$, of the execution times for the n applications. Here, we assume that a set of m available VMs in the virtual infrastructure, denoted by the set $\mathbf{v} = [v_1, ..., v_m]$, are scheduled to execute the n applications. Now suppose also that the ith application is hosted on the jth VM, whose ECU per vCPU is denoted by λ_{v_j}. Then the execution rate, namely the reciprocal of average execution time, of

the ith application, λ_{w_i}, is modeled by the following two realistic rules: 1) λ_{w_i} is proportional to λ_{v_j}; and 2) λ_{w_i} is inversely-proportional to the workload size of the application. Hence, we can write

$$\lambda_{w_i} = c\frac{\lambda_{v_j}}{w_i}, \tag{3.8}$$

where c is a constant indicating the rate for serving one task on a vCPU with one unit of ECU. For convenience, we assume that $c = 1$ in this thesis. Depending upon the specific probability distribution of the stochastic execution times of the applications, $F_{T_{w_i}}(t)$ can be further characterized given the knowledge of λ_{w_i}, for $i = 1, \ldots, n$.

We would like to clarify that $PT(\mathbf{w}; t)$ and $ET(\mathbf{w})$ also depend upon the set of m VMs $\mathbf{v}$ as well as the workload allocation algorithm that specifies how the n applications are hosted on the m VMs. However, the explicit reference to this dependence can be omitted from the notation for convenience.

Chapter 4

Optimal workload allocation

Workload allocation (or task scheduling) is a critical issue in heterogeneous computing environments such as cloud computing. Here, allocating the workloads efficiently can clearly improve service performance. However, finding the optimal workload-allocation algorithm in a heterogeneous computing environment is in general an NP-hard problem [30,31]. Hence many heuristic workload-allocation algorithms, such as the minimum completion time (MCT), the min-min and the max-min, were proposed in [29] and their efficacy were investigated under various schemes [29]. However, to the best of our knowledge, there is little work in the existing literature addressing the workload-allocation problem based upon the multi-tenancy principle in the cloud. In fact, the proposed algorithms in [29] behave quite differently in the multi-tenant model as illustrated later in this chapter.

In the following, we begin by reviewing the max-min-cloud algorithm, whose elementary version was first proposed in our prior work [16]. Here we rigorously prove the optimality of the max-min-cloud algorithm in Chapter 4.2. This proof illustrates that the max-min-cloud algorithm gives the best performance in the multi-tenant cloud-service environment for any arbitrary customer request, i.e., where the

optimality is in the sense of maximizing the percentile of the response time of the request and minimizing the mean of the response time of the request.

4.1 The Max-Min-Cloud Workload Allocation Algorithm

In brief, the motivation for devising the max-min-cloud algorithm is based upon the concept of load balancing for the case when the execution times of the applications are deterministic. Intuitively, the necessary condition for obtaining the minimum response time of a request is that the execution times of the n applications should be as close to each other as possible. Specifically, the max-min-cloud algorithm follows a greedy pattern for allocating the n heterogeneous applications to a fixed set of m VMs (with at least n vCPUs by default). It requires that the n applications in request $\mathbf{w}$ are sorted from the largest to the smallest, based upon their workload sizes, while the m VMs are also sorted in terms of ECU from the fastest to the slowest. Next, the application with the largest workload size is allocated to the fastest *available* VM. If there is at least one idle vCPU in a VM, this VM is defined as an available VM. For convenience, details of the max-min-cloud algorithm are summarized in Algorithm 1.

4.2 Optimality Proof of the Max-Min-Cloud Algorithm

Consider the case when a set of heterogeneous VMs, $\mathbf{v} = \{v_1, ..., v_m\}$, that have at least n vCPUs in total, is scheduled to serve a customer request $\mathbf{w} = \{w_1, ..., w_n\}$. We prove that the maximum $PT(\mathbf{w}; t)$, for any $t > 0$, and the minimum $ET(\mathbf{w})$

Algorithm 1 The max-min-cloud algorithm

Initiation: A customer submits his/her request $\mathbf{w}$ consisting of arbitrary n applications. Suppose that a set of m VMs in the cloud-service infrastructure are scheduled for serving the request, and the jth server has k_j vCPUs for $j = 1, ..., m$, respectively.

1: Sort the n applications in the request $\mathbf{w}$ from the largest to the smallest based upon the workload sizes of the n applications. Let a vector $\vec{\mathbf{w}} = [w_1, ..., w_n]$ represent the sorted n applications.
2: Sort the m VMs in terms of ECU from the largest to the smallest. Let a vector $\vec{\mathbf{v}} = [v_1, ..., v_m]$ represent the sorted m VMs.
3: **for** $i = 1$ **to** n **do**
4: **for** $j = 1$ **to** m **do**
5: **if** the number of applications that are hosted on the jth VM is smaller than k_j, **then**
6: allocate the ith application to the jth VM;
7: break;
8: **end if**
9: **end for**
10: **end for**

will be obtained by utilizing the max-min-cloud algorithm. The only assumption in our proof is that the stochastic execution times of applications are exponentially distributed. In this case, the formulas for $PT(\mathbf{w}; t)$ and $ET(\mathbf{w})$, from (3.2) and (3.6), become

$$PT(\mathbf{w}; t) = \prod_{i=1}^{n} \left(1 - e^{-\lambda_{w_i} t}\right) \tag{4.1}$$

and

$$ET(\mathbf{w}) = \int_0^{\infty} \left(1 - \prod_{i=1}^{n} (1 - e^{-\lambda_{w_i} t})\right) dt, \tag{4.2}$$

respectively. However, we will also conduct MC simulations to show the optimality of the max-min-cloud algorithm when the execution times are assigned with other probability distributions.

Remark 1: Equations (3.4) and (3.7) together imply the fact that the mean of the response time of a customer request, $ET(\mathbf{w})$, is minimized when the percentile of the response time of the request, $PT(\mathbf{w};t)$, is maximized for all $t > 0$.

Theorem 1: For any $t > 0$, the maximum percentile of the response time of a request $\mathbf{w} = \{w_1, ..., w_n\}$ is obtained when the max-min-cloud algorithm is utilized to allocate the n applications in the request $\mathbf{w}$ to a set of VMs $\mathbf{v} = \{v_1, ..., v_m\}$ that have exactly n vCPUs in total.

Clearly, Theorem 1 also implies that the minimum mean of the response time of the request $\mathbf{w}$ is obtained when the max-min-cloud algorithm is utilized as a consequence of Remark 1.

In order to prove Theorem 1, we begin by giving an example to define an operation termed *app-swap*, which helps us introduce Lemma 1. Suppose that two applications w_1 and w_2, with $w_1 > w_2$, are initially hosted on two VMs v_2 and v_1, with $\lambda_{v_1} > \lambda_{v_2}$, respectively. If we reallocate the application w_1 to v_1, and w_2 to v_2, then this operation is defined as an app-swap between w_1 and w_2. In particular, by performing an app-swap between w_1 and w_2, the application with the larger workload size (i.e., w_1) will be reallocated to the faster VM (i.e., v_1), while the application with smaller workload size (i.e., w_2) will be reallocated to the slower VM (i.e., v_2).

Lemma 1: Suppose that a customer request $\mathbf{w} = \{w_1, ..., w_n\}$ is executed by a set of heterogeneous VMs $\mathbf{v} = \{v_1, ..., v_m\}$ that have exactly n vCPUs in total. Suppose also that two of the applications w_1 and w_2, with $w_1 > w_2$, are allocated to the two VMs v_2 and v_1, with $\lambda_{v_1} > \lambda_{v_2}$, respectively. Let $PT^{\text{ini}}(\mathbf{w};t)$ denote the percentile of response time of the request $\mathbf{w}$ associated with the initial allocation

pattern. Suppose that we reallocate w_1 and w_2 by performing an app-swap between them, and let $PT^{\text{swap}}(\mathbf{w};t)$ denote the percentile of the response time of $\mathbf{w}$ after the reallocation. Then,

$$PT^{\text{swap}}(\mathbf{w};t) > PT^{\text{ini}}(\mathbf{w};t), \text{ for all } t > 0.$$

Now we first show Proposition 1 and Proposition 2 as the necessary steps before the proof of Lemma 1.

Proposition 1: $f(x) = \frac{xe^{-x}}{1-e^{-x}}$ is a monotonically decreasing function for $x > 0$.

Proof: We will show that the derivative of $f(x)$,

$$f'(x) = \frac{e^{-x}}{(1-e^{-x})^2}(1-x-e^{-x}), \tag{4.3}$$

is negative for $x > 0$. Let $g(x) = 1 - x - e^{-x}$, and note that $g(0) = 0$. To show that $f'(x)$ is negative, it is sufficient to show that $g(x)$ is a monotonically decreasing function, i.e., $g'(x) < 0$ for $x > 0$. Clearly,

$$g'(x) = e^{-x} - 1 < 0, \text{ for } x > 0. \ \square$$

Proposition 2: Suppose that $w_1 > w_2 > 0$ and $\lambda_{v_1} > \lambda_{v_2} > 0$. Let $\lambda_{w_1} = \lambda_{v_1}/w_1$, $\lambda_{w_2} = \lambda_{v_2}/w_2$, $\lambda'_{w_1} = \lambda_{v_1}/w_2$ and $\lambda'_{w_2} = \lambda_{v_2}/w_1$. Then the function

$$f(w_1, w_2, \lambda_{v_1}, \lambda_{v_2}, t) =$$
$$(1 - e^{-\frac{\lambda_{v_1}}{w_1}t})(1 - e^{-\frac{\lambda_{v_2}}{w_2}t}) - (1 - e^{-\frac{\lambda_{v_1}}{w_2}t})(1 - e^{-\frac{\lambda_{v_2}}{w_1}t}) \tag{4.4}$$

is positive for $t > 0$.

Proof: We rewrite the function $f(w_1, w_2, \lambda_{v_1}, \lambda_{v_2}, t)$ as a function of λ_{v_1}, denoted by $g(\cdot)$, with the other four variables fixed. We show that for all $w_1 > w_2 > 0$, $\lambda_{v_2} > 0$ and $t > 0$,

$$g(x) = (1 - e^{-\frac{x}{w_1}t})(1 - e^{-\frac{\lambda_{v_2}}{w_2}t}) - (1 - e^{-\frac{x}{w_2}t})(1 - e^{-\frac{\lambda_{v_2}}{w_1}t}) > 0,$$

when $x > \lambda_{v_2}$.

