Optimised Auto-scaling for Cloud-based Web Service

Jing Jiang

FACULTY OF ENGINEERING AND INFORMATION TECHNOLOGY

UNIVERSITY OF TECHNOLOGY SYDNEY

A thesis submitted for the degree of

Doctor of Philosophy

February 2015

CERTIFICATE OF AUTHORSHIP/ORIGINALITY

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Candidate

_____

Acknowledgements

This thesis is the fruit of my hard work during the whole of my PhD study since being in Australia. It has not been a straight path but it has been an amazing journey in which I have experienced happiness, excitement, depression, struggle and friendship. I feel quite lucky that so many people have offered me a helping hand in this journey. Now is the right time to thank all of them!

First, I would like to express my sincere appreciation and great gratitude to my principal supervisor, Professor Jie Lu, and my co-supervisor, Professor Guangquan Zhang. They are not only mentors in my research life but also the guides in my everyday life, especially in my first year in Australia. In my research work, they offered me ample trust and free space to choose the topic and never doubted my progress, but always made valuable suggestions if I encountered problems.They also guided me on how to express my research results efficiently including how to organise a paper, the right steps to take in writing, and even how to express ideas clearly in sentences. They both always work so

hard and have a strict attitude to research work, which stimulates me to further efforts and from which I will benefit throughout my life. Besides the research work, they were enormously helpful and caring and never said "No" to my requests - for example, they drove to pick up me at the airport on the first day I arrived in Sydney; made suggestions for renting a house; taught me to cook; invited me to their home for celebrations, and even guided me in managing work and life. There are no words to express all my thanks to them; the only way to repay them was to work hard.

I kindly thank all my colleagues in the Decision Systems and e-Service Intelligent (DeSI) lab for their careful participation in my presentation and valuable comments for my research. I also thank many members of the Centre for Quantum Computation and Intelligent Systems (QCIS) and the Faculty of Engineering and Information Technology, University of Technology, Sydney (UTS) for their various assistance and advice which have been of great benefit to this study. I appreciate the financial support from the International Postgraduate Research Scholarship (IPRS) and UTS President's Scholarship (UTSP).

I appreciate the conference and research travel support by Prof Chengqi Zhang, director of QCIS, the UTS Vice Chancellor's conference fund,

and the Faculty of Engineering and Information Technology travel fund.

I sincerely thank Ms. Sue Felix and Ms. Barbara Munday for polishing my English language and doing proofreading for my publications and this thesis; their comments have greatly helped me to improve my academic writing. I would also sincerely thank Ms.Teraesa Ashworth who spent much time introducing me to local culture and helping to improve my spoken English.

I would like to express my earnest appreciation and gratitude to my parents for all their selfless love and generous understanding, also to my brothers and sisters for their warm concern and for looking after our parents when I was not around for all those years.

Last but certainly not least certainly not the least, I feel extremely fortunate that I could share the joy and the pain with my dear husband, Guodong Long. He has been by my side during hard times and has comforted me and encouraged me to continue. I am grateful for his love and endless optimism at times when the barriers seemed impossible to break down.

Abstract

Elasticity and cost-effectiveness are two key features for ensuring that cloud-based web services appeal to more businesses. However, true elasticity and cost-effectiveness in the pay-per-use cloud business model has not yet been fully achieved. The explosion of cloud-based web services brings new challenges to enable the automatic scaling up and down of service provision when the workload is time-varying.

This research studies the problems associated with these challenges. It proposes a novel scheme to achieve optimised auto-scaling for cloud-based web services from three levels of cloud structure: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). At the various levels, auto-scaling for cloud-based web services has different problems and requires different solutions.

At the SaaS level, this study investigates how to design and develop scalable web services, especially for time-consuming applications. To achieve the greatest efficiency, the optimisation of service provision problem is studied by providing the minimum functionality and fastest

scalability performance concerning the speed-up curve and QoS (Quality of Service) of the SLA (Service-Level Agreement). At the PaaS level, this work studies how to support dynamic re-configuration when workloads change and the effective deployment of various kinds of web services to the cloud. To achieve optimised auto-scaling of this deployment, a platform is designed to deploy all web services automatically with the minimal number of cloud resources by satisfying the QoS of SLAs. At the IaaS level for two infrastructure resources of virtual machine (VM) and virtual network (VN), this research focuses on studying two types of cloud-based web service: computation-intensive and bandwidth-intensive. To address the optimised auto-scaling problem for computation-intensive cloud-based web service, data-driven VM auto-scaling approaches are proposed to handle the workload in both stable and dynamic environments. To address the optimised auto-scaling problem for bandwidth-intensive cloud-based web service, this study proposes a novel approach to predict the volume of requests and dynamically adjust the software defined network (SDN)-based network configuration in the cloud to auto-scale the service with minimal cost.

This research proposes comprehensive and profound perspectives to

solve the auto-scaling optimisation problems for cloud-based web services. The proposed approaches not only enable cloud-based web services to minimise resource consumption while auto-scaling service provision to achieve satisfying performance, but also save energy consumption for the global realisation of green computing. The performance of the proposed approaches has been evaluated on a public platform (e.g. Amazon EC2) with the real dataset workload of web services. The experiment results demonstrate that the proposed approaches are practicable and achieve superior performance to other benchmark methods.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing has achieved great success as a service oriented computing paradigm and is rapidly becoming the next generation computing environment in which the resources of storage, computation, network and applications are provided as services through the Internet. Moreover, this new computing paradigm facilitates the delivery of computing as an on-demand service, or "pay-as-you-go", similar to the delivery of other public utilities, such as electricity and gas. Major commercial cloud vendors have achieved great success by providing flexible "pay-as-you-go" or "on-demand" services with various service instance types and plans to meet customers' computing needs, supported by a large number of distributed cloud resources. For example, Amazon EC2 provides three families of cloud computing resources (on-demand, reserved, spot), and each family includes several service instance types. Various service instance types operate at different

quality of service (QoS) levels and have variable prices.

These pay-as-you-go or on-demand cloud services can be categorised into Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). In SaaS business models, users can access software applications and databases. SaaS is also referred to as "on-demand software" and is usually priced on a pay-per-use basis using a subscription fee. In PaaS models, cloud providers deliver computing platforms as services which typically include the operating system, programming language execution environment, database, and web servers. To deploy and develop software applications on the PaaS platform, the complex management environment for underlying hardware and software is not considered. The most popular service model for IaaS providers is to offer computing resources via virtual machines, according to the IETF (Internet Engineering Task Force). With the newly emerged Software Defined Network (SDN) techniques, network resources can be provided as new infrastructure resource services.

A web service is a communication method between two electronic devices over a network and is widely used in web applications. With the widespread development of various application frameworks, different modules within the same web application usually communicate with each other via web services. Thus the scalable problem is currently becoming the main challenge for web service delivery. Requests for web services mainly come from Internet users, and the workloads

of web services are therefore time-varying and visit bursts for web applications inevitably occur. To address these challenges, web services are required to be auto-scalable with time-varying workloads.

To address the auto-scaling problem and achieve optimal performance on SaaS, there are two main resource optimisation strategies. One is to allocate a minimal amount of computational resource to accomplish computation tasks by considering the given time limitation and computation complexity. This strategy relates to parallel programming and optimisation algorithms. Another strategy is to allocate a minimal amount of resource to tackle the numerous requests by considering SLAs (Service Level Agreements). This method cannot be solved at a separate SaaS level alone, as it involves in-platform deployment and infrastructure resource allocation.

PaaS can be considered as a large container for deploying different kinds of services and algorithms. Given a time-varying workload for various services and algorithms, how to allocate the minimum number of servers and how to configure each server's platform to tackle the changing workload while satisfying SLAs becomes a big challenge. Under these circumstances, the configuration optimisation platform must be adaptable and self-configurable in order to continuously meet QoS requirements for applications with workload intensity.

In IaaS business models, cloud operational systems can support large numbers of virtual machines (VMs) and are able to scale services up and down according to

users' requirements. VMs are one of the most important infrastructure resources for cloud services. During each time unit, service users are charged by the number of VMs consumed, therefore service users are motivated by pursuing minimal cost with optimal numbers of VMs while satisfying performance requirements. With the emergence of new SDN techniques, therefore the consumed network resources are able to be allocated as "on-demand" services. When the SDN is integrated into cloud computing, the network resource will become a new scalable cloud service. The allocation of optimal network resource to satisfy bandwidth-intensive cloud online application providers, such as Video-on-Demand (VoD) providers, is a new challenge.

1.1 Research Questions

This research aims to optimise resource allocation for auto-scaling web services in the cloud computing environment. This kind of service is defined as an **Auto-scaling Cloud-based Web Service (ACWS)** in this thesis. This research focuses on resource auto-scaling optimisation from the three perspectives of SaaS, PaaS and IaaS. In particular, this work studies the computation resource and network resource separately at the IaaS level. The research problems are drawn out as follows:

Q1: How should an optimised auto-scaling application from the the

SaaS level for cloud-based web services be designed?

For time-consuming applications, web services are required to be auto-scalable to achieve efficient accomplishments within a given time by increasing the computation resource (i.e.servers). In cloud-based web services, auto-scaling services aim to minimise the cost of the resource while satisfying the performance requirement (i.e. the QoS of SLAs). A successful cloud web service contains two components: an optimally designed parallel cloud application and a self-adaptive resource allocation system. This question focuses on the first component, and the following three questions address the second component.

Q2: How should an optimised auto-scaling platform from the PaaS level for cloud-based web services be designed?

A cloud web service platform needs to provide services to satisfy time-varying workload visits. A self-adaptive method is required to allocate the optimised amount of resource to tackle time-changing workloads while satisfying QoS. Moreover, the platform needs to provide multiple kinds of web service. The question addresses how to auto-configure these services to the platform to achieve optimal cost and satisfactory performance.

Q3: How should optimised auto-scaling resource allocation from the IaaS level for computation-intensive cloud-based web services be designed?

Computation resource is one of the most important infrastructure resources in

the cloud computing system. For computation-intensive cloud web services, optimised resource allocation achieves a trade-off between cost and QoS. How to predict the workload and how to estimate the optimal volume of resource to satisfy QoS are the new challenges. To simplify the situation, this study only considers one kind of web service, namely the single-service scenario. The scenario of multiple kinds of service can be solved by integrating the result of Question 2 with the solutions of the single-service scenario.

Q4: How should optimised auto-scaling resource allocation from the IaaS level for bandwidth-intensive cloud web service be designed? Network resource is another important infrastructure resource in a cloud computing system, especially for bandwidth-intensive cloud web services, such as VoD (Video-on-Demand). Similar to Question 3, web service visits need to be predicted in advance, and then allocated on the optimal volume of resource from private clouds or public clouds. The optimisation value is also a trade-off between the cost and the QoS.

1.2 Research Objectives

A web service system consists of web service programs, service configuration platform, and the necessary infrastructure resource, e.g. VMs and networks. To enable cloud-based web service systems to be auto-scaling, these web service

components are allocated at different levels of cloud paradigm. Particularly, the service program is set up at the level of SaaS, the configuration platform can be allocated at the level of PaaS, and the infrasructure resource components are related to the level of IaaS. This study thus proposes four primary objectives based on this allocation consideration and the identified research questions:

Obj1: To enable application auto-scaling for time-consuming cloud-based web services (aims to answer Q1);

This research will investigate how to develop the scalability of web service applications in the cloud computing environment. In particular, this work will carry on the research work from a case study - a time-consuming web application (i.e. recommendation application) - and will implement the application on a popular cloud platform, Hadoop. Once the scalable program has been developed, the optimised resource allocation for this cloud web service can then be solved.