To see this, first note that $g(x) = 0$ when $x = \lambda_{v_2}$. Hence, it is sufficient to show that the function $g(x)$ is monotonically increasing for $x > \lambda_{v_2}$. We proceed by showing that the derivative of $g(x)$ is positive for $x > \lambda_{v_2}$, i.e.,

$$g'(x) = (1-e^{-\frac{\lambda_{v_2} t}{w_2}})\frac{t}{w_1}e^{-\frac{t}{w_1}x} - (1-e^{-\frac{\lambda_{v_2} t}{w_1}})\frac{t}{w_2}e^{-\frac{t}{w_2}x} > 0. \tag{4.5}$$

To do so, note that if $x_2 > x_1 > 0$ we immediately have

$$\frac{x_1 e^{-x_1}}{1-e^{-x_1}} \Big/ \frac{x_2 e^{-x_2}}{1-e^{-x_2}} > 1, \tag{4.6}$$

as a consequence of Proposition 1. Let $x_1 = \lambda_{v_2} t/w_1$ and $x_2 = \lambda_{v_2} t/w_2$. It is clear that $x_2 > x_1 > 0$. Hence, we can rewrite the inequality in (4.6) as

$$\frac{1-e^{-\frac{\lambda_{v_2} t}{w_2}}}{1-e^{-\frac{\lambda_{v_2} t}{w_1}}} \cdot \frac{(\frac{t}{w_1}\lambda_{v_2})e^{-\frac{t}{w_1}\lambda_{v_2}}}{(\frac{t}{w_2}\lambda_{v_2})e^{-\frac{t}{w_2}\lambda_{v_2}}} > 1. \tag{4.7}$$

Now for $x \geq \lambda_{v_2}$, define $h(x) \triangleq \frac{(\frac{t}{w_1}x)e^{-\frac{t}{w_1}x}}{(\frac{t}{w_2}x)e^{-\frac{t}{w_2}x}} = \frac{w_2}{w_1}e^{(\frac{1}{w_2}-\frac{1}{w_1})xt}$. We then rewrite the inequality in (4.7) as

$$\frac{1-e^{-\frac{\lambda_{v_2} t}{w_2}}}{1-e^{-\frac{\lambda_{v_2} t}{w_1}}} h(\lambda_{v_2}) > 1.$$

Note that $h(x)$ is monotonically increasing since it is an exponential function. In addition, it is obvious that the term $\frac{1-e^{-\frac{\lambda_{v_2} t}{w_2}}}{1-e^{-\frac{\lambda_{v_2} t}{w_1}}} > 0$. As such,

$$\frac{1-e^{-\frac{\lambda_{v_2} t}{w_2}}}{1-e^{-\frac{\lambda_{v_2} t}{w_1}}} h(x) > 1 \text{ for } x > \lambda_{v_2}, \tag{4.8}$$

which implies the inequality in (4.5). □

Proof of Lemma 1: Note that by performing the app-swap between w_1 and w_2, the execution rates of w_1 and w_2 change while the execution rates for the rest of the $n-2$ applications in $\mathbf{w}$ remain the same. According to (4.1),

$$PT^{\text{ini}}(\mathbf{w};t) = (1-e^{\lambda_{w_1}t})(1-e^{\lambda_{w_2}t})\prod_{i=3}^{n}\left(1-e^{\lambda_{w_i}t}\right)$$

and

$$PT^{\text{swap}}(\mathbf{w};t) = (1-e^{\lambda'_{w_1}t})(1-e^{\lambda'_{w_2}t})\prod_{i=3}^{n}\left(1-e^{\lambda_{w_i}t}\right).$$

Hence,

$$\begin{aligned} PT^{\text{ini}}(\mathbf{w};t) - PT^{\text{swap}}(\mathbf{w};t) = \big((1-e^{\lambda_{v_1}t})(1-e^{\lambda_{v_2}t}) \\ - (1-e^{\lambda_{v'_1}t})(1-e^{\lambda_{v'_2}t})\big)\prod_{i=3}^{n}\left(1-e^{\lambda_{v_i}t}\right). \end{aligned}$$

Using Proposition 2, we conclude that

$$(1-e^{\lambda_{w_1}t})(1-e^{\lambda_{w_2}t}) - (1-e^{\lambda'_{w_1}t})(1-e^{\lambda'_{w_2}t}) < 0 \text{ for all } t>0.$$

Therefore,

$$PT^{\text{ini}}(\mathbf{w};t) < PT^{\text{swap}}(\mathbf{w};t), \text{ for all } t>0.\square \tag{4.9}$$

Proof of Theorem 1: Let the n applications in the request be allocated to the m VMs in an arbitrary initial allocation pattern. We will then implement a sequence of app-swaps between two of the n applications based on a sorting algorithm, say bubble sort. After some finite number of app-swaps, there can be no more app-swaps within the n applications that can be implemented. At this point, the n applications are reallocated to the m VMs following the allocation pattern where the largest applications is hosted on the fastest VMs. Note that this final allocation pattern obtained at the end of the sequence of app-swaps is precisely prescribed by the max-min-cloud algorithm.

With the knowledge of Lemma 1, it is clear that each app-swap enhances the performance for serving the request $\mathbf{w}$. Therefore, we can claim that the maximal percentile of the response time of the request $\mathbf{w}$ is achieved for all $t > 0$ by utilizing the max-min-cloud algorithm. □

Next, we extend Theorem 1 by considering the general case when the set of m VMs has more than n vCPUs in total.

Theorem 2: Suppose that a request with n applications is to be executed by a set of m VMs that have more than n vCPUs in total. Then for any $t > 0$, the maximum percentile of the response time of the request $\mathbf{w}$ is obtained when the max-min-cloud algorithm is utilized.

Proof: First, consider the case when the n applications in the request are allocated to the set of m' fastest VMs, denoted by $\mathbf{v}_{\text{mmc}}$, of the m VMs. Without loss of generality, we can assume that the set of VMs in $\mathbf{v}_{\text{mmc}}$ has exactly n vCPUs. (For the case when there are more than n vCPUs in $\mathbf{v}_{\text{mmc}}$, the vCPUs that has the smallest ECU per vCPU can be ignored, since they will not be executing any applications.) By Theorem 1, the best performance for serving the n applications by the set of VMs $\mathbf{v}_{\text{mmc}}$ is obtained by applying the max-min-cloud algorithm. In this case, we use the term $PT_{\text{mmc}}(\mathbf{w}; t)$ to denote the percentile of the response time of the request $\mathbf{w}$, and the execution rates of the n applications are denoted by $\lambda_{w_1}^{\text{mmc}}, \ldots, \lambda_{w_n}^{\text{mmc}}$.

Next, consider the other case when a different collection of m'' VMs, denoted by $\mathbf{v}_{\text{arb}}$, are selected to serve the n applications, i.e., $\mathbf{v}_{\text{arb}} \neq \mathbf{v}_{\text{mmc}}$. Without loss of generality, we can also assume that the set of VMs in $\mathbf{v}_{\text{arb}}$ has exactly n vCPUs. Again by Theorem 1, the best performance is obtained by applying the max-min-cloud algorithm. Let $PT_{\text{arb}}(\mathbf{w}; t)$ denote the percentile of the response time of the request $\mathbf{w}$ for this scenario, and the execution rates of the n applications are denoted by $\lambda_{w_1}^{\text{arb}}, \ldots, \lambda_{w_n}^{\text{arb}}$.

Note that the VMs in $\mathbf{v}_{\text{mmc}}$ are the selection of the fastest VMs. Now when the max-min-cloud algorithm is utilized, the vCPU of the VM in $\mathbf{v}_{\text{mmc}}$ is faster than the vCPU of the VM in $\mathbf{v}_{\text{arb}}$ pair-wisely for executing the same workload, i.e.,

$$\lambda_{w_i}^{\text{mmc}} \geq \lambda_{w_i}^{\text{arb}} \text{ for } i = 1, \ldots, n.$$

Hence, we have

$$PT_{\text{mmc}}(\mathbf{w}; t) > PT_{\text{arb}}(\mathbf{w}; t) \text{ for all } t > 0$$

as a result of (3.4). The proof of Theorem 2 is completed by noting that the selection of the collection of m'' VMs was arbitrary. □

Theorem 2 also implies that the minimum mean of the response time is obtained when the max-min-cloud algorithm is utilized due to Remark 1. To this end, we have rigorously proven the optimality of the max-min-cloud algorithm in the sense of maximizing the percentile of the response time for any $t > 0$ as well as minimizing the mean of the response time for serving an arbitrary customer request when a fixed set of VMs has been scheduled to serve this request.

4.3 Numerical and Simulation Results

In this section, we conduct MC simulations to validate the efficacy of the max-min-cloud algorithm under various scenarios. To do so, we compare its performance to that of a widely-adopted workload-allocation algorithm termed the MCT-cloud algorithm, which extends the minimum completion time (MCT) algorithm introduced in [31]. The MCT-cloud algorithm allocates the applications in a request $\mathbf{w}$, in arbitrary order, to the fastest available VM in a scheduled set of VMs $\mathbf{v}$. The MCT-cloud algorithm performs well for cloud services whose virtual infrastructure consists of near-homogeneous VMs [6].

Table 4.1: Six workload patterns of the request $\mathbf{w}$

request pattern	workloads in the request $[w_1, \ldots, w_9]$
1	$[370, 10, 10, 10, 10, 10, 10, 10, 10]$
2	$[90, 90, 90, 90, 50, 10, 10, 10, 10]$
3	$[50, 50, 50, 50, 50, 50, 50, 50, 50]$
4	$[90, 50, 50, 50, 50, 50, 50, 50, 10]$
5	$[90, 80, 70, 60, 50, 40, 30, 20, 10]$
6	$[60, 60, 60, 60, 52, 52, 38, 38, 30]$

4.3.1 Response Times with Exponential Distributions

Consider the case for which a customer submits his/her request $\mathbf{w}$ consisting of nine applications with 450 tasks to the virtual cloud-service infrastructure. We assume that four VMs, including one c4.xlarge instance, one c3.large instance, one c1.medium instance and one m1.medium instance, are scheduled to serve the request. Six workload patterns of the request with the same total number of tasks are studied. The execution times of the applications in the request are exponentially distributed. The mean of response times of the six requests are listed in Table 4.1 for the cases when the max-min-cloud algorithm and the MCT-cloud algorithm are utilized. Here, the columns labeled as "Theo." and "Sim." present the results obtained by numerically evaluating the analytical characterization of $ET(\mathbf{w})$ in (3.2) and after averaging 10,000 realization of independent experiments, respectively. It is noted that the execution rates of the applications cannot be determined when the MCT-cloud algorithm is utilized. Hence, there is no theoretical prediction of $ET(\mathbf{w})$ for the MCT-cloud algorithm.

It can be observed from Table 4.1 that for any of the six patterns, the mean of the response time corresponding the max-min-cloud algorithm are less than or equal

Table 4.2: The mean of the response time of request **w** for the six workload patterns

request pattern	mean of response time (in seconds)		
	theo. prediction	sim. max-min-cloud	sim. MCT-cloud
1	93.06	92.97	133.12
2	48.40	48.37	67.83
3	47.55	47.53	47.58
4	45.45	45.43	53.92
5	43.55	43.52	59.46
6	42.47	42.44	50.83

to that for the MCT-cloud algorithm. Note that the theoretical prediction of $ET(\mathbf{w})$ also matches the MC simulation results. It is also important to note that different patterns of the request **w** may lead to variation in the values of $ET(\mathbf{w})$, even though these patterns have the same number of tasks in total. For example, $ET(\mathbf{w})$ of "pattern 1" request is 167.8 seconds, which is about three times longer than that of "pattern 6" request (58.5 seconds). This is because the workloads in "pattern 1" request are highly unbalanced as compared to the distribution of the ECU per vCPU of the four scheduled VMs that serve the request. Meanwhile, the workloads in "pattern 6" request have the most similar distribution of ECU per vCPU of the four VMs among the six patterns of the request under study.

Next, in Fig. 4.1, we show the percentile of the response time of the request **w** by applying the max-min-cloud algorithm and the MCT-cloud algorithm (for the six patterns described earlier). The solid curves representing the theoretical predictions are obtained by evaluating (3.2) numerically when the max-min-cloud algorithm is utilized. The MC simulation results marked by squares and crosses are presented respectively for the max-min-cloud algorithm, and the MCT-cloud algorithm. Each point in the MC simulations are averaged using 10,000 independent experiments. As expected, the max-min-cloud algorithm leads to larger values of $PT(\mathbf{w}; t)$ for all

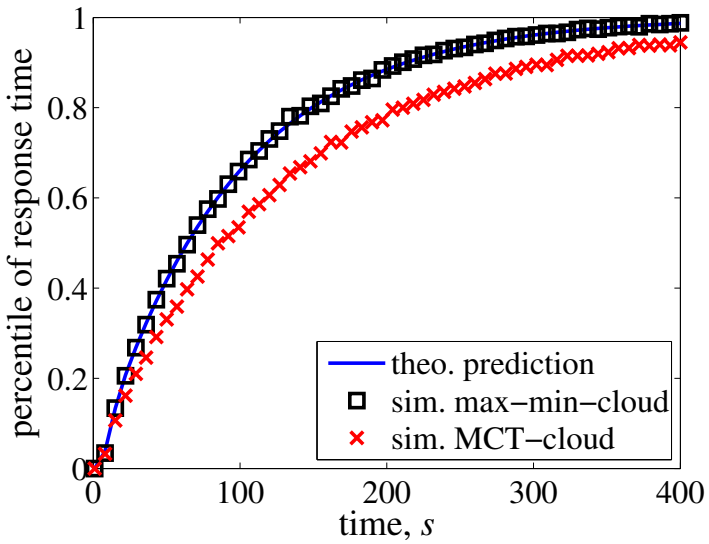
$t > 0$. Here too, our theoretical predictions agree well with the MC results when the max-min-cloud algorithm is utilized. It is interesting to observe in Fig. 4.1(e) that all the curves for "pattern 5" requst coincide. This is because all the applications in the request have the same workload sizes. As such, there is no difference for allocating applications in the request when the two algorithms are utilized.

To better illustrate the impact of the workload pattern of the request on $PT(\mathbf{w}; t)$ we also show results obtained by theoretical predictions for the six workload patterns of the request together in Fig. 4.2.

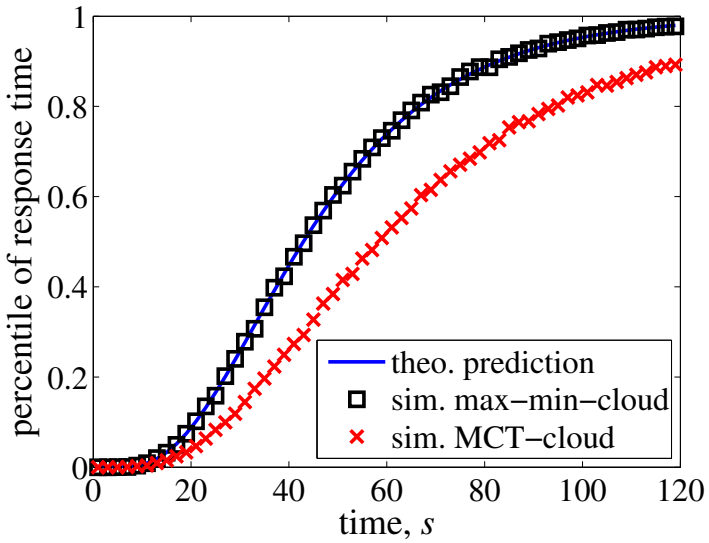
4.3.2 Response Times with Non-Exponential Distributions

Next, we illustrate the optimality of the max-min-cloud algorithm when the execution times of the applications are not exponentially-distributed. We consider two cases when the execution times are (a) truncated normally-distributed whose variance is one third of its mean (as similarly considered in [6]) and (b) Erlang-distributed. The corresponding percentile of the response time of the "pattern 4" request under these two distributions are shown in Figs. 4.3(a) and (b), respectively.

It can be observed from Fig. 4.3 that the superiority of the max-min-cloud algorithm over the MCT-cloud algorithm is more profound versus the case when the execution times of the applications are exponentially-distributed. For example, in Fig. 4.1(a) and (b) when $t = 60$ seconds, the max-min-cloud algorithm can lead to approximately 55% higher probability for completing the request than that for the MCT-cloud algorithm. Meanwhile in Fig. 4.1(c) the related probability is about 50% higher.



(a)



(b)

Figure 4.1: Percentile of the response time of request with pattern 1 and 2

4.3.3 Computation Complexity

Besides the accuracy, another critical criteria for judging the proposed analytical model is the computation complexity. In general, we wish that computation com-

plexity of the analytical model is comparable to or even lower than MC simulation for acquiring the same results. Furthermore, we also wish that the computation complexity of the analytical model is not growing exponentially (typically linear complexity is the best case), so that the model can still be utilized to deal with similar problems when they may highly scale in other scenarios.

With the max-min-cloud algorithm at hand, we now can show the computational efficacy of the analytical model in this section. To do so, we compare the computational costs for acquiring the mean (rather than the percentile) of the response time of the request by numerically evaluating the analytical characterization in (3.6) against by MC simulations, since an extra integration is needed for evaluating $ET(\mathbf{w})$ compared to $PT(\mathbf{w})$ as explicitly stated in (3.7).

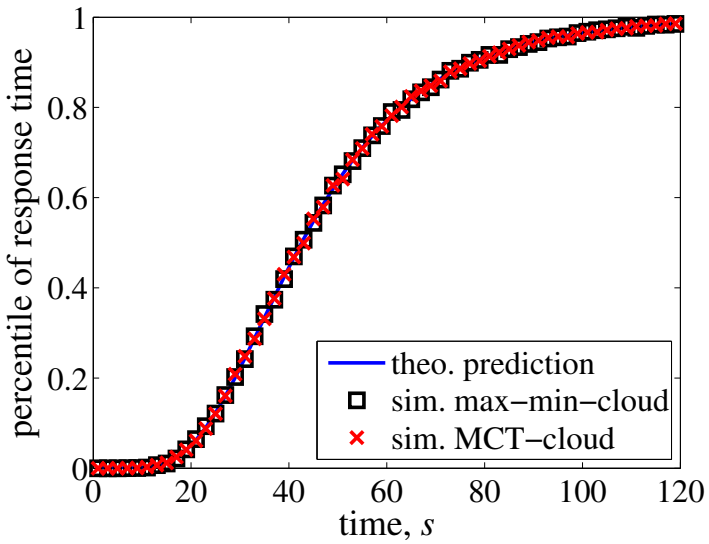
Table 4.3 lists the computation times for the "pattern 4" request under four cases. The execution times of the applications are assumed exponentially distributed. All the results in Table 4.3 as well as in this section are acquired from Matlab R2011b on a laptop equipped with Intel Core i5-3210M 2.5GHz 4-thread CPU and 16GB RAM. As expected, the error in MC simulations decreases as the number of independent experiments increases. In order to have a precise approximation, i.e., say the error is less than 2%, it is necessary to run the simulation for at least 10,000 independent experiments. In that case, the computational cost for running simulation would be more than ten times compared to cost by numerically evaluating the analytical characterization.

Next, we investigate how computation complexity of the proposed analytical model grows as the number of applications increases. Without loss of generality, we can simply consider the case that a customer request consisting of n applications with the same workload size, say 100, are allocated onto n vCPUs with the same ECU per CPU, say 1. In Fig. 4.4, we show the computational cost for evaluating $ET(\mathbf{w})$ as a function of n. The results are averaged by 100 iterations. It is clear

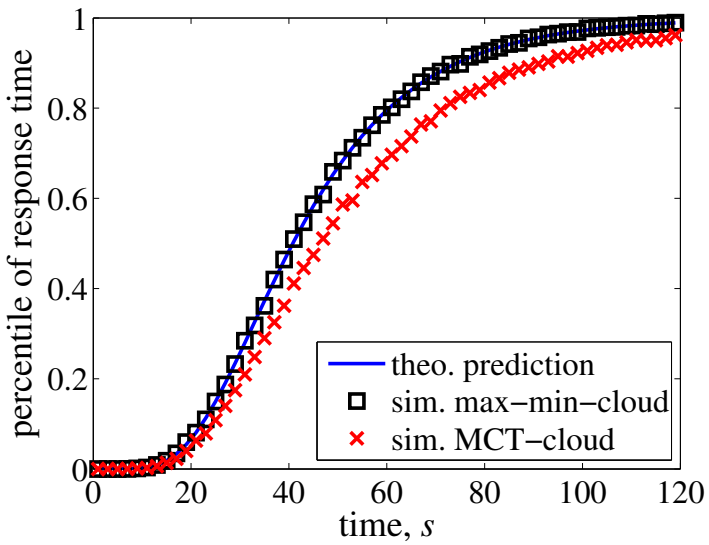
Table 4.3: Computational costs: analytical model vs. MC simulations

	Anal. result	Simu. results		
Number of experiments	N/A	100	1,000	10,000
Mean of response time	45.4515 s	49.0853 s	43.6593 s	46.6074 s
Error(approx.)	N/A	8%	5%	2%
Computation cost (in seconds)	0.00698	0.00267	0.00951	0.07584

that the computation complexity of the proposed model is approximately growing linearly, i.e., $\mathcal{O}(n)$.

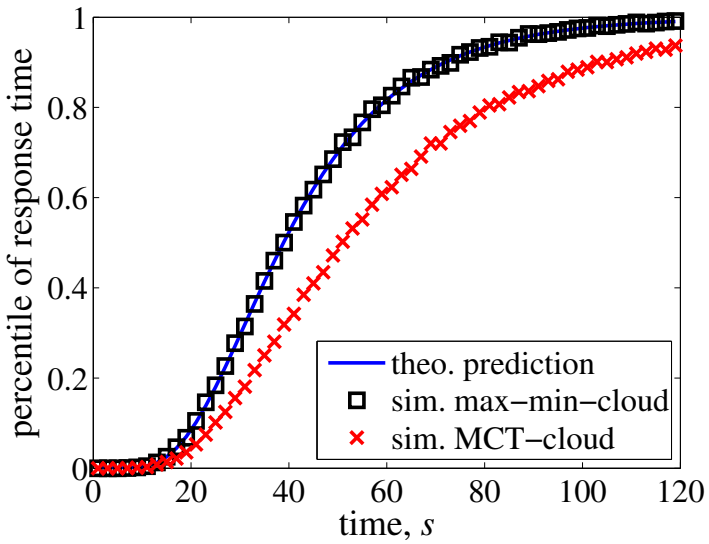


(c)

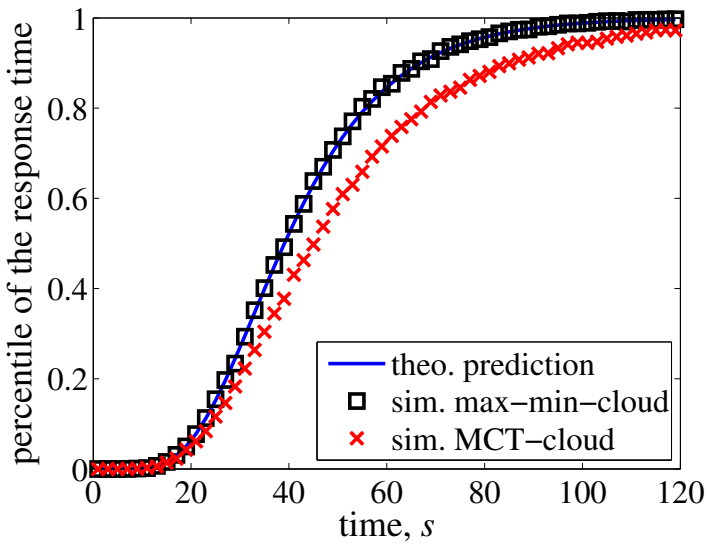


(d)

Figure 4.1: Percentile of the response time of request with pattern 3 and 4



(e)



(f)