Obj2: To develop a self-adaptive configuration optimisation platform for cloud-based web services (aims to answer Q2);

Considering that workloads are time-varying, the platform is required to self-adaptively allocate resources to tackle the requests. The challenge of optimised resource allocation is how to make a trade-off between the cost and QoS. Because the platform needs to support multiple kinds of service, another challenge is how to configure the services on the platform, particularly in respect of deploying services on different servers to minimise the cost and satisfy the QoS of

the SLA. In summary, the platform is required to be a self-adaptive configuration optimisation system.

Obj3: To propose an optimisation methodology to allocate computational resources for auto-scaling computation-intensive cloud-based web services (aims to answer Q3);

The computation-intensive web service provider can utilise cloud computing for auto-scaling; the challenge is how to define an optimal scaling method by leveraging cost and QoS. To make the methodology practicable, this work forecasts the workload by integrating time series analysis and predictive-based techniques. Queueing theory and optimisation techniques are studied to estimate the optimal volume of computation resource.

Obj4: To propose a optimisation methodology for allocating network resources for auto-scaling bandwidth-intensive cloud web service (aims to answer Q4);

The SDN-based cloud infrastructure resource enables bandwidth-intensive web service providers to self-adaptively allocate network resources. To develop time-varying adaptive optimisation algorithms, requests for web services need to be predicted and the optimal value for the network resource must be evaluated. With this method, web service providers can minimise their network cost while satisfying the QoS of SLAs.

1.3 Research Significance

The main significance of the proposed work can be summarised from the following three aspects:

- **This work provide a novel solution to the auto-scaling problems of cloud-based web services from multiple levels of cloud computing**

  This study proposes an overall solution for auto-scaling cloud-based web services by exploring the research questions at three levels: SaaS, PaaS and IaaS. At the SaaS level, the auto-scaling solution focuses on efficiently scaling the program of the application. At the PaaS level, the work develops an auto-configuring platform for multiple kinds of web services to achieve service auto-scaling. At the IaaS level, the auto-scaling solution concentrates on the optimised allocation for infrastructure resources.

- **Optimised resource allocation can save cost and reduce energy consumption**

  The optimised resource allocation solution can directly save cost for web service providers. Moreover, the saved energy consumption not only contributes to savings for service providers but also benefits the global environment. The optimisation solutions achieve the best trade-off between cost and QoS. All optimisation results incur minimal cost while satisfying the QoS of SLAs.

- **Self-adaptive platform can fit the time-varying workload scenario of cloud-based web services**

  The self-adaptive platform can change the volume of allocated resources to tackle the time-varying workload of cloud-based web services in real time. This self-adaptive platform can also configure multiple kinds of service to the allocated servers so that the utilisation of resources is high and the system is robust for the visit burst scenario of web services.

- **This work allocates optimal resources combining prediction, modelling and optimisation**

  To achieve the best performance and robustness, the system not only proactively allocates resources, but also adjusts resource in real time corresponding to the real changed workload. To forecast the workload, this study develops a data-driven perdition method by integrating time series analysis with machine learning techniques. Based on the predicted results, the visit workloads, allocated resource and performance are modelled as a queueing theory system to achieve a trade-off between cost and QoS.

1.4 Research Methodology

This section presents the designed methodology for the study, in which the problem is first confirmed and the objectives identified, and then a solution is proposed

and implemented. Lastly the proposed solutions will be tested and evaluated through experiments to achieve the expected research outcomes. The testing and evaluation process modifies the solution iteratively until satisfactory results are obtained.

To achieve the research objectives (presented in section 1.2), the following four main tasks will be completed.

- Develop a parallel algorithm for the time-consuming cloud-based web service on SaaS (to achieve Obj1):

  For a time-consuming cloud-based web service, the main challenge is program parallelisation of the web service. To obtain auto-scaling properties from cloud systems, the web service program needs to be developed following the rules of cloud platforms, i.e. Map-Reduce. In this work, a recommendation algorithm (i.e. collaborative filtering) is studied as the special case for time-consuming cloud-based web service. This algorithm can recommend products to the targeted customer according to the purchase history of all existing customers. It is a time-consuming process due to the large-scale purchase history data. The parallel web service program can speed up the performance by allocating numerous computational resource from cloud systems.

- Develop a platform of the self-adaptive configuration optimisation system

(to achieve Obj2):

As the cloud web service platform has time-varying visit workloads, this work needs to develop a self-adaptive system to fit the time-changing resource volume. Moreover, the platform needs to support multiple kinds of web services, thus a study of the self-configuration of different services to cloud servers is required. Considering the system's robustness, not all cloud servers can simply be set up to support all services, nor can one server be set up to support only one kind of service to improve performance. The developed configuration optimisation system will deploy multiple kinds of service to allocate servers automatically. To simplify the scenario, the fast-completed services on this platform are studies, since a time-consuming web service can be divided to many fast-completed services.

- Propose an optimisation framework for **computation** resource allocation for cloud-based web service (to achieve Obj3):

  For computation-intensive cloud web services, the framework will allocate numerous computation resources, e.g. VMs to deploy the web services. Considering the time-varying workloads, the system will predict future workload requests by integrating time-series analysis and machine learning techniques. Moreover, the framework uses queueing theory to model the workload requests, and then optimises the resource allocation by con-

sidering the trade-off between cost and QoS.

- Propose an optimisation framework for **network** resource allocation for cloud-based web service (to achieve Obj4):

  For network-intensive cloud web services, such as VoD, the framework will allocate corresponding resources to satisfy the service requests. Emerging SDN techniques will enable the cloud resource to be a new auto-scaling resource. The framework first predicts the required network resource, e.g. bandwidth, in advance. Then the relationship between the cost and QoS is modelled. Finally, an optimisation algorithm is developed for resource allocation while satisfying the QoS of SLAs.

1.5 Thesis Structure

This thesis consists of seven chapters, the relationships of all chapters are shown in Figure 1.1. The introduction of each chapter is described as follows:

Chapter 1: presents an overview of this research. This chapter introduces a cohesive picture of the research conducted in this study. It consists of six main sections: the research questions and problems to be solved in this study, the research objectives to be achieved, a brief description of the significance and methodology of this study, an illustration of the overall structure of this thesis, and the list of publications related to this study.

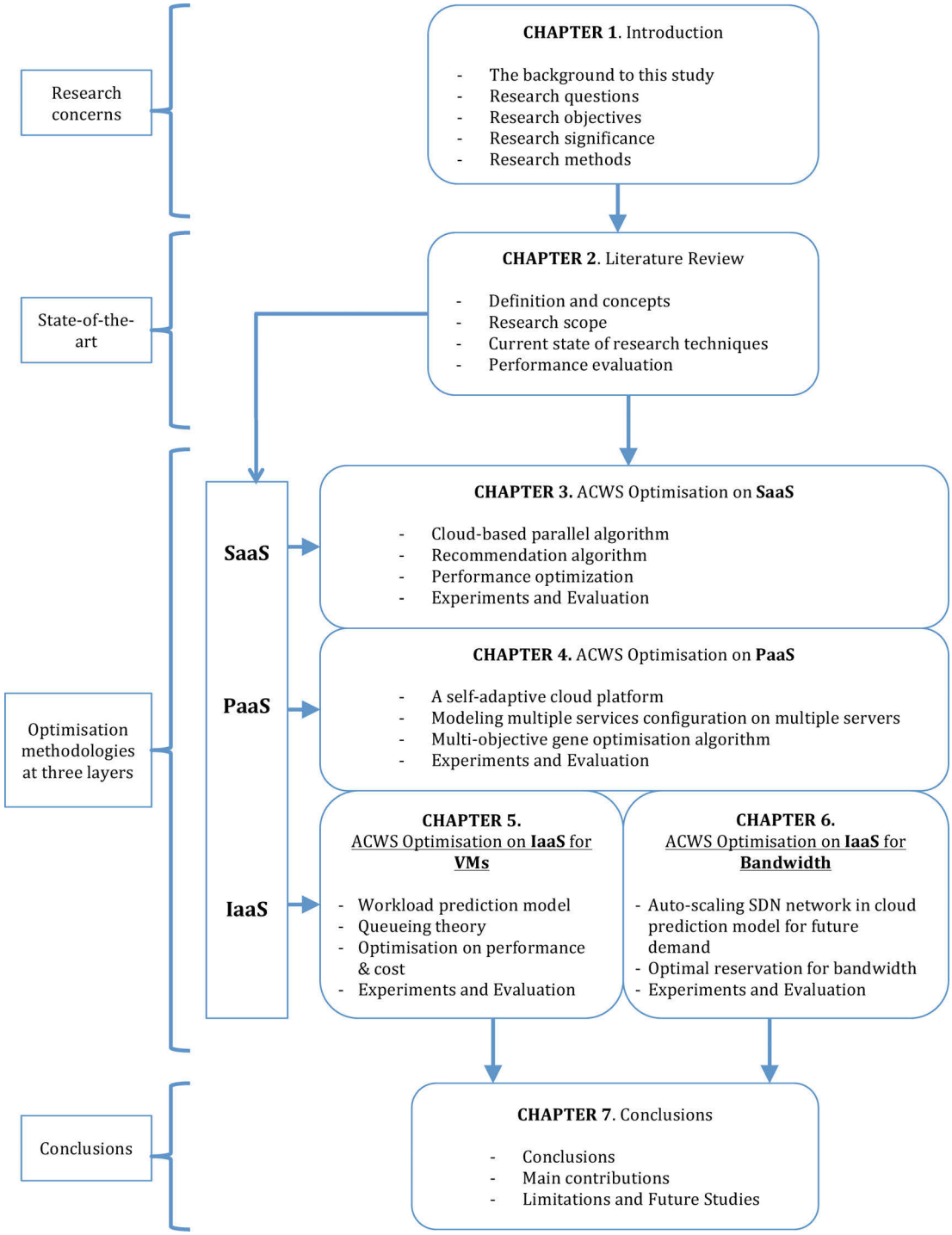


Figure 1.1: Thesis Structure

Chapter 2: expresses a clear review of related research topics. It aims to describe the definitions, technologies, and concepts to understand issues associated with the current state-of-the-art. It provides a summary of the current general state of research in cloud resource consumption in the relevant areas of concern that are most relevant to this research. It also discusses a number of proposed models and techniques for consumption of several types of cloud resource and how to analyse the amount of consumption in the big data era.

Chapter 3: studies the auto-scaling of cloud web services at the SaaS level. Considering a recommendation algorithm as a case study, a parallelised time-consuming algorithm is developed to achieve optimal performance with auto-scaling ability.

Chapter 4: develops a self-adaptive optimisation framework for cloud web service at the PaaS level. This work focuses on the fast-completion web service. First, the server configuration problem is defined and the relationship between the system parameters and the QoS of the SLA is then modelled. Based on the model and configuration problem, a multi-objective optimisation algorithm is developed to achieve optimised auto-scaling at the PaaS level.

Chapter 5: presents an optimised VM auto-scaling for the computation-intensive cloud web service at the IaaS level. This chapter develops workload prediction models and studies the optimisation algorithms to solve the problem of VM auto-scaling.

Chapter 6: designs an optimised resource allocation framework for the network-intensive cloud web service at the IaaS level. The new SDN technology is studied to enable the auto-scaling of network resources for cloud-based web services. The prediction of the required network resource is modelled by integrating time series analysis and machine learning methods. Based on the models, the optimisation algorithm is developed to achieve the optimal auto-scaling of network resources and the framework is evaluated by simulation.

Chapter 7: summarises the entire thesis and highlights possible questions for future research work.

1.6 Publications Related to the Thesis

The following is a list of my publications related to the thesis during my PhD study:

- **Jiang, J.**, Lu, J., Zhang, G., and Long, G., 2013, Optimal Cloud Resource Auto-Scaling for Web Applications, *13th IEEE/ACM Symposium on Cluster, Cloud and Grid Computing (CCGrid2013)*, May 13-16, 2013, Delft, The Netherlands. pp.58-65. (ERA tier A, Google Scholar citation by 14 viewed on 15 Feb 2015)

- **Jiang, J.**, Lu, J., Zhang, G., and Long, G., 2011, Scaling-up Item-based Collaborative Filtering Recommendation Algorithm based on Hadoop, *IEEE*

the 7th World Congress on Services (SERVICE2011), July 4-9, 2011, Washington DC, USA, pp.490-497. (ERA tier B, Google Scholar citation by 30 viewed on 15 Feb 2015)

- **Jiang, J.**, Lu, J., and Zhang, G., 2011, An Innovative Self-adaptive Configuration Optimization System in Cloud Computing, *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC2011)*, Dec. 12-14, 2011, Sydney, Australia. pp.621 - 627. (ERA tier B)

- **Jiang, J.**, Lu, J., and Zhang, G., Data-driven Auto-scaling for Time-varying Virtual Machine Consumption, *Information Systems* (*submitted*).(ERA tier A*)

- **Jiang, J.**, Lu, J., and Zhang, G., Predictive Bandwidth Auto-scaling for Cloud-based VoD Service, *Future Generation Computer Systems* (*submitted*). (ERA tier A)

Chapter 2

Literature Review

The primary goal of this research is to achieve optimised auto-scaling for cloud-based web service from the perspectives of three cloud levels: SaaS, PaaS and IaaS. In this chapter, the concepts, motivations, existing methods and current technologies of ACSW will be reviewed (Section 2.1). The state-of-the-art techniques of auto-scaling optimisation on cloud computing will then be reviewed in Section 2.2.

2.1 Auto-scaling Cloud-based Web Service (ACSW)

2.1.1 Cloud Computing

Cloud computing will be the next generation of computation [20]. The work in [81] defines cloud computing as a model for enabling ubiquitous, convenient,

on-demand network access to a shared pool of configurable computing resources (for example, networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management efforts or service provider interaction. People can possibly have everything they need on the cloud through the internet. Moreover, cloud computing has emerged as an extremely popular and cost-effective computational service model for the on-demand provisioning of virtual resources. Major commercial cloud vendors have obtained great success by providing the flexible "pay-as-you-go" or "on-demand" services [69] with a number of various service instance types and plans to meet customers computing needs, which are supported by a large number of distributed physical resources. For example, Amazon EC2 provides three families (on-demand, reserved, spot), and each family includes several service instance types. Different types of service instances are operated at different quality of service (QoS) levels and have a different price.

In this pay-as-you-go cloud business model, users can purchase and increase or release resources by manual when they need them, rather than owning these devices all the time, so that users greatly reduce their capital expenses without upfront investment. However, most users lack domain knowledge for configuring and monitoring systems. Consequently, this new cloud scenario presents new challenges for them and it is difficult for them to make correct decisions on purchasing appropriate resources before the systems are executed, especially under

the situation of an uncertain and dynamically changing workload (for example, application requests). The problems are summaried as:

(1) System configuration: what types of cloud instances should be used? How big a system (how many CPUs, how many disks, how much network bandwidth) is required to execute the workload within completion time and cost constraints?

(2) Resource schedule: when workload changes (peak and non-peak), should users increase new resource or release existing idle resources? How should they reconfigure the system?

(3) Workload management: what kind of attributes do the workloads have? How to model the workload distributions and how to predict them become problems to be addressed.

To address the above challenges, the cloud providers have to: (1) model workload distributions correctly to anticipate spikes and to account for diurnal patterns; (2) model the correlations among system performance, configuration and workloads, to optimise the configuration by maximising the performance and minimising cost and contentions; (3) design self-adaptive configuration optimisation systems along with time-changing workload distributions.

2.1.2 Cloud Resource Scaling

In literatures, the problems of cloud resource scaling, such as scaling for virtual machine, storage, and other computation resource, have been addressed using different techniques. Considering the two principle aspects of cloud resource, namely cloud resource vendors (resource provisioning) and their tenants (resource consuming), most of the existing work can be classified into two categories: auto-scaling for cloud resource provisioning and auto-scaling for cloud resource consumption. There are some work which have proposed to address the scaling for VM provision, such as the paper [102] proposed a scaling method of VM provision for NoSQL application; a framework was developed in [10] for the implementation of scaling services for VM deployment in IaaS that follows both reactive and proactive approaches; The work in [13] focused on the cost-efficient VM Provisioning for multi-tier web applications and video transcoding. In contrast, the problem of scaling for VM consumption was addressed in [55] and [77] to achieve cost-effective outcomes. Additionally, the authors in [40] analysed an auto-scaling scheme to manage the data centres effectively.

Little research work has been found recently which has been done on cloud bandwidth auto-scaling. In paper [104], the dynamic network bandwidth demands of VMs in data centres was considered, but the authors mainly focused on solving VM consolidation rather than solving the auto-scaling for bandwidth

provision. A predictive auto-scaling method was proposed in [85] to achieve minimal bandwidth consumption for VoD applications from multiple data centres, but the authors analysed the bandwidth demand based on the traditional network architecture and mainly considered the optimal network selection among data centres. The work in this chapter studies the bandwidth auto-scaling based on the new SDN-assisted cloud network architecture and this work mainly focuses on outgoing bandwidth requirements as the data is transferred between the end users and service servers which is deployed in the cloud by bandwidth-intensive application providers.

The development of new network virtualisation technologies, such as SDN-assisted cloud network, enables the auto-scaling for cloud bandwidth reservation to be technically feasible. SDN-assisted cloud networks will facilitate cloud network tenants choosing optimal network consumption from multiple cloud network vendors, while avoiding vendor lock-in, and providing dynamic access for timely internal bandwidth of data processing and large-scale outgoing bandwidth of transferring requirement, and enhance the current bandwidth utilisation. Especially, the effective bandwidth auto-scaling approaches are urgently expected in the era of big data where large-scale, dynamic and quick-response network bandwidth is required for big data transferring.

2.1.3 Auto-Scaling for Cloud Resource

The approach of auto-scaling proposed in the paper [1] allowed cloud users to automatically scale up or down the capacity of required resource according to predefined conditions. This kind of service was supported by essential characteristics of the cloud computing: on-demand self-service, broad network access, resource pooling, rapid elasticity, and Measured service. Particularly, the auto-scaling service could be provided from three levels of cloud computing service models: SaaS, PaaS and IaaS.

Lagar-Cavilla et al. [66] proposed a rapid VM cloning method in cloud computing, namely SnowFlock. It introduced VM fork, a clean abstraction that simplifies development and deployment of cloud applications by dynamically changing the execution footprints. Amazon also proposed a similar product, namely Amazon Machine Images (AMI), to simplify the deployment of launching VMs.

"Scaling Quickly" is one of the obstacles for cloud computing [36]. Utilising the statistical machine learning as a diagnostic and predictive tool would allow dynamic scaling, automatic reaction to performance and correctness problems, and generally automatic management of many aspects of these cloud systems.

Gandhi et al. [38] presented firstly the theoretical optimality on server farm management by considering multiple status, for example, idle, busy, sleep, power off. They also proposed a simple and traffic-oblivious policy-based method for

modelling time-varying demand patterns and provided analytical and empirical evidences to demonstrate that the performance achieved near-optimality.

A profile-based auto-scaling system was developed in [57] to enable the just-in-time scalability for cloud applications. Particularly, profiles capture expert knowledge on scaling applications dynamically. Guided by profiles, profile driver automates the setup and scaling of execution environments. The authors in [26] proposed a scaling approach with a front-end load-balancer for routing and balancing user requests to web applications in a virtualised cloud computing environment. The proposed dynamic scaling algorithm automatically provided resources based on a threshold number of active sessions. The authors in [112] proposed an approach to manage infrastructure resources in PaaS by leveraging two adaptive control loops: the resource consumption optimisation loop and the resource allocation loop.

Like all businesses, both cloud infrastructure vendors and their tenants concentrate mainly on two aspects: income and costs. They need to provide good quality of service to attract more customers and thus increase their income. At the same time, efficient management of their system should be developed to reduce overall costs. The study of VM management should focus on both aspects of VMs. The VM vendors and research work on VM provision need to consider issues, such as convenient VM offering, VM allocation for multiple tenants, configurations of VMs underlying physical machines, price model, efficient energy

consumption, anomaly or failure identification. In contrast, the research work on VM consumption, on which cloud tenants deploy their applications, concerns some topics, such as the efficient selection of VM service from different vendors, the workload of application environments, optimal number of VMs, quality of service.

In general, the auto-scaling management for VM provisioning can be divided into two types: reactive approaches and predictive-based approaches.

Reactive Approach. This mainly refers to the rule-based or policy-based method [84] and control theoretic-based method [86] or reinforcement learning [109]. All involve reacting to the current system by monitoring one or more variables (for example, the current CPU utilisation, or current power, or current response time, or current request rate) to turn VMs on or off to achieve maximal resource utilisation and high quality of service. The reactive approach is easy to execute, but when the boot-up time of VMs is high (for example, 120s) and workloads change significantly and rapidly, it can be inadequate for meeting high quality of service because the availability of increased VMs only takes place 120 seconds later.

The most popular cloud infrastructure vendors have adopted rules-based auto-scaling mechanisms to manage their VM provisioning, such as Amazon AWS AutoScaling [1] and RightScale [3]. These rules are typically divided into two classes: one for scaling up and one for scaling down. Each rule involves one

or more parameters manually defined by cloud tenants (i.e. application service providers), such as upper and lower threshold for CPU utilisation, and the period of running VMs that is the time point of the action for scaling up and scaling down. These rule-based mechanisms require cloud tenants to manually specify scaling rules, and it is hard to achieve full automation when the workload changes in time. Consequently, it is difficult for cloud tenants to achieve cost-effective and high quality of service with a time-varying workload, using these rule-based auto-scaling mechanisms.

Several research works [73][90][61] have extensively studied applying control theory to achieve adaptive fine-grained cloud resource provisioning based on reactive feedback mechanisms. However, such control-based approaches often have parameters that need to be specified or tuned offline, and they are suitable for slowly varying workload environments, but have difficulty dealing with sudden burst and time-varying workload circumstances.