Figure 4.1: Percentile of the response time of request with pattern 5 and 6

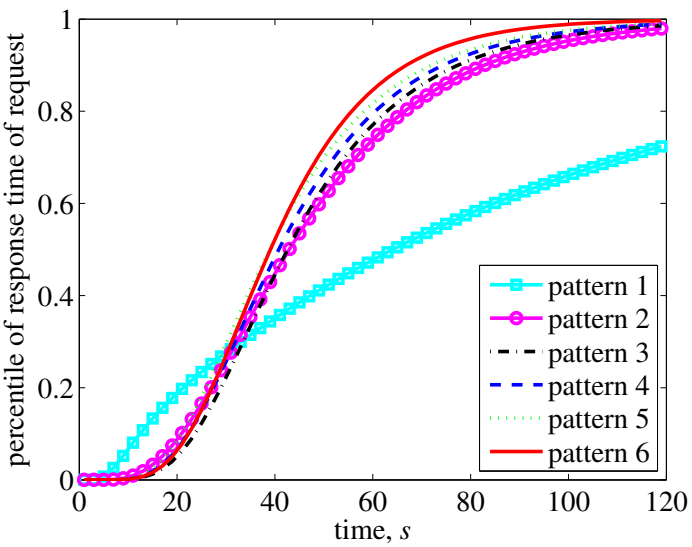
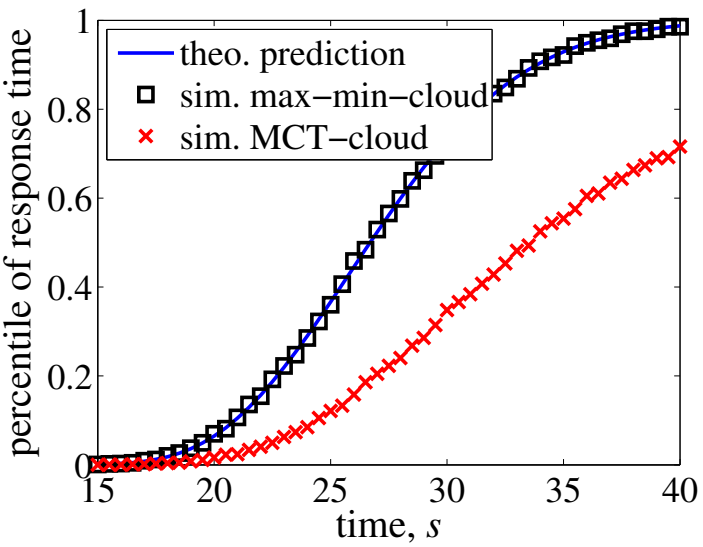
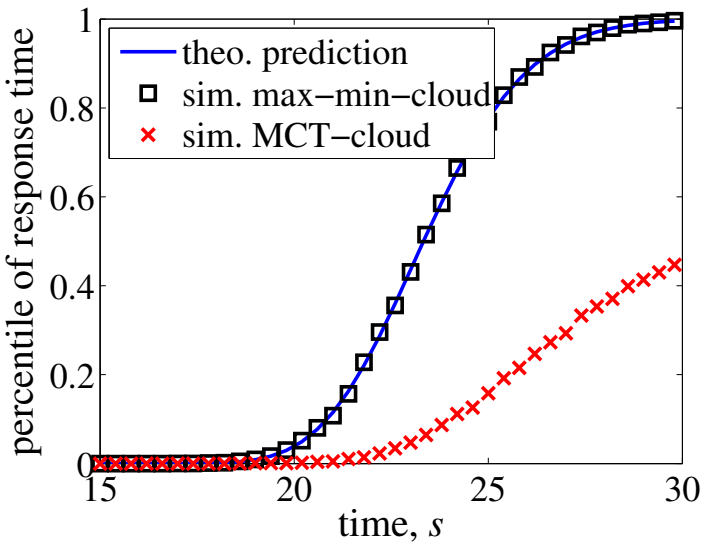


Figure 4.2: Percentile of the response times for the six patterns



(a)



(b)

Figure 4.3: Percentile of the response times, where the execution times of the applications in the request are (a) truncated normally-distributed and (b) Erlang-distributed

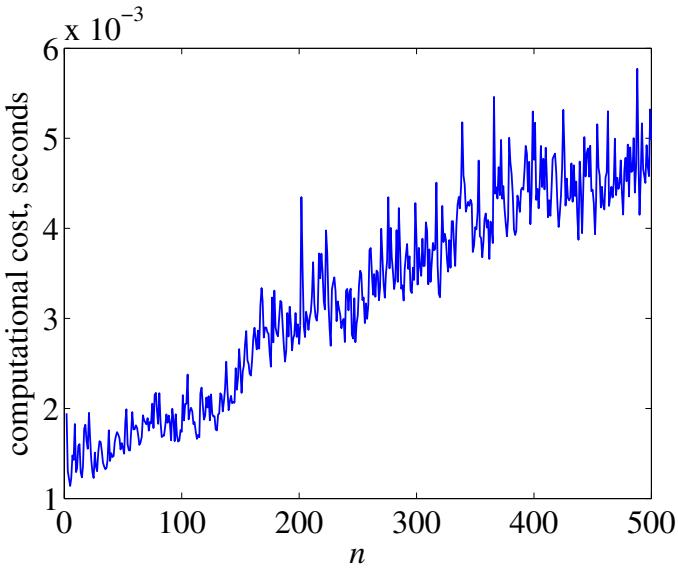


Figure 4.4: The computation complexity of the model as a function of the number of applications n.

Chapter 5

Efficient Resource Provisioning

When a customer submits his/her request, the amount of computing resources for serving this request needs to be determined and scheduled before the execution of the request. To schedule an appropriate set of VMs for serving customer requests is a challenging problem, typically termed as resource provisioning in the cloud [53]. Here the challenge in devising an efficient provisioning strategy is twofold. On one hand, the business must provide a service that meets the customer expectations (i.e., avoid under-provisioning). On the other hand, the business must try to maximize its profit or keep the cost at a minimum for operating the cloud service (i.e., avoid over-provisioning).

Therefore, our goal is to find an efficient provisioning strategy that brings as much profit or revenue to the business as possible. To this end, we in this chapter propose a minimum-provisioning-cost (MPC) resource-provisioning strategy for serving the incoming customer requests and compare its performance with other resource-provisioning strategies via MC simulations.

5.1 The Minimum-Provisioning-Cost Strategy

In brief, the idea is to minimize provisioning costs while guaranteeing the performance for serving customer requests. Specifically, suppose that there are M types of VMs that can be chosen by the business to serve a customer request $\mathbf{w} = \{w_1, ..., w_n\}$. The jth type VM has k_j vCPUs and its usage price is denoted as price_j. The MPC strategy needs to determine the appropriate set of VMs, denoted by $\mathbf{s} = \{s_1, \ldots, s_j, \ldots, s_M\}$, to be scheduled to serve the request $\mathbf{w}$. Here, s_j represents the number of jth type VM in $\mathbf{s}$ for $j = 1, \ldots, M$. We model the provisioning cost for serving the request $\mathbf{w}$ as the product of the price paid for using the set of VMs $\mathbf{s}$ and the mean of the response time of the request $\mathbf{w}$. Mathematically, we define the provisioning cost as

$$C(\mathbf{s}) = ET(\mathbf{w}) \sum_{j=1}^{M} \text{price}_j. \tag{5.1}$$

The set of VMs determined by the MPC strategy is the set $\mathbf{s}$ that minimizes $C(\mathbf{s})$ subject to the following three constraints:

(a) $\sum_{i=1}^{M} k_j s_j \geq n$, namely the total number of vCPUs of the VMs in $\mathbf{s}$ should be greater than or equal to the number of applications in the request $\mathbf{w}$.

(b) $\sum_{i=1}^{M} k_j s_j \leq n + \max_{j=1,\ldots,M}(k_j)$. In light of the optimality proof of the max-min-cloud algorithm given in Chapter 4.2, it is clear that the redundant and idle VMs/vCPUs that do not run any workload do not enhance the service performance of the customer request.

(c) $PT(\mathbf{w}; t_D) \geq \alpha$. Given the desired response time t_D, the probability of completing the request $\mathbf{w}$ before t_D should be greater than or equal to a confidence factor α. Here $PT(\mathbf{w}; t_D)$ is computed by utilizing the max-min-cloud algorithm.

To solve this optimization problem required by the MPC strategy, we have to

exhaustively check all the combinations of possible values of $n_1, n_2, ..., n_M$ that satisfy the three constraints. The solution is one of the elements in the search space that leads to the minimum $C(\mathbf{s})$. However, it is important to note that constraints (a) and (b) together reduce the size of the search space. Hence, the computational complexity for finding the solution is much lower than a typical combinatorial optimization problem.

Next, we investigate the efficacy of the MPC strategy under two practical scenarios when the arrival of customer requests is unpredictable and predictable. For the unpredictable case, we assume that on-demand VMs are utilized to serve the customer requests and that the virtual cloud-service infrastructure is elastic. For the predictable case, on the other hand, reserved VMs are utilized and the virtual cloud-service infrastructure is fixed.

5.2 Elastic Virtual Cloud-Service Infrastructure with On-Demand VMs

Consider a scenario where the arrival of customer requests is highly dynamic and unpredictable. In this case, on-demand VMs are typically used by the business for serving these customer requests. The business can take full advantage of the elasticity of the cloud, i.e., the VMs in the virtual infrastructure can be unilaterally expanded and reduced depending upon the demand [1, 7]. When a request is completed, the on-demand VMs will then be shut down and removed from the virtual infrastructure. The cost for operating the cloud service on an elastic virtual infrastructure is *pay-per-use*, i.e., the longer a VM is used the more the business pays.

Next, we present a experimental study to illustrate the efficacy of the MPC strategy. Assume that four types of VMs, including c4.xlarge, c3.large, c1.medium

and m1.medium, can be chosen to serve the customer requests. The costs for utilizing these four VMs are listed in Table 1. For one realization of the experiment, we assume that 100 customers submit their requests. Here, each request is randomly chosen from 8 different patterns of a request with 450 tasks in total (as listed in Table 5.1).

Table 5.1: Eight patterns of a request with 450 tasks in total

pattern #	workloads in the request
1	$[80, 70, 60, 50, 40, 40, 35, 30, 25, 20]$
2	$[80, 80, 45, 45, 45, 45, 45, 45, 10, 10]$
3	$[90, 90, 50, 50, 50, 50, 50, 10, 10]$
4	$[90, 80, 70, 60, 50, 40, 30, 20, 10]$
5	$[90, 80, 70, 65, 55, 40, 30, 20]$
6	$[90, 90, 60, 60, 55, 55, 20, 20]$
7	$[90, 80, 70, 60, 60, 50, 40]$
8	$[90, 85, 65, 65, 65, 50, 30]$

In this particular scenario, the metric to be used for evaluating the efficacy of a resource-provisioning strategy is the average profit for serving a certain number of customers. We further assume that the business can earn full revenue from a customer if his/her request is completed within the desired response time, and partial (or even zero) revenue is generated if the completion time is larger than the desired response time. Specifically in our experiment, the revenue for serving a customer request (same for the eight patterns) is modeled as

$$\text{rev}(\mathbf{w}) = \begin{cases} r & \text{if } T_{\mathbf{w}} \leq t_D \\ r/2 & \text{if } t_D < T_{\mathbf{w}} \leq 2t_D, \\ 0 & \text{if } T_{\mathbf{w}} > 2t_D \end{cases} \tag{5.2}$$

where r is the rated revenue when a request can be completed within the desired response time t_D. Hence the profit for serving a customer is then

$$\text{profit}(\mathbf{w}) = \text{rev}(\mathbf{w}) - C(\mathbf{s})T_{\mathbf{w}}. \tag{5.3}$$

For comparison, we apply two other commonly-used and straightforward resource-provisioning strategies as benchmarks. One is the greedy-provisioning (GP) strategy, where the minimum number of VMs with the highest ECU that have at least n vCPUs will be scheduled to serve the request. The other is the random-provisioning (RP) strategy, where VMs that have at least n vCPUs will be randomly chosen and scheduled to serve the request.