Predictive-based Approach. This aims to predict in advance what the workload will be in future unit time, and to start up VMs straight away if necessary. Predictive approaches work well when the workload changes as a result of periodic or seasonal characteristics. In [44], Gong et al. proposed a lightweight online demand prediction approach for cloud resource provisioning that can handle both cyclic and non-cyclic workloads. The work in [53] proposed a workload prediction approach by using linear regression and histogram-based methods. [89] [45]

proposed a dynamic cloud resource provision approach by multiplying estimated resource usage based on different QoS levels. In [59], a web cache model was introduced to adjust IaaS resources to assist application providers to make decisions. A lightweight scaling algorithm was proposed by [50] to enable fine-grained scaling application at the level of underlying resources. The authors in [56] and [33] aimed to find a satisfactory balance between reducing energy consumption and operating cost while maintaining an acceptable service level for minimising the number of idle VMs. Gandhi et al. [40] proposed a dynamic capacity management policy for a Multi-Tier Data Centre to improve resource utilisation while meeting response time SLA by analysing the time-varying load.

The auto-scaling problem for the VM consumption of cloud tenants can be divided into two steps: accounting for the number of VMs required and making decisions. The first step involves estimating the future workload or reactively computing the VMs required. The second step consists of deciding on scaling-up or scaling-down by considering a set of pre-defined rules or by solving an optimisation problem based on the value obtained in the first step. Incorrect decisions result in insufficient consumption and over-consumption of VMs. Insufficient VMs will unavoidably harm performance and cause SLA violations, while VM over-consumption will create cost waste and resource idleness. Therefore, the final objective of an auto-scaling mechanism is to automatically adjust the required VM consumption to minimise cost while satisfying the SLA when the

workload changes over time.

A cloud tenant (a web application provider) serves the end users' requests by deploying its application on rented VMs from an infrastructure vendor. There are various types of VM instances with different configurations and price models, whereby cloud tenants have considerable flexibility to set up their system. As the parameters of the VM deployed on physical resources, such as CPU, memory, I/O or network bandwidth, are not necessarily dependent [109] and not managed by cloud tenants, it is not trivial for cloud tenants to model the VM demand directly by using these parameters that is resource-level. It makes sense for cloud tenants to analyse VM consumption at VM-level rather than at resource-level that is considering the number of VMs, rather than the number of configuration parameters as the resource demand quantity.

Little research has been conducted on cloud tenants' resource consumption, because cloud services have only been in existence for a few years and little resource consumption data is public. [23] analysed a set of factors on cloud service providers (i.e. cloud tenants) for maximising the profits, but VM auto-scaling is not considered in that paper. Other work in [55] studied optimal VM auto-scaling for web applications deployed on rented VMs, but it is not suitable for managing fast-varying, time-varying and large-scale workloads. [77] proposed an auto-scaling method to minimise cost and meet application deadlines in the cloud workflow. In the absence of the deep understanding of the behaviour of

resource consumption for cloud tenants, one possible way is to leverage the data-driven method by online or offline analysis of the collected data.

The management of allocating cloud resource adaptively to on-demand requirements of an application, called auto-scaling, can be very challenging. Resource under-provisioning will unavoidably harm performance and cause SLAs violations, while resource over-provisioning will result in cost waste and resource idleness. Therefore, the final objective of an auto-scaling mechanism is to automatically adjust acquired resources to minimise cost while satisfying the SLAs. This section surveyed related state-of-the-art work in the field of cloud resource auto-scaling to achieve optimal utilisation and dynamic scalability.

The revolution of cloud computing means going to think about keeping the business on the basis of services that provide necessary functionality and automation is one of the criteria among them. The automation of the data centre is of particular importance, not only because it leads to higher productivity and efficiency, but also because the transition to cloud computing will be easier to make. Data Centre automation is concerned with the processes and technologies for automated deployment and management of IT services. Cloud computing has put centrally control on all aspects of the data centre (for example, computing, networking, storage) through hardware-independent management and virtualisation software. Now, automation can help data centre providers as agile as possible in the cloud world. Automation solutions allow cloud users to take advantage of

specialized facilities and equipment without having to create their own expensive infrastructure and construction of specialized facilities. Data centre automation enables an IT organization to automate the management of virtual and physical elements of its infrastructure, including services, network devices, storage devices, and to automate the deployment of their applications. Vendors like HP, Microsoft, and BMC are working to provide massive systems under one management engine for better management of environment and workload.

2.2 Optimised Auto-scaling on Cloud Computing

The energy consumed by the servers in data centres is very large. The report in [19] states that the data centres in the USA consumed about 61 billion kilowatt-hours (kWh) in 2006 (1.5 percent of total U.S. electricity consumption) with a total electricity cost of $4.5 billion. At the same time, energy consumed by the servers and data centres increases annually and occupies a significant part of global energy consumption with the rapid development of cloud-based applications. Therefore, optimising the resource usage for cloud-based web services can not only reduce energy consumption but also benefit the environment and the economy. This section will consider the resource allocation for auto-scaling on cloud computing as an optimisation problem among conflicted factors, for exam-

ple, energy consumption, system robustness, running cost and QoS. Particularly, this work studies the optimisation problem from three-levels as shown in Figure 7.1.

2.2.1 SaaS Level Optimised Auto-scaling for Cloud Computing

On SaaS level, this work will consider how to parallelize the application (for example, programs) in the cloud computing environment, such as Hadoop, Amazon EC2 and Windows Azure. Mahout is a project of the Apache Software Foundation to produce free implementations of distributed scalable machine learning algorithms focused primarily in the areas of collaborative filtering, clustering and classification which is implemented on the Hadoop platform [106],[16].

Some research focused on designing a new cloud-based framework for complex business processing and workflow. The work in [70] proposed a model-based methodology to size and plan enterprise applications while satisfying SLAs. Particularly, the proposed approach was illustrated in a real-world ERP application for SAP ERP. A prediction-based auto-scaling methodology for data-centric workflow tasks was proposed in [29]. The authors in [103] treated applications as a whole single entity and dealt with the relations among different application components.

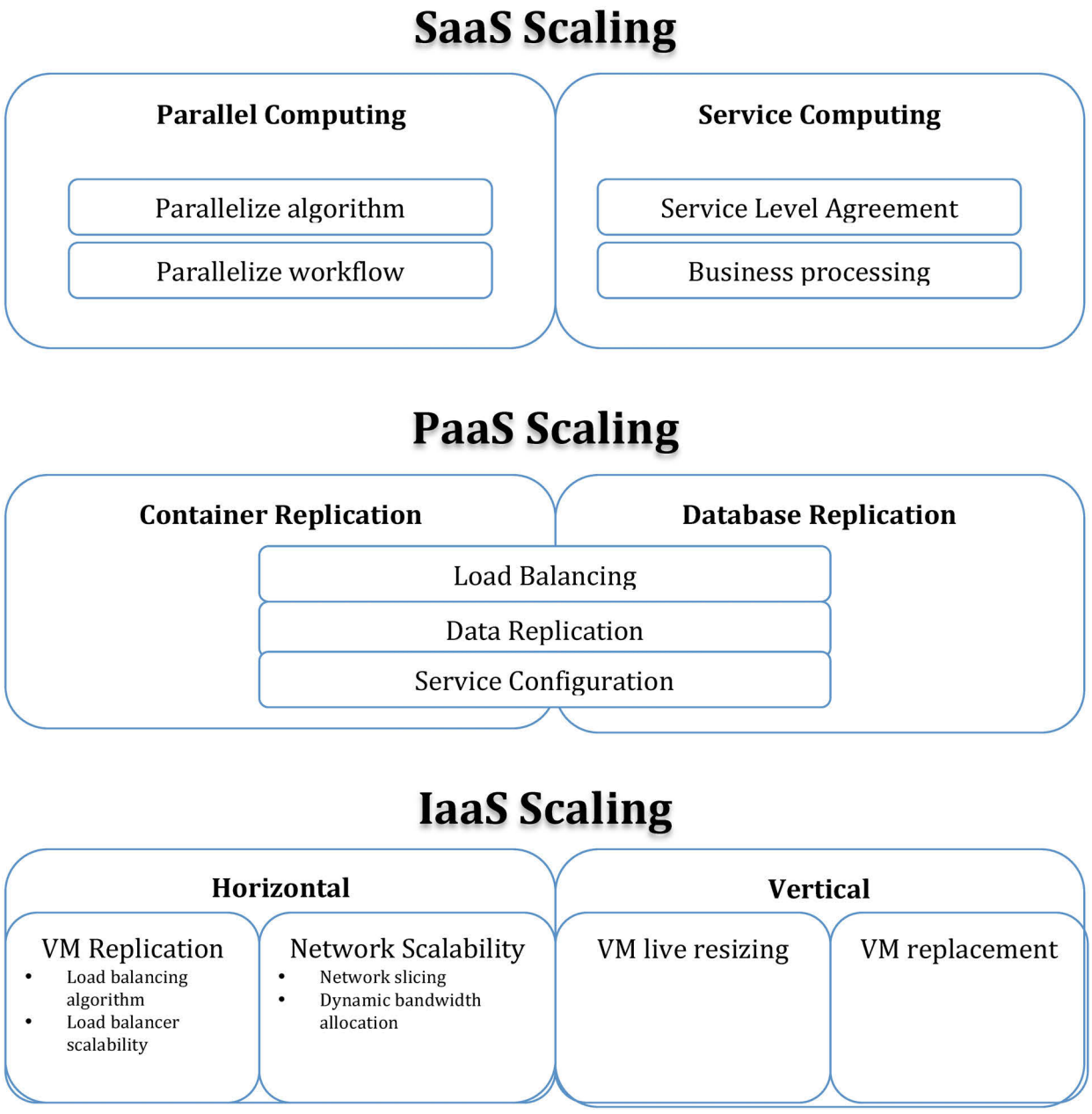


Figure 2.1: Summary of Cloud Scaling Techniques

When designing the distributed applications and systems especially web service, the quality of service (QoS) is a key factors to be considered. In service

computing, SLAs are used to define the agreed QoS indicators between the service providers and service users. As show in the sections to follow, SLAs are widely using as QoS evaluation indicators in cloud-based web service.

2.2.2 PaaS Level Optimised Auto-scaling for Cloud Computing

In the PaaS model, cloud providers deliver a computing platform, typically including operating system, programming language execution environment, database, and web server. Under this topic, this section will study how to optimise the platform from the three facts: load balancing, data replication and service configuration.

(1) Load Balancing

In computing, load balancing distributes workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units or disk drives. In cloud computing, the general idea is use the existing load balancing method to distribute the workload across multiple cloud resources, such as VMs, networks and storages. Load balancing methods are well studied. Due to space limitation, this section doesn't list all those works.

Another idea is to use the load balancing information to optimise the resource allocation in a cloud-based system. In [52], the authors proposed a methodology

and presented a system for automatic detection and resolution of bottlenecks in a multi-tier cloud-based Web application host to satisfy specific maximum response time requirements. The system included a novel method for identifying and retracting over-provisioned resources in multi-tier cloud-hosted Web applications. The method used real-time information of load balancing to identify the bottlenecks. The work in [71] presented a method to optimise cloud resource allocation by using a performance model which was composed of load balancing information. A time-varying performance model was developed which dynamically updated each application's performance model at runtime to adapt to the changes of load balancing in the system [43].

(2) Data Replication

For data-intensive application, especially database platform, scaling data replication can provide time-varying large-scale users with fast access to current data. The authors in [93] solved the query optimisation problem for the database platform on cloud computing environment by dynamically allocating an optimised VM resource. Particularly, the system recorded and modelled the relationship between the volume of allocated resource and the correspond performance of special kinds of query, and then made the optimal solution for each query correspondingly.

(3) Service Configuration

The low average utilisation of servers is a well known cost concern in data centre

management. The paper [17] studied the dynamic placement algorithm of VMs for managing SLA violation. Compared to the static placement solution, the dynamic algorithm was developed in [62] to reduce the cost of VMs while keeping the rate of SLA violations. Moreover, dynamic placement for web applications was also proposed to minimise the configuration changes while satisfying the time-varying user demands for the configuration of services. The work in [25] developed a self-adaptive policy which helps application providers decide on when and how to reallocate resources by considering the load of VMs. Particularly, it used Nash equilibrium to build up the global performance-to-price model.

In the rule-based approach, various alerts (for example, events) were defined heuristically on instance metrics (for example, CPU utilisation), which were then aggregated at a global scale in order to optimise the provisioning decisions for a given application tier. Different criteria were enumerated such as design complexity, ease of comprehension, and maintenance to compare different methods, and concluded how these approaches can optimise the resource allocation [41]. The work in [54] involved fuzzy logic to improve the elasticity rules made for cloud-based software.

Large complex service centres must provide various services to a number of users with separate service contracts, while managing their overall costs. A fast scalable hybrid optimisation procedure wass proposed for service deployment by combining fast linear programming and nonlinear performance model [72]. The

work in [78] and [77] proposed auto-scaling methods to minimise the resource cost for work-flow task while satisfying the predefined deadline. An allocation and provisioning model of science cloud was discussed in [63], especially for high throughput computing application.

2.2.3 IaaS Level Optimised Auto-scaling for Cloud Computing

In cloud computing, there are three kinds of infrastructural resources, computation, network and storage. Most auto-scaling related research focued on computation and network resource scaling, and few of them studied the storage scaling. [76] conducted a survey for resource management at IaaS level in cloud computing. Some papers focussed on some of the important resource management techniques such as resource provisioning, resource allocation, resource mapping and resource adaptation. [21] developed a toolkit, namely CloudSim, for modelling and simulation of Scalable Cloud Computing Environments. [22] presented vision, challenges, and architectural elements for energy-efficient management of Cloud computing environments. In particular, the paper proposed (a) architectural principles for energy-efficient management of Clouds; (b) energy-efficient resource allocation policies and scheduling algorithms considering quality-of-service expectations, and devices power usage characteristics; and (c) a novel software

technology for energy-efficient management of Clouds.

(1)Vertical Scaling

Vertical scaling aims to enhance the provision capability by providing more powerful servers. The imaginary scenario is that of on-the-fly changing of the assigned resources to an already running instance; for example, adding more physical CPU to a running VM. Unfortunately, the most common operating systems do not support on-the-fly (without rebooting) changes on the available CPU or memory to support this "vertical scaling" [103]. More of current research focussed on VM replacement method which increased provision capability by using more powerful servers to replace less powerful ones [88]. [67] proposed vertically scaling VMs by employing dynamic server consolidation by periodic VM migration. Particularly, the method periodically optimised the mapping and configuration between VMs and the physical machine by considering the workload and the capacity for physical machines. The vertical elasticity was much more limited as it could not scale outside single physical machines [88].

(2)Horizontal Scaling

Horizontal scaling is defined to improve the provision capability by increasing the number of servers. Similarly, VM Replication is used to increase provision capacity of the system by adjusting the volume of VM replications. For this kind of approach, a few researches used control theory to solve the VM horizontal scaling problems. The research [60] [61] integrated the Kalman filter into

feedback controllers to dynamically allocate CPU resources to VMs. The VM resource provision problem was modelled as a sequential optimisation under uncertainty problem in [65], and was solved by utilising a predictive control approach of limited lookahead control (LLC) strategies [5]. The paper [42] introduced a novel auto-scaling approach in which both cloud and application dynamics were modelled in the context of a stochastic, model predictive control problem. The approach exploits trade-off between satisfying performance related objectives for the consumer's application while minimising their cost. To support different SLA level services on cloud platform, the authors in [74] presented an adaptive multivariate controller that dynamically adjusted the resource configuration on individual tiers. The controller parameters were automatically tuned at runtime based on a quadratic cost function and a system model that was learned online using a recursive least-squares (RLS) method.

Some research work used a predictive based optimisation method to solve the VM allocation problem. According to the predicted load within a given time slot, a series of heuristic optimisation policies were applied to solve this VM allocation problem in [80] and [79]. All policies were easily implementable, and the selected policy was practicable for different visiting pattern and load. In [55], the scaling problem was modelled as a constraint optimisation problem for cost and performance. Particularly, the visiting pattern of each time slot was modelled with Queueing theory, and the relationship between cost and performance was

inferred from the predicted workload and SLA. This approach built a theory foundation for Queueing theory based predictive resource provision in VM scaling.

Other research works used a reactive method to optimally adjust the number of VMs. [15] dynamically allocated the workload to VMs with modification Best Fit Decreasing (BFD) algorithm. It allowed leveraging of the heterogeneity of the nodes by choosing the most power efficient ones. To use this method, the system needed to know the workload of each task and the capacity of each cloud node before allocating. [97] considered a service system model primarily motivated by the problem of efficient assignment of virtual machines to physical machines, so that the number of occupied hosts was minimised. Particularly, it modelled the resource assignment task as an optimisation problem with general packing constraints, and solved it with a simple parsimonious real-time algorithm, called Greedy.[33] presented a model-driven engineering approach to optimising the configuration, energy consumption, and operating cost of cloud auto-scaling infrastructure to create greener computing environments that reduce emissions resulting from superfluous idle resources. To dynamically adjust server farm capacity without requiring any prediction of the future load, or any feedback control, [39] designed and implemented a class of Distributed and Robust Auto-Scaling policies (DRAS policies), for power management in compute intensive server farms. [105] addressed the autonomic provisioning problem for a virtualised outsourcing cloud data centre by utilising a non-linear constrained

optimisation model.

Network resource is another resource for horizontal scaling. Network scaling is a key factor in providing high-quality online VoD service. YouTube, the most popular online VoD service, experiences over 100 million video views per day [4]. To deliver all that video to all those users, YouTube video delivery system consists of three major components: a "flat" video identify space, multiple DNS name spaces and a 3-tier physical cache hierarchy [8]. The multiple DNS name spaces is presented as a multi-layer logical organization for video servers. To quickly scale servers to support growing user views, the content delivery network (CDN) is designed as the server scaling strategies in Youtube. The RTT (Round-Trip Time) between users and data centers plays a key role in the video server scaling process [101]. Moreover, a variety of other factors influence the scaling including load-balancing, diurnal effects, variations across DNS servers within a network, limited availability of rarely accessed video and the requirement of alleviating hot-spots which arise due to popular video contents.

Netflix is the leading provider of VoD (Video-on-Demand) in the US and Canada. In fact, Netflix is the single largest source of Internet traffic in the US, consuming 29.7% of peak downstream traffic [31]. Netflix employs data centers and CDN (Content Delivery Networks) for content distribution [28]. Particular, Netflix makes use of multiple CDNs under changing bandwidth conditions. Based on the architecture of Netflix, [7] proposed a measurement-based adaptive CDN

selection strategy and a multiple-CDN-based video delivery strategy to increase QoS, for example, user's average bandwidth.

Hulu, with more than 27 millions unique viewers per month, is one of the largest online video service provider. Based on an active measurement study of Hulu, [6] discovered that Hulu frequently switch users' preferred CDNs. However, once a CDN is selected, a user will stay with the same CDN for the entire playing time of the movie even when this CDN's performance degrades. While the preferred CDN selection is not fixed, this work observed that Hulu attempts to divide video requests among CDNs to attain a fixed target ratio. In terms of CDNs, this work will consider the different CDNs employed in different amounts of resources (servers) to serve Hulu content.

For these kind of bandwidth-intensive applications, for example, online VoD, [24] proposed a preliminary framework to support content delivery network interconnection (CDNI) assuming two ISPs have deployed OpenFlow. For data centre network virtualisation, [48] proposed virtual data center (VDC) as the unit of resource allocation for multiple tenants in the cloud with bandwidth guarantees.

To assist a cloud based VoD service video provider, [32] reserved optimal network resource from the cloud data centre according to predicted user demands. [85] proposed a predictive resource auto-scaling system that dynamically books the minimum bandwidth resources from multiple data centres for the VoD provider to match its short-term demand projections.

(3) Hybrid Scaling

Some research focused on optimising the resource provision by considering both VM and network resources. The work in [95] proposed a dynamic service placement algorithm to get the maximum benefit from a distributed cloud system. Particularly, to minimise the latency and bandwidth cost for various kinds of services, the algorithm considered optimising the allocation for both computation and storage resource. The joint application-VM-Physical-Machine assignment problem in a cloud environment is discussed in [49] , where application requests are serviced by VMs residing on Physical Machines. Particularly, the paper solved this joint assignment problem by a shadow routing based approach. In the geographically distributed clouds, assuming that users specify their resource needs, such as the number of virtual machines needed for a large computational task, [9] developed an efficient 2-approximation algorithm for the optimal resource allocation of data centres and servers by considering the network distance or latency.

Some proposed methods solve the scaling problem by integrating horizontal and vertical scaling. The work in [92] optimised the packing VM allocation by considering both the number of VMs (Horizontal Elasticity) and the capacity of VMs (Vertical Elasticity). Particular, it analysed the price/performance effects achieved by different strategies for selecting VM-sizes for handling increasing load and a cost-benefit based approach was developed to determine when to (partly) replace a current set of VMs with a different set.

Chapter 3

ACWS Optimisation on SaaS

Cloud-based web services in the SaaS business model are provided to users as applications. Cloud-based web services can be categorised into three kinds: (a) rapidly completed; (b) time-consuming and parallelisable; and (c) time-consuming and non-parallelisable. Little research can be conducted at the SaaS level to efficiently scale the rapidly completed and time-consuming and non-parallelisable categories of web services. An auto-scaling platform can be developed at the PaaS level to achieve high performance of provision for rapidly completed web services, and the scaling of time-consuming and non-parallelisable services can be easily achieved by increasing powerful machines at the IaaS level. If a web service is time-consuming and parallelisable, the parallelisation optimisation is the key factor to be studied to improve the performance at the SaaS level. Therefore, the work in this chapter focuses on studying this type of time-consuming and

parallelisable cloud-based web service.

In this chapter, a recommendation algorithm, namely item-based collaborative filtering (ICF), is studied as the case study of cloud-based web services. The ICF algorithm is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). In a single thread scenario, the computation complexity of the ICF algorithm is $O(MN^2)$ (M is the number of users, and N is the number of items). Since a recommender system typically involves large-scale data sets, i.e big M and big N, the execution time of ICF will be very long which is unacceptable for auto-scaling cloud-based web services to satisfy the QoS of SLA. To solve this problem, it is necessary for the ICF to be parallelised and allocated in a cloud environment so that the execution time can be reduced by distributing the ICF algorithm into increased computation resources. Hadoop is applied as the cloud computation platform in this research. Hadoop is a popular open source cloud computation platform in which the computation jobs are separated into multiple Map tasks and Reduce tasks. It is implemented to process large-scale datasets, where hundreds of terabytes of data are allocated on thousands of common machine clusters [68].