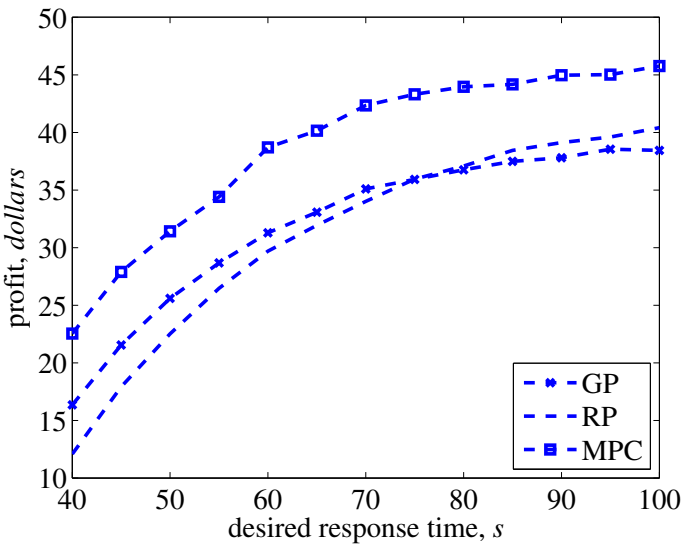


Figure 5.1: The average profit for serving 10,000 customer requests as a function of the desired response time

The average profit for serving 10,000 customers as a function of the desired response time are shown in Fig. 5.1, where the rated revenue $r = 0.01$ (namely 1 cent). The three curves represent the results obtained by utilizing the GP strategy, the RP algorithm and the MPC strategy, respectively. Each point in the three curves is averaged by 1,000 realizations. As expected, the average profit increases as t_D becomes larger. It is clear that the MPC strategy outperforms the other two strategies in terms of generating more profit for the business under any t_D. It is also interesting to find that the GP strategy outperforms the RP strategy when $t_D < 75$

but falls behind when $t_D > 75$. The reason here is that the RP strategy is not able to guarantee the service performance. As a result, a substantial proportion of the requests may be completed beyond t_D when the desired response time is relatively short, and this leads to a large loss in revenue. Meanwhile, when the desired response time is relatively long, the main factor to determine the revenue is the provisioning cost. Hence the performance of the GP strategy becomes worse as t_D increases. In summary, the MPC strategy overcomes the inherent shortcomings of the other two resource-provisioning strategies and emerges the best performance in terms of generating the most profit.

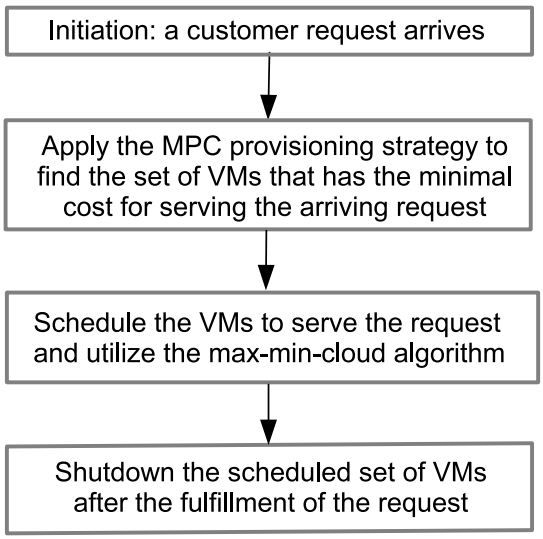


Figure 5.2: Flowchart for efficiently serving dynamically arriving customer requests when the virtual cloud-service infrastructure is elastic by utilizing on-demand VMs

The results shown in Fig. 5.1 are highly helpful for businesses negotiating with their customers on some critical attributes of the SLAs, e.g., such as the charge plan and the desired response time for serving the requests. For convenience, in Fig. 5.2 we show a flowchart for efficiently serving dynamically arriving customer requests in an elastic virtual cloud-service infrastructure with on-demand VMs.

5.3 Fixed Virtual Cloud-Service Infrastructure with Reserved VMs

Next, we consider another practical scenario when all the VMs in the virtual cloud-service infrastructure are reserved, and no on-demand VMs will be added to the virtual infrastructure. Such cloud service is typically used when the response time is not the first concern by the customers, and the arrival of the customers follows a certain pattern. The advantages of utilizing reserved VMs compared to the on-demand VMs include less cost (e.g., a significant discount up to 75% in Amazon EC2 [19]) for obtaining the same amount of computing power per unit time and the ease of maintenance and management of the VMs (in the virtual infrastructure).

In this case, the virtual cloud-service infrastructure and the cost for operating such cloud service is fixed. When a customer submits his/her request, the business can only utilize the available VMs in the fixed virtual infrastructure to serve the request. The business also has the right to reject a customer's request [11] if there is insufficient number of available VMs. We also assume that the business can earn full revenue from a customer as long as his/her request is accepted.

Note that the MPC strategy proposed in Chapter 5.1 needs to be slightly modified by adding one more constraint in order to suit the fixed virtual infrastructure:

(d) $s_j \leq N_j$, $j = 1, ..., M$, that is, the number of VMs in $\mathbf{s}$ is limited by the number of available VMs, denoted by N_j for $j = 1, ..., M$, in the virtual cloud-service infrastructure.

Note that there may be no solution to this optimization problem, which implies that the there is insufficient number of available VMs to serve the request. Such request will be rejected. Clearly in this scenario an efficient resource-provisioning strategy aims to serve as many customers as possible. Hence, we use the rejection

rate of customers to be served as a metric to evaluate the performance of a resource-provisioning strategy. Next, we present a experimental study to illustrate the efficacy of the MPC strategy in the case only the reserved VMs are utilized to serve the dynamically arriving customers.

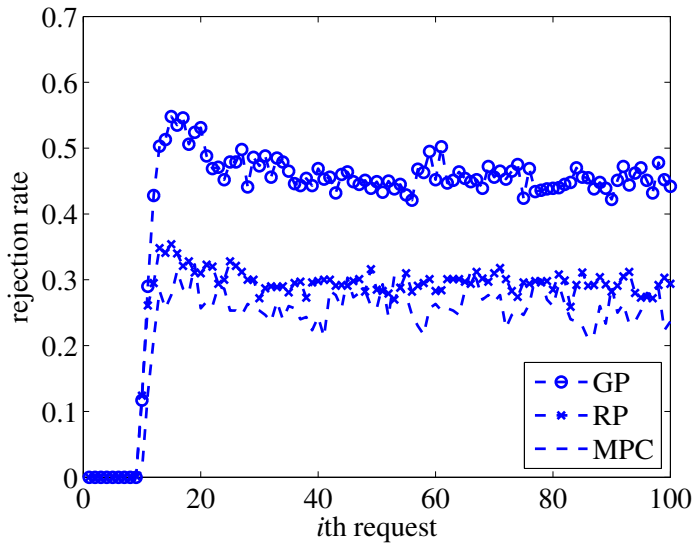
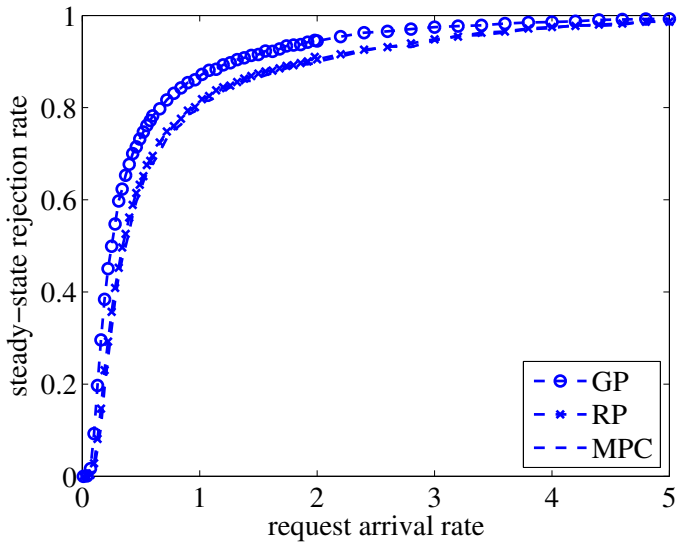


Figure 5.3: The rejection rate for serving 100 customers as a function of the index of the customers

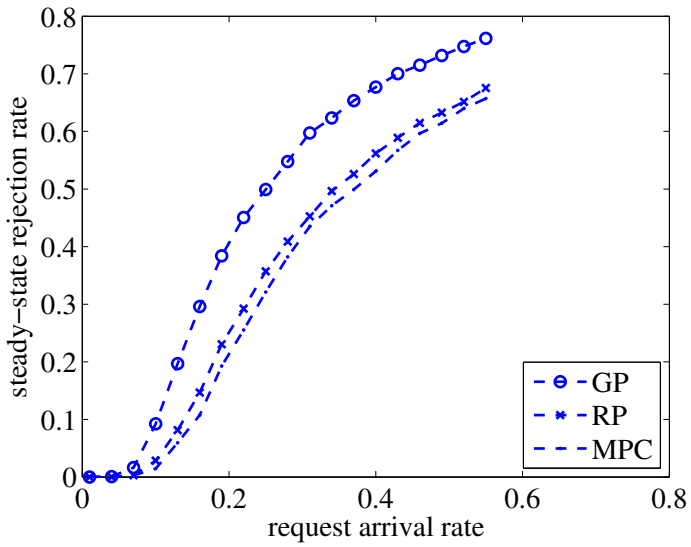
Suppose that the fixed virtual cloud-service infrastructure consisted of 50 reserved VMs, comprising 5 c4.xlarge VMs, 10 c3.large VMs, 15 c1.medium VMs and 20 m1.medium VMs. Customers submit their requests sequentially following a Poisson process with arrival rate λ_c. The rejection rate of the ith customer (averaged over 10,000 realizations) is shown in Fig. 5.3 for $\lambda_c = 0.4$. The three curves represent the results obtained by utilizing the GP strategy, the RP strategy and the MPC strategy, respectively. It can be observed that the rejection rate starts increasing after the service of about 10 requests and then becomes stable after serving 30 requests. The MPC strategy leads to the minimum steady-state rejection rate compared to the other two strategies.

In Fig. 5.4, we further show the steady-state rejection rate for customer requests as a function of the arrival rate, λ_c. It is noted that the MPC strategy again outperforms the other two strategies in terms of reducing the steady-state rejection rate for any λ_c. The GP strategy has the worst performance in this scenario. The difference in the performance of the three strategies is subtle only when $\lambda_c < 0.05$ and $\lambda_c > 4$. This is simply because that either none or all of the customer requests will be rejected if the average time between the arrivals of two requests (namely $1/\lambda_c$) is relatively too long or too short compared to the average response time of a request, respectively.