3.1 Methodology

In this section, a scalable item-based collaborative filtering (ICF) algorithm is developed as a series of Map tasks and Reduce tasks deployed in a Hapdoop cluster. The costly computation of ICF is split into a number of small computing partitions, each of which is independently executed on different nodes in parallel. Efficient partition strategies are also proposed to maximise data locality on increased computation resources to reduce the communication cost and to control the algorithm complexity. This enables the ICF to auto-scale efficiently even on large-scale datasets.

(1) Overview of MapReduce

MapReduce [30] [30] is a popular programming model developed by Google. MapReduce is ideal for data already stored on a distributed file system which offers data replication as well as the ability to perform computation locally on each data node [35] [27] [114].

As its name suggests, there are two functional programming phases in the MapReduce framework: the Map phase and the Reduce phase. The input of the computation is a set of (key, value) pairs and the output of the computation is also a set of (key, value) pairs. It defines these (key, value) pairs using angle brackets $< k, v >$ on both phases. The key is used primarily in the reduce phase, to determine which values are combined. The values describe arbitrary information.

These are specified using two functions:

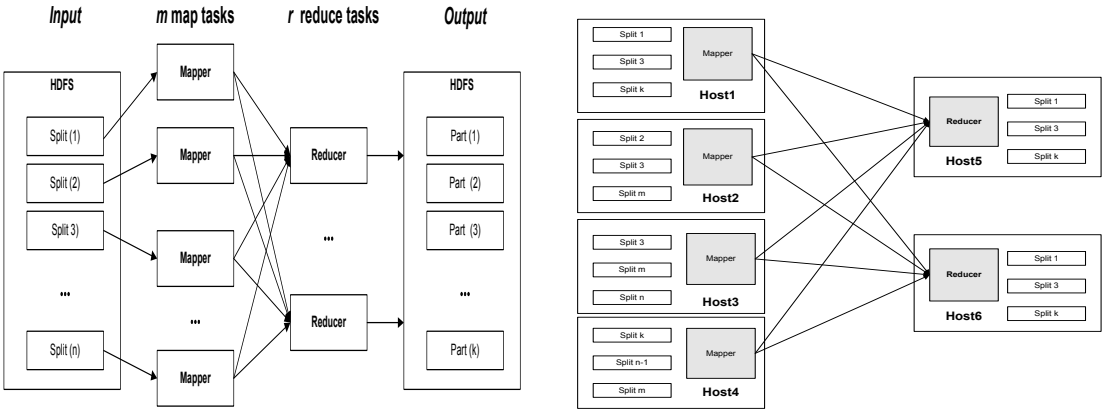
$$Mapper : (k1, v1) \rightarrow list(k2, v2)$$

$$Reducer : (k2, list(v2)) \rightarrow list(k3, v3)$$

Figure 3.1 shows the MapReduce computation framework. The computation starts from a map phase in which the map functions are executed in parallel with various partitions of input data which are stored in a Distributed File System (DFS). Processing each partition is assigned to one map task. The output pairs of each map function are hash-partitioned on the inter-mediate key. Each partition is sorted and merged in sort order by their keys. All the partitions which share the same key are sent to a single reduce task in which the reducer function obtains the final results. The output of each reduce function is written across the cluster in DFS.

Besides the map and reduce phase, developers are also allowed to program a combined phase which is processed between the map phase and the reduce phase but executed on the same node as the mapper function. The aim of the combined function is to reduce the communication load through the network between clusters by operating the local pairs.

In the open-source community, Hadoop uses the same MapReduce architec-



(a) Data flow (b) MapReduce Placement

Figure 3.1: MapReduce Computation Framework

ture as the computation model but is implemented in Java, which is used in this research. Programs written in this function style are automatically parallelised and implemented on a cluster of machines. For MapReduce jobs in Hadoop, a jobtracker acting as a MasterNode splits job data into several pieces as the input of the map phase, and the tasktrackers acting as a DataNode store the intermediate results of the map functions in a local distributed file system called Hadoop Distributed File System (HDFS). Hadoop schedules the MapReduce computation based on locality and improves the efficiency of overall I/O throughput in the cluster to reduce the computation cost.

(2) Item-based Collaborative Filtering (ICF) Algorithm

The Collaborative Filtering (CF) algorithm has two presentation forms: User-based CF and Item-based CF. In this research, item-based CF (ICF) will be

studied.

Unlike the user-based CF algorithm, the ICF method checks a set of items rated by the active user, computes the similarity between them and the target item i, then selects the k nearest neighbourhood items. A set of corresponding similarity is denoted as $S = \{s_{i1}, s_{i2}, \ldots, s_{ik}\}$. Once the most similar items have been selected, the prediction is computed. (One fundamental difference between the similarity computation in user-based CF and ICF is that in user-based CF, the similarity is computed along the row of the matrix but in case of the ICF, the similarity is computed along the columns (i.e. each pair in the co-rated set corresponds to a different user).

The purpose of the ICF algorithm is to recommend new items or predict the probability of a certain kind of item for a target user, based on the user's previous ranking and the likeness of other users who have similar preferences to the target user. A typical ICF operates a set of m users $U = \{u_1, u_2, \ldots, u_m\}$, and a set of n items $I = \{i_1, i_2, \ldots, i_n\}$. Each user $u_i \in U$ gives a rating value set $r_{ij} \in R$ for a corresponding subset of items $I_j \in I$. In this research, the rating value domain R is the set of integers, such as from 1 to 5. Note that it is possible for r_{ij} to be a null-set. The ICF algorithm can be divided into two steps as follows:

- Similarity computation. Seeks to find a neighbourhood for target u_a or I_a based on the similarity of their rating values. There are several methods

for computing similarity, such as Pearson correlation, and cosine distance.

- Prediction and Recommendation. Once the most similar users or items have been found, selecting new items or users with high predicate value and recommending them.

One critical step in the ICF algorithm is to compute the similarity between items and then to select the most similar items. The basic idea in the similarity computation between item i and item j is to segregate users who have rated both of these items and then to apply a similarity computation method to determine the similarity s_{ij}. In this research, Pearson correlation technique is applied to measure similarity between two items i and j. Let the set of users who both rated items i and j be denoted by U, then the correlation similarity is given by

$$S_{i,j} = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_i)(R_{u,j} - \bar{R}_j)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_i)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_j)^2}} \tag{3.1}$$

where $R_{u,i}$ denotes the rating of user u on item i, $\bar{R}_i$ is the average rating of the i -th item.

Once the set of most similar items based on the similarity measures have been generated, the next step is to compute the prediction on an item i for a user j by computing the sum of the ratings given by the user on the items similar to i. Each rating is weighted by the corresponding similarity $S_{i,j}$ between items i and j. Here, a weighted average method is applied to measure the prediction $P_{u,i}$ as

demonstrated in Equation (3.2) [91].

$$P_{u,i} = \bar{R}_i + \frac{\sum_{u \in U}(R_{u,j} - \bar{R}_u)S_{i,j}}{\sum_{u \in U} S_{i,j}} \tag{3.2}$$

3.1.1 Scaling-up ICF Algorithm on MapReduce

This section studies how the ICF algorithm is implemented on MapReduce and what kinds of partition strategies are used to maximise data computation with locality and parallelism.

As for Equations (3.1) and (3.2) shown above, the computation of the ICF algorithm requires intensive computation power that increases with both the number of users and the number of items. In this work, the three most intensive computations in the ICF algorithm are partitioned into four Map-Reduce phases. The three elements of intensive computation are: (1) computing the average rating for each item; (2) computing the similarity between item pairs; and (3) computing predicted items for the target user. The whole computation flowchart on MapReduce is executed in four phases, as Figure 3.2 shows. $Map-I$ and $Reduce-I$ computes the average rating (in Section 3.1.2); $Map-II$ and $Reduce-II$ computes the similarity (in Section 3.1.3); $Map-III$ and $Reduce-III$ records the similarity matrix in preparation for the computation of prediction items in $Map-IV$ and $Reduce-IV$ (in Section 3.1.4).

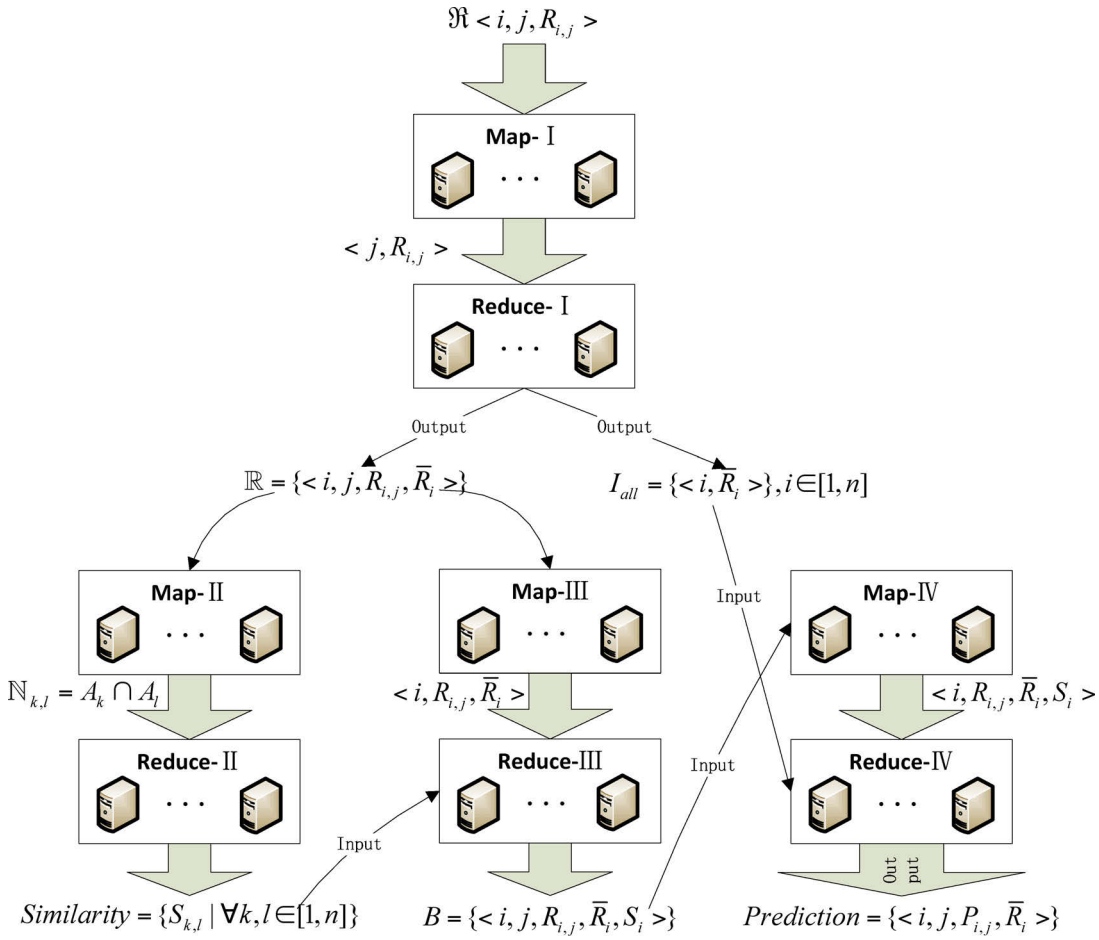


Figure 3.2: Workflow of Item-based Collaborative Filtering Algorithm

3.1.2 Computing Average Rating for Items

In computing the average rating for items, $R_{i,j} \in R$ is defined as the rating value on item i by user j. Being a sparse matrix, the matrix R is represented as a tuple $\Re < i, j, R_{i,j} >$, which records the rating values and the corresponding item i and user j as non-null. Then the average rating of each item is calculated by

the following

$$\bar{R_i} = \frac{\sum_{j=1}^{m} R_{i,j}}{\sum_{k=1}^{l} 1}, \forall i \in [1, n] \tag{3.3}$$

where l indicates the number of items rated by the given user.

A tuple $I_{all} < i, \bar{R_i} >, i \in [1, n]$ is denoted to record all items and their average ratings.

The computation of average ratings of each item can be implemented by the following MapReduce operations.

$Map-I$: Map $< i, j, R_{i,j} >$ on i such that tuples with the same i are shuffled to the same machine in the form of $< j, R_{i,j} >$. If there are n items and m users, and there are k Mappers, the complexity of algorithm in each mapper is $o(\frac{nm}{k})$.

$Reduce-I$: Take $< j, R_{i,j} >$ and output $< i, j, R_{i,j}, \bar{R_i} >$ for each $< i, j, R_{i,j} > \in$ $\Re$. At the same time, the reducer records all the items in a single file $I_{all} = \{<$ $i, \bar{R_i} >\}, \forall i \in [1, n]$. The two outputs of tuples are all ordered by i . If there are k Reducers, the complexity of the algorithm in each reducer is $o(\frac{nm}{k})$.

The output from $Reduce - I$ will replace the tuple $\Re < i, j, R_{i,j} >$ by the tuple $\mathbb{R} < i, j, R_{i,j}, \bar{R_i} >$. In fact, the storage of average rating in each tuple $\mathbb{R} < i, j, R_{i,j}, \bar{R_i} >$ is not redundant, because there are many calculations on $R_{i,j} - \bar{R_i}$ in the steps that follow.

This phase computes the average rating for each item and outputs. It will

output this information in two formats. The average rating is added to the existing rating record $< i, j, R_{i,j}, \bar{R}_i >$ which is used to calculate the similarity (in Section 3.1.3). An overall item list is also recorded with the average rating $i, \bar{R}_i$ which is used to calculate the prediction rating (in Section 3.1.4).

3.1.3 Computing Similarity

In this step, the similarity between two arbitrary items and the formation of the similarity matrix can be calculated. The $S_{i,j}$ is denoted as the similarity between item i and item j and the similarity matrix as $S = \{S_{i,j} | \forall i, j \in [1, n]\}$.

$A_i = \{< i, j, R_{i,j}, \bar{R}_i > | \forall j \in [1, n], < i, j, R_{i,j}, \bar{R}_i > \in \mathbb{R}\}$ is defined as a subset of users who rated the item i. The matrix $\mathbb{N}_{k,l} = A_k \bigcap A_l, \forall k, l \in [1, n]$ is redefined, where $\mathbb{N}_{k,l}$ means the set of users who rated the item k and item l. To obtain good scalability, all the data required in the calculation of $\mathbb{N}_{k,l}$ should be stored in the same node. Each computation of $\mathbb{N}_{k,l}$ could be completed in the same node, while each calculation of A_i should be distributed to different nodes in the MapReduce cluster. Equation (3.1) demonstrates that the computation of $S_{i,j}$ can acquire all the required data from $\mathbb{N}_{i,j}$, and can run locally.

$Map-II$: Map $\{A_i | i \in [1, m]\}$ to $\{\mathbb{N}_{k,l} | \forall k, l \in [1, n]\}$. If there are k Mappers, the complexity of the algorithm in each mapper is $o(\frac{n^2 m}{k})$.

$Reduce-II$: Take $\{\mathbb{N}_{k,l} | \forall k, l \in [1, n]\}$, and output $\{S - k, l | \forall k, l \in [1, n]\}$. If

there are k Reducers, the complexity of the algorithm in each Reducer is $o(\frac{n^2m}{k})$.

This phase computes the similarity between item pairs for users. This tuple will be used to calculate the prediction rating (in Section 3.1.4).

It can be demonstrated that n (the number of items) has more influence in the computation workload than m and k. The item-based algorithm is more suitable for the situation where $m \gg n$ which means it can solve the explosion problem of the number of users. Because the tuple $\mathbb{R} < i, j, R_{i,j}, \bar{R}_i >$ is ordered by the identity numbers (ID) of the items in the $Reduce - I$, the tuples that have the same item ID can easily be arranged in the same node.

3.1.4 Computing Prediction Matrix

The matrix $\mathbb{R} < i, j, R_{i,j}, \bar{R}_i >$ is transferred to the matrix U, which is sorted by user ID, as follows:

$$U = \left| \begin{array}{l} \left[\begin{array}{c} < 1, 1, R_{1,1}, \overline{R}_1 > \\ \cdots \\ < 1, i, R_{i,1}, \overline{R}_i > \\ \cdots \end{array} \right] \\ \vdots \\ \left[\begin{array}{c} < 1, 1, R_{1,j}, \overline{R}_1 > \\ \cdots \\ < 1, i, R_{i,j}, \overline{R}_i > \\ \cdots \end{array} \right] \\ \vdots \end{array} \right| = \left| \begin{array}{c} U_1 \\ U_2 \\ \vdots \\ U_j \\ \vdots \\ U_m \end{array} \right| = \{U_j | \forall j \in [1, m]\} \tag{3.4}$$

where $U_j = \{< i, j, R_{i,j}, \bar{R}_i > | \forall i \in [1, n], < i, j, R_{i,j}, \bar{R}_i > \in \mathbb{R}\}$ and U_j means the set of rating tuples by user j .

The similarity matrix is defined as

$$S = \begin{vmatrix} S_{11} & S_{12} & \dots & S_{1n} \\ \dots\dots & & & \\ S_{i1} & S_{i2} & \dots & S_{in} \\ \dots\dots & & & \\ S_{n1} & S_{n2} & \dots & S_{nn} \end{vmatrix} = \begin{vmatrix} S_1 \\ \dots \\ S_i \\ \dots \\ S_n \end{vmatrix}$$

and $S_i = \{S_{i,j} | \forall j \in [1, n]\}$; S_i means the set of similarity between item i and another arbitrary item j.

Map-III: Map $\mathbb{R} < i, j, R_{i,j}, \bar{R}_i >$ on j, such that tuples with the same j are shuffled to the same machine in the form of $< i, R_{i,j}, \bar{R}_i >$.

If there are k Mappers, the complexity of the algorithm in each Mapper is $o(\frac{nm}{k})$.

Reduce-III: Take $< i, R_{i,j}, \bar{R}_i >$ and S, then output $B =< i, j, R_{i,j}, S_i >$ for each $< i, j, R_{i,j} > \in \Re$. The tuples are ordered by the user ID j.

If there are k Reducers, the complex of the algorithm in each machine is $o(\frac{nm}{k})$.

This phase prepares the data for the next step of prediction calculation. It integrates the existing rating record, average rating for each item and similarity tuples for each item.

Define the tuple $B =< i, j, R_{i,j}, \bar{R}_i, S_i >$, by which the prediction $P_{i,j}$ to item i by user j is computed according to Equation (3.2). The prediction matrix is defined as

$$P = \begin{vmatrix} \{P_{1,1}, & P_{3,1}, & P_{5,1}\} & \\ \cdots\cdots & & & \\ \{P_{2,j}, & P_{3,j}, & P_{4,j}, & P_{7,j}\} \\ \cdots\cdots & & & \\ \{P_{1,m}, & P_{2,m}\} & & \end{vmatrix} = \begin{vmatrix} P_1 \\ \cdots\cdots \\ P_j \\ \cdots\cdots \\ P_m \end{vmatrix}$$

where $P_j = \{P_{i,j} | \forall i \in [1, n], < i, j, R_{i,j} > \notin \mathbb{R}\}; \forall j \in [1, m]$. P_j means the set of prediction values of item i by an arbitrary user j who never rated the item i.

Map-IV: Map $B =< i, j, R_{i,j}, \overline{R_i}, S_i >$ on j such that tuples with the same j are shuffled to the same machine in the form of $< i, R_{i,j}, \overline{R_i}, S_i >$. If there are k Mappers, the complexity of the algorithm in each Mapper is $o(\frac{nm}{k})$.

Reduce-IV: Take $< i, R_{i,j}, \overline{R_i}, S_i >$ and I_{all} (which is the output of the Reducer-III), then output $< i, j, P_{i,j}, \overline{R_i} >$ for each $< i, j, R_{i,j} > \in \Re$. The tuple is ordered by the key j and $P_{i,j}$. If there are k Reducers, in each machine, the complex of the algorithm is $o(\frac{nm}{k})$.

In the last phase, the prediction rating for all users to all items is calculated, following which the recommendation list can be obtained.

In summary, throughout the four Map-Reduce phases, the ICF algorithm

has been implemented in parallel on MapReduce by splitting data files with the userID or itemID into the same Mapper or Reducer. The partition strategies ensure the local computation. In each Map-Reduce phase, the complexity of the algorithm for each node can be controlled by the number of Mapper and Reducer k when the size of the data increases.

3.1.5 Optimisation Algorithm

The The optimised resource allocation problem of auto-scaling cloud-based web services at the SaaS level can be described as below. Given a cloud-based parallel program and workload, how can an optimised volume of resources to satisfy the QoS of SLA be allocated?

The QoS of SLA can be simply measured by the execution time. This means that

$$T_p \leq T_{SLA} \tag{3.5}$$

where T_p is the amount of parallel execution time with p processors, and T_{SLA} is the maximal execution time defined by SLA.

Amdahls law [11] roughly models the performance of speedup S_p,

$$S_p = \frac{T_1}{T_p} \tag{3.6}$$

where T_1 is the amount of sequential execution time with a single processor. In an ideal scenario, the speedup has a linear relationship with the numbers of nodes with fixed data size. As shown in Figure 3.3, however, the real speedup is less than the ideal speedup. Moreover, when the number of cloud nodes exceed a special threshold, the speedup decreases as a result of the increased communication and management cost.