As seen in Fig. 5.4, the GP strategy in general yields the worst performance by having the largest steady-state rejection rate among the three strategies. It is also noted that the MPC strategy outperforms the RP strategy about 2-5% in the sense of reducing the steady-state rejection rate when $0.1 < \lambda_c < 0.5$. The superiority of the MPC strategy is not as prominent as in the previous scenario when the virtual cloud-service infrastructure is elastic and on-demand VMs are utilized. However, this approach still gives tremendous benefits for this particular scenario, since the cloud service based upon reserved VMs is typically running for long terms, i.e., several years. Reducing the steady-state rejection rate by 5% directly implies an increase in the generated revenue by 5%. Overall, investigation of the steady-state rejection rate in the cloud (as shown in Fig. 5.4) can help businesses better understand how much VMs they need to purchase in order to maintain quality of service and satisfy their customers, which implies keeping the rejection rate to a minimum. For convenience, we show in Fig. 5.5 a flowchart for efficiently serving dynamically arriving customer requests in a fixed virtual cloud-service infrastructure with reserved VMs.



(a)



(b)

Figure 5.4: (a) The steady-state rejection rate (smaller is better) under three resource-provisioning strategies as a function of arrival rate of the customer requests, and (b) the zoomed-in version of Fig. 5.4(a)

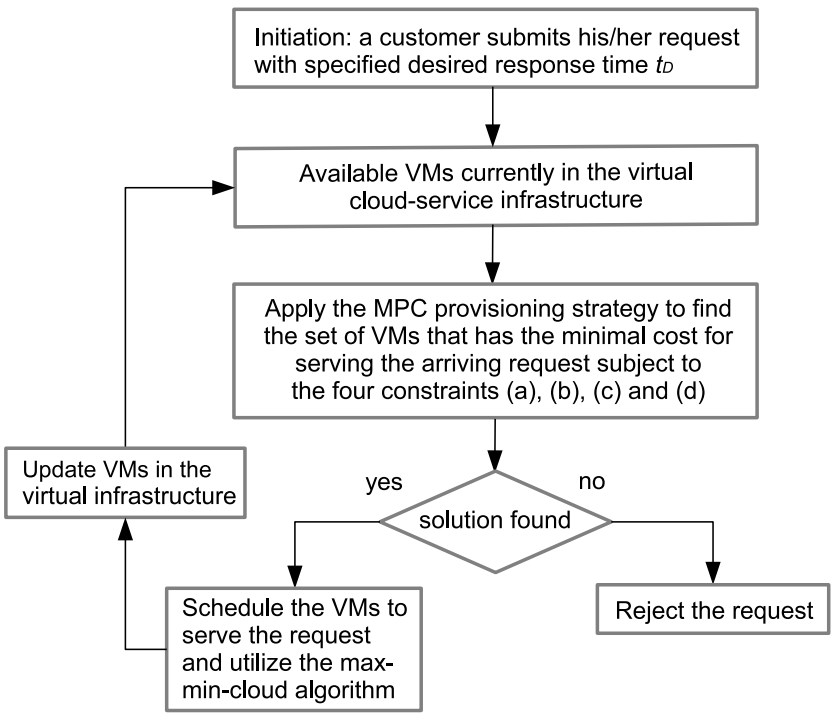


Figure 5.5: Flowchart for efficiently serving dynamically arriving customer requests when the virtual cloud-service infrastructure is fixed by utilizing the reserved VMs

Chapter 6

Smart Virtual Machine Placement

In this chapter, we apply the proposed mult-tenant framework and the max-min-cloud algorithm to investigate the VM placement problem in the cloud as an extension.

6.1 Problem Formulation

Consider a cloud system consisting of certain servers (namely physical machines) a network. The servers are heterogeneous in CPU speed as well as their capacities, including the number of CPU cores, memory size and storage size. When a customer submits his/her request to the system, the request will be served by exactly one virtual machine. We consider a customer's request as an entire workload that can be evenly divided into certain tasks. Similarly as described in Chapter 3.1, a task is the smallest computing task in the cloud system.

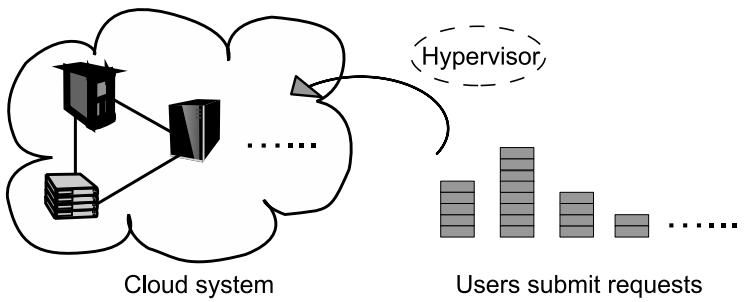
Depending on the workload size of a customer's request, i.e., the number of tasks, we assume that the server virtualiztion software, known as *hypervisor* in the cloud will create an appropriate VM (without a specific type) to serve the customer's

request and map the VM onto a physical server in the system according to a VM-placement algorithm. The physical server must have certain amount of available resources to host such VM depending on the workload size of the customer request to be completed.

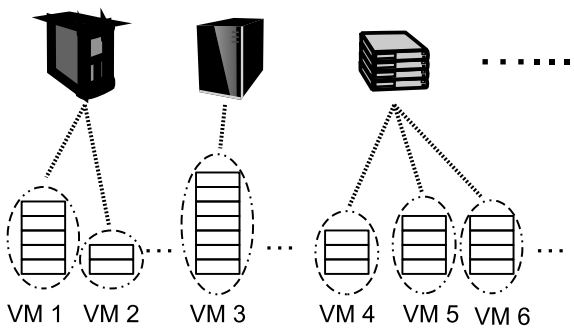
Due to the multi-tenancy support in cloud computing, multiple VMs could be hosted concurrently on a single physical server. As also explained in Chapter 3.1, the time for serving a customer' requests to be stochastic, due to the fact that VMs may exhibit a varying and unstable performance when multiple VMs are concurrently running on the same physical server.

Here, we also assume that certain VM isolation mechanisms are utilized here. As a result, the execution of customer requests are assumed to be mutually independent with a mean proportional to the size of the workload to be served. Furthermore, the new VMs to be hosted on a physical machine would not affect the performance of the VMs that have already been hosted on the physical machine.

To make the VM placement problem more challenging, the performance metric of the cloud service we considered is the mean of stochastic completion time of a group of arbitrary customer requests, (namely the completion time of a batch of VMs) rather than a specific customer request. We assume that these customers submit their requests to the system all at the same time. For example, the group of customers is from a large company, a school or a community. Due to the large granularity of VM resources and the great amount of data transferred in migration and the suspension of VM service, the VM migration is not allowed in this preliminary study. Our goal is to find a suboptimal solution for smartly mapping the batch of VMs to the physical machines in the cloud computing system to obtain a satisfactory completion time.



(a)



(b)

Figure 6.1: The VM placement problem.

6.2 Service Performance Characterization

Given the service environment of the cloud system, we next develop a state-space model to characterize the performance of the cloud service. At any time t, the state of a cloud computing system can be described by the following information: (i) the number of servers in the cloud; (ii) the capacities of the servers; and (iii) the type and number of VMs hosted on each server. Suppose that there are m physical machines in the cloud. We assign a time-varying vector, $\mathbf{r}(t) = [r_1(t), \ldots, r_m(t)]$, to represent the real-time load ratio of the servers in the cloud. The value of $r_j(t)$, for $i = 1, \ldots, m$, denotes the load ratio of server j at t, where $r_j(t) \in [0, 1]$. The load

ratio of a server is defined as the quotient between the consumed memory by all the VMs on the server and the memory capacity of the server. A server can only host a certain number of VMs that is limited by its capacity. If other types of resources, such as the number of CPU cores and storage space, are to be considered we can easily extend the definition of the load ratio to be weighted average of the load ratios for all the resources.

Let the vector $\mathbf{u} = [u_1, ..., u_n]$ denote the requests submitted by a group of arbitrary n customers, where u_i is the workload size of the ith customer's request for $i = 1, \ldots, n$. Given the initial state of the cloud computing system $S_0 = [r_1, ..., r_m]$ before the n customers submit their requests to the cloud system, we proceed to characterize the mean of stochastic completion time of the n customers' requests. We denote such average completion time by $T_{S_0}(\mathbf{u})$. Clearly, $T_{S_0}(\mathbf{u})$ also depends on the VM-placement algorithm, but the explicit reference to this dependence can be omitted from the notation for convenience.

When the n customers submit their requests, the cloud system creates n VMs and maps the n VMs to the physical servers to process the requests. Due to the multi-tenancy characteristic considered in the model, the n VMs are served concurrently with their own stochastic service times and the VM that finishes last determines the completion time. Therefore, $T_{S_0}(\mathbf{u})$ is actually the mean of the maximum of the stochastic service times of the n VMs, i.e.,

$$ET_{S_0}(\mathbf{u}) = \mathsf{E}[\max\{T_{u_1}, ..., T_{u_n}\}], \tag{6.1}$$

where T_{u_i} is the stochastic execution time (with rate λ_{u_i}) of the ith VM.

6.2.1 Impact of a Server's Load Ratio on the Processing Rate of a VM

Due to the existence of performance interference between VMs hosted in the same physical machine, we assume that as the consumed resource of a physical server is approaching its capacity, i.e., as more VMs are hosted by the physical server concurrently, the performance and efficacy of the server will encounter bottlenecks. To model such behavior, we assume here that it takes more time for a server to process a job when it is heavily loaded than that for the case when the server is lightly loaded.

Specifically, suppose that the current load ratio of the jth server is r_j and the ith VM is mapped to the jth server. To capture the impact of the jth server's load ratio on the processing rate of the ith VM, we propose the following simple model

$$\lambda_{ij} = \frac{\lambda_{s_j}}{u_i} d(r_j), \tag{6.2}$$

where λ_{ij} is the execution rate of the ith VM is when it is mapped to the jth server. The term λ_{s_j} is the CPU speed of server j, and $d(\cdot)$ is a decreasing function that reflects the reduction of the processing rate resulting from the load ratio of server j. An example of $d(\cdot)$ is

$$d(r_j) = \begin{cases} 1 & \text{if } 0 \leq x \leq 0.5 \\ 0.75 - \arctan(40r_j - 30)/6 & \text{if } 0.5 < x \leq 1 \end{cases}, \tag{6.3}$$

which is shown in Figure 6.2.

6.3 The Max-Load-First Algorithm

In this section, we aim to devise a heuristic VM-placement algorithm that can result in obtaining the minimum completion time of the batch of n VMs. We assume

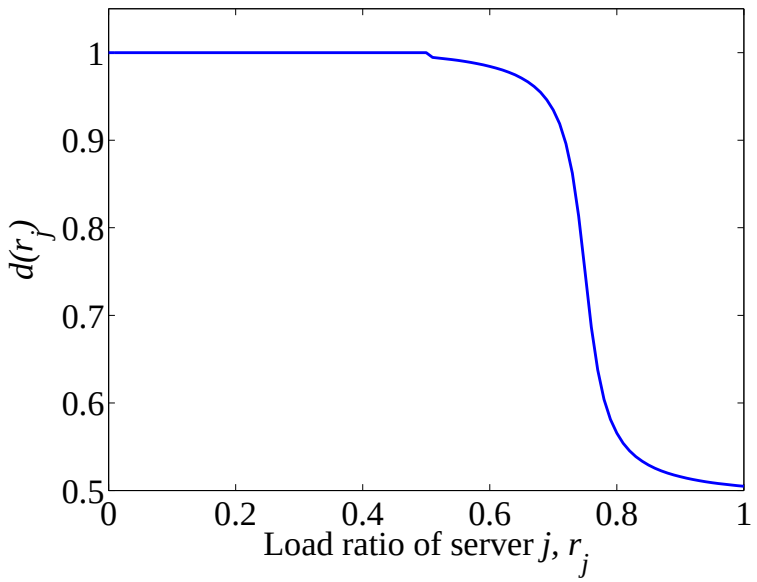


Figure 6.2: The decreasing function $d(\cdot)$ models the impact of r_j on the processing rate, λ_{u_i}, of the ith VM.

that the n VMs are allocated sequentially. To this end, there are two issues to be considered for devising the VM-placement algorithm. The first issue is the decision on which server is the best one for each incoming VM to be mapped. The second issue is the placement order of the n VMs.

To address the first issue, each VM should be mapped to the fastest physical server, since the VMs are no longer queued but are served concurrently in the cloud system. To address the second issue, we consider three ordering strategies for placement the VMs in terms of the workload size they serve. Specifically, the three ordering strategies are as follows: 1) from the largest workload size to the smallest workload size; 2) from the smallest workload size to the largest workload size; and 3) random ordering. Based on the three VM-ordering strategies described above, we can propose three algorithms, respectively. In light of the max-min-cloud algorithm proposed in Chapter 4.1, we intuitively believe that the first VM-ordering strategy

would lead to the best performance. We will then compare the efficacy of the three algorithms by means of MC simulations to verify our guess.

As an extension of the max-min-cloud algorithm, we elaborate the max-load-first algorithm in Algorithm 2. The max-load-first algorithm is different and more complicated than the max-min-cloud algorithm in three aspects. First, there is a memory capacity limit of physical servers to be checked for mapping the VMs; while the max-min-cloud has the limit on the number of vCPUs in VMs. Secondly, a pre-calculation of the execution rates of the VMs after the mapping has to be obtained in order to determine which physical machines is the best one to choose among all the candidates. On the contrary, only a sorting of VMs based on ECU per CPU is needed in the max-min-cloud algorithm. Thirdly, after a mapping of a new VM the load ratio of the selected physical machine needs to be updated.

Similarly, we can also propose the min-load-first algorithm and the random-load algorithm based on the other two VM-ordering strategies. Note that the concepts of the three algorithms proposed above are similar to the max-min, min-min and OLB algorithms that have been proposed in [29]. However, the three algorithms proposed here are modified and generalized to fit the cloud environment described in Chapter 6.1. Furthermore, our proposed algorithms exhibit a totally different performance trend compared to those reported in [29] because our model is based on multi-tenant principle whereas the algorithms in [29] are based on queuing models.

6.4 Simulation Results

With the three VM-placement algorithms at hand, we first present MC simulation results to compare the performance of the three algorithms. The algorithm that leads to the smallest average-completion-time metric implies the best efficacy.

Algorithm 2 The max-load-first algorithm

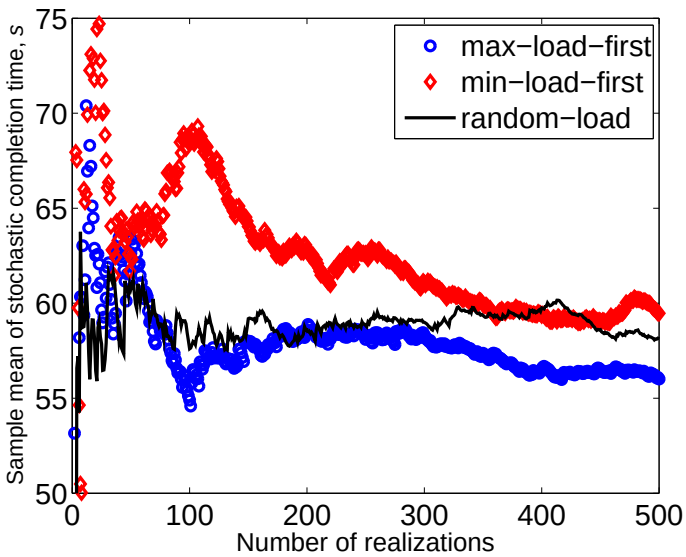
order the n VMs from the largest to the smallest based on their workload size;
use a vector $\mathbf{V} = [v_1, ..., v_n]$ to restore the ranked VMs;
for $i = 1$ **to** n **do**
  choose the ith VM, v_i;
  for $j = 1$ **to** m **do**
    if the jth server does not have enough memory to host v_i, **then**
      $\lambda_{ij} \leftarrow 0$;
    else
      $\lambda_{ij} \leftarrow \lambda_j d(r_j)$;
    end if
  end for
  $j_i^* = \underset{j=1:m}{\text{argmax}}[\lambda_{i1}, ..., \lambda_{ij}, ..., \lambda_{im}]$;
  allocate v_i to the j_i^*th server;
  update the load ratio, $r_{j_i^*}$, of the j_i^*th server;
end for

For a better comparison of the performance of the three algorithms, the cloud system simulated in this section is similar to a private cloud, which is consisted of small number of physical servers with limited memory capacities, rather than a public cloud that is assumed to have unlimited resources. Specifically, without loss of generality, the simulated cloud system is composed of three heterogeneous physical servers. The CPU speeds of the three servers are 1, 1.2 and 1.5 GHz, respectively. The memory capacities of the servers are 1 GB, 2 GB and 4 GB, respectively. At $t = 0$, we assume that a group of 10 customers submit their requests to the system, and the requests are consisted of 500, 450, 400, 150, 125, 100, 50, 40, 30 and 20 tasks, respectively. We also assume that each job requires 1 MB for a physical server to host such job and ideally it takes on average 0.1 second for the server with 1GHz CPU speed to process a task.

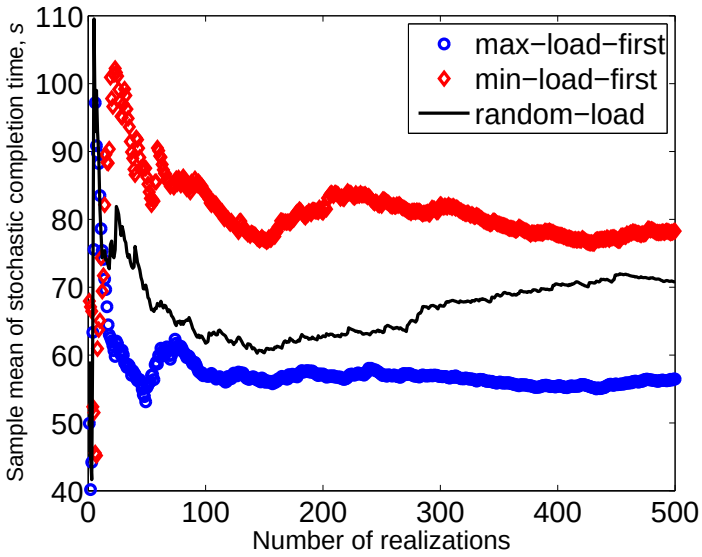
In Figure 6.3(a), the sample mean of stochastic completion time of the 10 customers' requests is compared among the three algorithms. In this scenario, all the servers have the same initial load ratio of 15%. The sample mean curves may exhibit unpredictable fluctuations especially for the first dozens of realizations because the stochastic service times of VMs are assumed to be exponentially distributed and they could be highly different in each realization. However, the sample mean curves become stable asymptotically after 500 realizations of the experiment. Clearly, the max-load-first algorithm has the best performance among the three algorithms, while the min-load-first algorithm results in the largest mean of stochastic completion time. Similar results have also been shown in Figure 6.3(b) and (c), where the initial load ratios of all the servers are 25% and 50%. This is because that the mean of stochastic service time of the largest customer's request (500 jobs) has the most impact on the mean of stochastic completion time of all the customers' requests. According to our modeling of the impact of a server's load ratio on the processing rate of a VM proposed in Chapter 6.2.1, given the same initial system state the mean of stochastic service time of the largest customer's request when applying the max-load-first algorithm will be smaller than that for the scenario when applying the min-load-first or the random-load algorithms. As such, the max-load-first algorithm will result in the minimum average-completion-time metric among the three algorithms.

After finding out the supremacy of the max-load-first algorithm, we further study how the initial load ratio of all the servers affects the performance of the max-load-first algorithm. Figure 6.4 shows the mean of stochastic completion time of the 10 customers' requests as a function of initial load ratio of the servers. Each point on the curve is a average over 2000 realizations. It is clear that as the load ratio increases the performance deteriorates. It is also interesting that there appears to be a phase transition in the slope of the curve. In particular, when the load ratio rises beyond 28% the slope of the curve increases abruptly. This observation suggests that it is critical to keep the load ratio under a certain threshold in order to maintain a

good performance. Such threshold mainly depends on the memory capacities of the servers and the total number of jobs of the customers in the group.

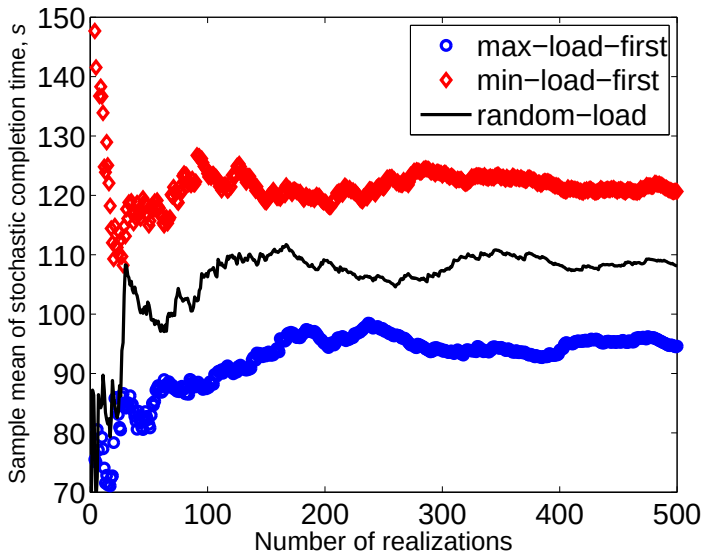


(a)



(b)

Figure 6.3: The sample mean of stochastic completion times of the 10 customers' requests applying the three algorithms when the three servers are initially loaded at (a) 15% and (b) 25%.



(c)

Figure 6.3: The sample mean of stochastic completion times of the 10 customers' requests applying the three algorithms when the three servers are initially loaded at (c) 50%.

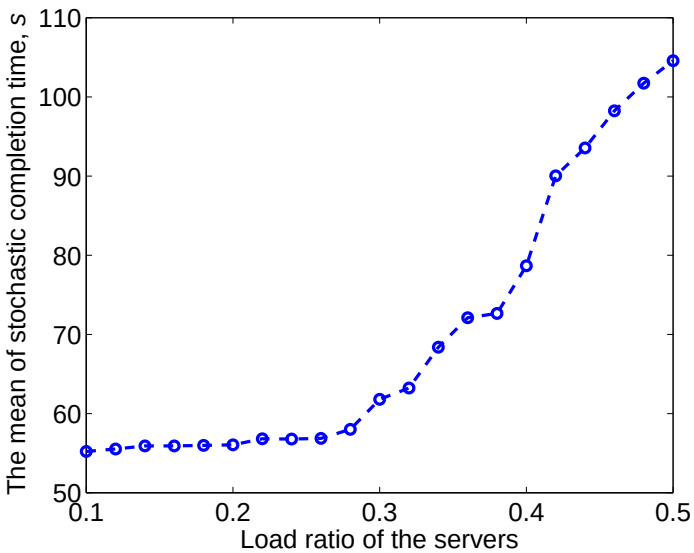


Figure 6.4: The mean of stochastic completion time of 10 customers' requests as a function of initial load ratio of the three servers applying the max-load-first algorithm.

Chapter 7

Conclusions

In this dissertation we have proposed a novel probabilistic multi-tenant framework to model the service of customers in the cloud. The model considers essential features and concerns in modern cloud services including server virtualization, multi-tenancy, heterogeneity of VMs in the virtual infrastructure, as well as security and performance isolation mechanisms. To this end, the percentile and mean of the stochastic response times of customer requests have been analytically characterized. These two quantities are widely-used metrics for evaluating the performance of cloud services in the research community as well as cloud SLAs.

Based upon the proposed cloud-service framework, we have devised a max-min-cloud algorithm for allocating the workloads in an arriving request to VMs. We have rigorously proved the optimality of the max-min-cloud algorithm and further conducted extensive MC simulations to demonstrate its optimality under various scenarios.

In light of the max-min-cloud algorithm, we also devised an efficient resource-provisioning strategy, termed the MPC strategy, for determining the appropriate amount of computing resources in the cloud required to serve dynamically arriving

customer requests. Our practical case study shows that utilizing the MPC strategy can yield a 10 - 40% increase in profit to businesses compared to other resource-provisioning strategies when the cloud service is based upon on-demand VMs. On the other hand, when the cloud service is based upon reserved VMs with fixed cost, we have found that the MPC strategy outperforms the other strategies by accepting 2 - 20% more requests when customers are submitting their requests with a normal pace.

As a byproduct of the max-min-cloud algorithm, we proposed a VM-placement algorithm, termed the max-load-first algorithm, to improve the performance of the cloud system. The supremacy of the max-load-first algorithm has been shown by means of MC simulations in comparison to other two algorithms. We have also found the presence of critical threshold of servers' load ratios beyond which the average-completion-time metric increases abruptly. The findings in our preliminary investigation on the VM placement problem could be of great benefit to cloud providers in the sense of smartly scheduling the VMs/requests as well as efficiently managing and updating the physical infrastructure that is used to support cloud computing, which, in turn, optimizes the performance of their cloud services.

Finally, we would like to emphasize that the results presented in this dissertation work are not limited to the case when cloud-service performance is dependent on the ECU (i.e., vCPU speed) of the VMs. Namely, our framework can also be extended to scenarios when the cloud-service performance is determined and affected by other factors, such as network bandwidth, energy consumption, memory capacity and storage space. For example, consider the data transmission service in the cloud where data throughput is the key performance metric. In this case, the throughput is mainly dependent upon the network bandwidth of the VMs that are used to process and transfer data. To improve the utilization of the network bandwidth, the network bandwidth in a VM is typically shared by several workloads/tenants. Hence, we can

replace ECU with network bandwidth in the proposed multi-tenant framework to characterize the cloud-service performance. Here the max-min-cloud algorithm is also useful for optimizing the utilization of network bandwidth in the VMs. For the case when energy consumption is a concern, the MPC strategy can also be extended to include additional constraints on the energy consumption of the VMs.

References

[1] P. Mell and T. Grance, "The nist definition of cloud computing," *NIST Special Publication 800-145*, September 2011.

[2] K. Xiong and H. Perros, "Service performance and analysis in cloud computing," in *Proc. IEEE World Conf. Services*, 2009, pp. 693–700.

[3] B. Yang, F. Tan, Y. Dai, and S. Guo, "Performance evaluation of cloud service considering fault recovery," in *Proc. First Intl Conf. Cloud Computing (CloudCom 09)*, 2009.

[4] H. Khazaei, J. Mišić, and V. B. Mišić, "Performance analysis of cloud computing centers using m/g/m/m+r queueing systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 23, no. 5, pp. 936–943, 2012.

[5] B. Yang, F. Tan, and Y.-S. Dai, "Performance evaluation of cloud service considering fault recovery," *J. Supercomput.*, vol. 65, pp. 426–444, 2013.

[6] S. Yeo and H. Lee, "Using mathematical modeling in provisioning a heterogeneous cloud computing environment," *Computer*, vol. 44, no. 8, pp. 55–62, 2011.

[7] T. Sridhar, "Cloud computing–a primer," *The Internet Protocol Journal*, vol. 12, no. 3, 2009.

[8] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, 2010.

[9] I. Foster *et al.*, "Cloud computing and grid computing 360-degree compared," in *Proc. Grid Computing Environments Workshop GCE'08*, 2008.

[10] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in xen," in *Middleware 2006*. Springer, 2006.

[11] X. H. Guo *et al.*, "Spin: Service performance isolation infrastructure in multi-tenancy environment," *Service-Oriented ComputingICSOC*, 2008.

[12] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of network and computer applications*, vol. 34, no. 1, pp. 1–11, 2011.

[13] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu, "Resource provisioning for cloud computing," in *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, 2009, pp. 101–111.

[14] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *Services Computing, IEEE Transactions on*, vol. 5, no. 2, pp. 164–177, 2012.

[15] A. Iosup *et al.*, "Performance analysis of cloud computing services for many-tasks scientific computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 6, pp. 931–945, 2011.

[16] Z. Wang, M. M. Hayat, N. Ghani, and K. B. Shaban, "A probabilistic multi-tenant model for virtual machine allocation in cloud systems," in *the 2014 3rd IEEE International Conference on Cloud Networking (CloudNet)*, 2014.

[17] ——, "Optimizing cloud-service performance: Efficient resource provisioning via optimal workload allocation," *submitted to IEEE Trans. Parallel and Dist. Systems*.

[18] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[19] Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/.

[20] Google ComputeEngine, https://cloud.google.com/compute/.

[21] Microsoft Azure, http://azure.microsoft.com/.

[22] Y. Koh *et al.*, "An analysis of performance interference effects in virtual environments," in *Performance Analysis of Systems and Software, 2007. ISPASS 2007. IEEE International Symposium on*, 2007.

[23] X. Pu *et al.*, "Understanding performance interference of i/o workload in virtualized cloud environments," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, 2010.

[24] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: managing performance interference effects for qos-aware clouds," in *Proceedings of the 5th European conference on Computer systems*. ACM, 2010, pp. 237–250.

[25] K. Xiong and H. Perros, "Sla-based resource allocation in cluster computing systems," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE, 2008, pp. 1–12.

[26] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

[27] A. L. Rosenberg and R. C. Chiang, "Toward understanding heterogeneity in computing," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on.* IEEE, 2010, pp. 1–10.

[28] M. Rahman, X. Li, and H. Palit, "Hybrid heuristic for scheduling data analytics workflow applications in hybrid cloud environment," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on.* IEEE, 2011, pp. 966–974.

[29] T. Braun *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.

[30] M.-Y. Wu, W. Shu, and H. Zhang, "Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems," in *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000).* IEEE, 2000, p. 375.

[31] G. Ritchie and J. Levine, "A fast, effective local search for scheduling independent jobs in heterogeneous computing environments," *Technical report, Centre for Intelligent Systems and their Applications, School of Informatics, University of Edinburgh*, 2003.

[32] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *Proc. IEEE INFOCOM*, 2012, pp. 702–710.

[33] M. Cardosa, M. R. Korupolu, and A. Singh, "Shares and utilities based power consolidation in virtualized server environments," in *Integrated Network Management, 2009. IM'09. IFIP/IEEE International Symposium on.* IEEE, 2009, pp. 327–334.

[34] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 2009, pp. 41–50.

[35] J. Hu, J. Gu, G. Sun, and T. Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," in *Third International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*, 2010.

[36] H. Liu, S. Liu, X. Meng, C. Yang, and Y. Zhang, "Load balancing strategy for virtual storage," in *IEEE International Conference on Service Sciences (ICSS)*, 2010.

[37] A. Bhadani and S. Chaudhary, "Performance evaluation of web servers using central load balancing policy over virtual machines on cloud," in *Proceedings of the Third Annual ACM Bangalore Conference (COMPUTE)*, 2010.

[38] J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*. IEEE, 2010, pp. 179–188.

[39] N. J. Kansal and I. Chana, "Cloud load balancing techniques: A step towards green computing," *International Journal of Computer Science Issues*, vol. 9, no. 1, pp. 238–246, 2012.

[40] K. A. Nuaimi, N. Mohamed, M. A. Nuaimi, and J. Al-Jaroodi, "A survey of load balancing in cloud computing: Challenges and algorithms," in *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*, 2012.

[41] G. Juve and E. Deelman, "Resource provisioning options for large-scale scientific workflows," in *eScience, 2008. eScience'08. IEEE Fourth International Conference on.* IEEE, 2008, pp. 608–613.

[42] J. Yang, J. Qiu, and Y. Li, "A profile-based approach to just-in-time scalability for cloud applications," in *Cloud Computing, 2009. CLOUD'09. IEEE International Conference on.* IEEE, 2009, pp. 9–16.

[43] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific.* IEEE, 2009, pp. 103–110.

[44] M. Mao and M. Humphrey, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on.* IEEE, 2013, pp. 67–78.

[45] Q. Li and Y. Guo, "Optimization of resource scheduling in cloud computing," in *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing.* IEEE, 2010, pp. 315–320.

[46] G. Feng, S. Garg, R. Buyya, and W. Li, "Revenue maximization using adaptive resource provisioning in cloud computing environments," in *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing.* IEEE Computer Society, 2012, pp. 192–200.

[47] L. Zhao, S. Sakr, and A. Liu, "A framework for consumer-centric sla management of cloud-hosted databases," *Services computing, IEEE Transactions on*, vol. 8, no. 4, pp. 534–549, 2015.

[48] R.-H. Hwang, C.-N. Lee, Y.-R. Chen, and D.-J. Zhang-Jian, "Cost optimiza-

tion of elasticity cloud resource subscription policy," *Services Computing, IEEE Transactions on*, vol. 7, no. 4, pp. 561–574, 2014.

[49] Y. Ran, J. Yang, S. Zhang, and H. Xi, "Dynamic iaas computing resource provisioning strategy with qos constraint," *Services Computing, IEEE Transactions on*.

[50] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*, 2009, pp. 44–51.

[51] M. Armbrust *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[52] L. Wu, S. K. Garg, and R. Buyya, "Sla-based resource allocation for software as a service provider (saas) in cloud computing environments," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*. IEEE, 2011, pp. 195–204.

[53] R. Buyya *et al.*, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[54] P. Patel, A. H. Ranabahu, and A. P. Sheth, "Service level agreement in cloud computing," in *Cloud Workshops at OOPSLA09, Orlando, Florida, USA*, 2009.

[55] I. Foster, "Designing and building parallel programs," 1995.

[56] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 871–879, 2011.

[57] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Cloud Computing*. Springer, 2009, pp. 254–265.

[58] K. Boloor, R. Chirkova, Y. Viniotis, and T. Salo, "Dynamic request allocation and scheduling for context aware applications subject to a percentile response time sla in a distributed cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 464–472.

[59] W. Iqbal, M. Dailey, and D. Carrera, "Sla-driven adaptive resource management for web applications on a heterogeneous compute cloud," in *Cloud Computing*. Springer, 2009, pp. 243–253.

[60] H. N. Van, F. D. Tran, and J.-M. Menaud, "Sla-aware virtual resource management for cloud infrastructures," in *Computer and Information Technology, 2009. CIT'09. Ninth IEEE International Conference on*, vol. 1. IEEE, 2009, pp. 357–362.

[61] K. H. Kim, A. Beloglazov, and R. Buyya, "Power-aware provisioning of cloud resources for real-time services," in *Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science*. ACM, 2009.