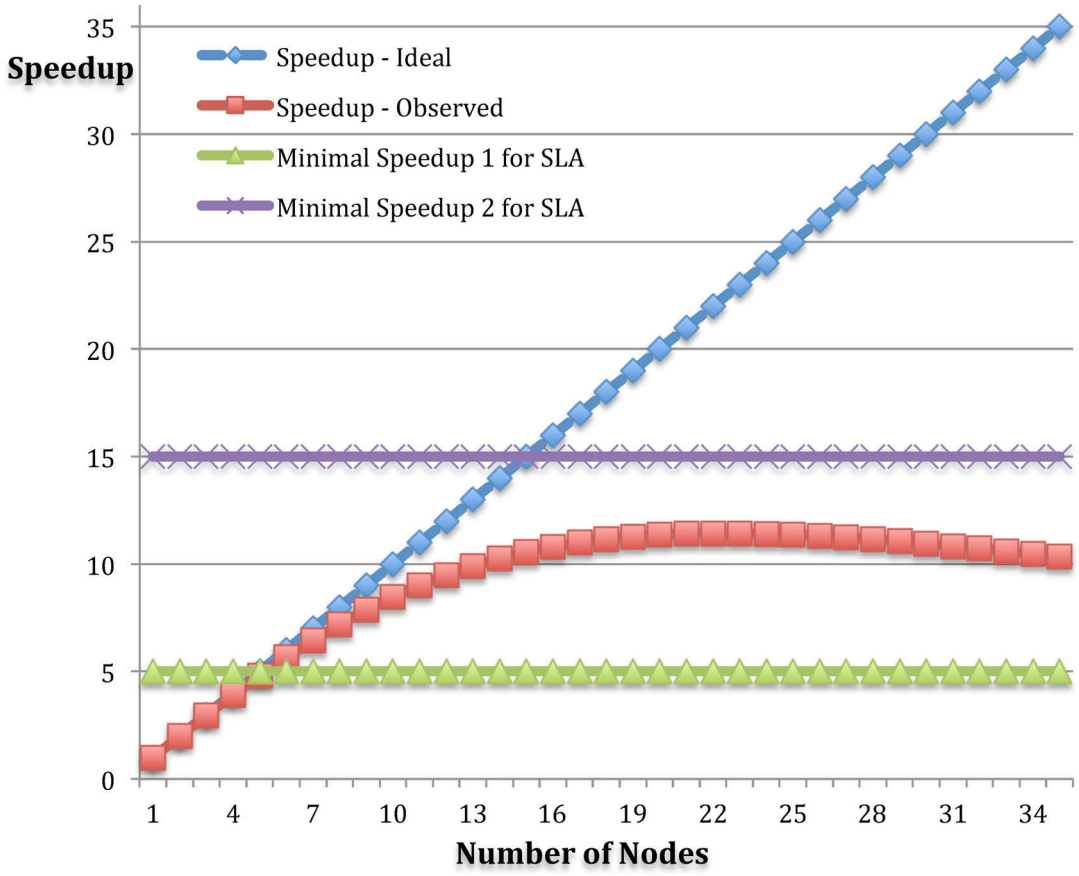


Figure 3.3: Speedup and Number of Nodes

By combining Equation 3.5 and 3.6, an equation can be obtained as below

$$S_p \geq S_{SLA} = \frac{T_1}{T_{SLA}} \tag{3.7}$$

where S_{SLA} is the minimise value of S_p by satisfying SLA.

As shown in Figure 3.3, the optimal value of p is

$$p_{opt} = \arg\min_{p} F(p) \; constrain \; to \; S_p \geq S_{SLA} \tag{3.8}$$

where $F(p)$ is the function of the model for the real speedup curve. To demonstrate the possible scenario of optimal speedup curve, two auxiliary horizontal lines for SLA examples are added in Figure 3. If the S_{SLA}, like the "Minimal Speedup 2 for SLA" line in Figure 3.3, is bigger than the maximal $F(p)$, there is no optimal solution because cloud-based web services can not achieve the required performance due to the bottleneck. If the S_{SLA}, like the "Minimal Speedup 1 for SLA" line in Figure 3.3, is not bigger than the maximal $F(p)$, the optimal value of p will be the cross point of the "Minimal Speedup 1 for SLA" line and the "Speedup - Observed " curve.

3.2 Experimental Evaluation

In this section, the performance of the ICF algorithm on the Hadoop cluster is investigated, focusing particularly on its scalability and efficiency. Scalability and efficiency are the most frequently used metrics in performance evaluation, because an algorithm that does not scale well is of limited value in practice.

3.2.1 Experiment Setup

All experiments in this work are performed on a Hadoop cluster consisting of three nodes, one node as the MasterNode and the other two nodes as the DataNodes. These nodes are common PC machines with Intel P4 CPU, 1G RAM, 80 G disk and all running Ubuntu Linux 10. All the machines are connected with one 100 Mbps switch.

In this research, each node can support a maximum of five Map or Reduce tasks and equal numbers of map and reduce tasks are set up. Because the map and reduce tasks cannot be executed at the same time, this platform can support 10 map tasks and 10 reduce tasks. The number of map or reduce tasks is recorded as the number of nodes. For example, if 20 map tasks and 20 reduce tasks are applied in one experiment, then the number of nodes is recorded as 20.

MoveLens [27] [114] [83] datasets, which consist of movie rating data collected using a web-based research recommendation system, are used in the experiments.

The datasets contain 943 users, 1670 movie items and about 54,000 ratings on a scale from 1 to 5. As algorithm accuracy and recall are not considered in this experiment, more data are produced based on the Movielens data format. When the data size is increased, the number of users will also be increased to keep the same ratio of sparse data, therefore, the computation load of the algorithm is able to maintain a liner increase.

3.2.2 Performance Evaluation

To measure the performance in terms of the scalability and efficiency of parallel algorithms, there are several performance metrics such as speedup, scaled speedup, sizeup, experimentally determined serial fraction and isoefficiency function [87]. Isoefficiency and speedup are two useful scalability metrics [111]. The former evaluates the performance of an algorithm-machine combination by modelling an isoefficiency function. The latter evaluates the execution performance change of a fixed size problem as the number of processors increases. In the experiments, the speed-up and isoefficency are applied as the performance metrics.

(1) Speedup: Amdahls law [11] roughly models the performance of speedup S_p,

$$S_p = \frac{T_1}{T_p} \tag{3.9}$$

where T_1 is the amount of sequential execution time with a single processor. T_p

is the amount of parallel execution time with p processors. If the algorithm is scalable, the speedup has a linear relation with the numbers of nodes with a fixed data size. Figure 3.4 shows how the speedup is almost linearly increased with the growth of node numbers on different data scales (10 K, 100K, 1M, 10M). Hadoop has the best speedup on the largest data size and therefore has good scalability in processing large-scale datasets.

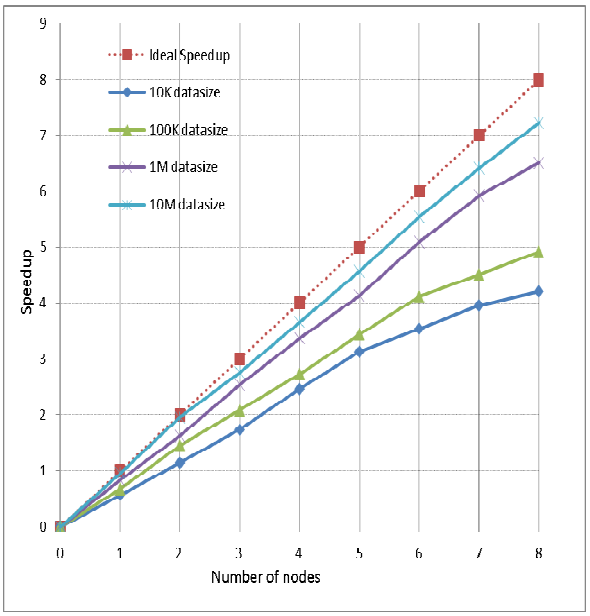


Figure 3.4: Speedup of Item-based CF Algorithm

(2) Isoefficiency: Kumar et al. [64] proposed the isoefficiency concept which measures the necessary workload increase to maintain efficiency. The isoefficiency function can be expressed as

$$E_p = \frac{S_p}{p} = \frac{T_1}{T_p \times p} = \frac{T_1}{T_1 + T_0} = \frac{1}{1 + T_0/T_1} \tag{3.10}$$

where T_0 is the time of communication cost in a parallel process. If the algorithm is scalable, E_p should remain constant. For the fixed E_p , $T_1 = f_E(P) = kP$ and $f_E(P)$ should be a linear function. That is $T_1 = T(m) = lm$ which means the size of data m has a linear relationship with the number of nodes P. Therefore, the number of nodes P has a linear relationship with the size of the data m. i.e. $P = \frac{l}{k}m$.

Thus if P (the number of nodes) has a linear relationship with m (the data size), it means that the algorithm is scalable. As shown in Figure 3.5, by setting a fixed execution time (10 mins, 20 mins, 30 mins), the number of nodes has good linear increase with growing data size. This also demonstrates that the longer fixed execution time achieves better performance.

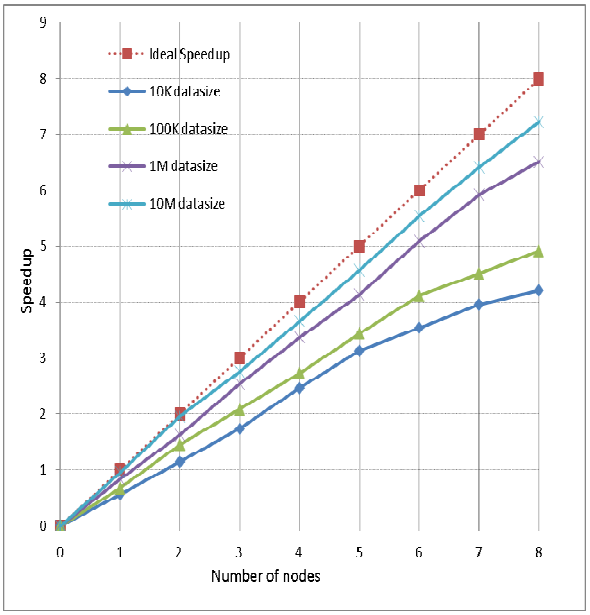


Figure 3.5: Isoefficiency Function for Varying Nodes and Data-size with Fixed Running-time

3.2.3 Summary

In this chapter, a scaling-up ICF algorithm on MapReduce is parallelised, by splitting the whole data file and allocating those partitions with the same userID or itemID into the same Mapper or Reducer. To minimise the communication cost, effective partition strategies are developed to realise the local computation in each Map phase and each Reduce phase. The experiment results show that good scalability and performance efficiency is obtained by the proposed scaled up ICF algorithm, which is executed in a Hadoop cluster. Due to the limited experimental resources, only three physical nodes are implemented in the Hadoop cluster. This work is a case study of SaaS-level resource optimisation for auto-scaling cloud-based web services in which a novel solution to the key component of time-consuming computation is studied. The proposed methods can be adapted to other auto-scaling cloud-based web services which have time-consuming computation.

Chapter 4

ACWS Optimisation on PaaS

TDM In the PaaS models, cloud providers deliver a computing platform that typically includes an operating system, a programming language execution environment, a database, and a web server. The auto-scaling character of the computing platform enables the cloud-based web services provider self-adaptively satisfy the time-varying workload. For a rapidly-completed service, the cloud platform automatically assigns requests to different cloud nodes. For a time-consuming service, the cloud platform will automatically distribute the requests to different groups of cloud nodes. As discussed in Chapter 3, each group of cloud nodes can accomplish a parallelisable time-consuming service task within the given time. In this chapter, we only focus on the cloud platform for rapidly-completed services, because it is easy to adapt the platform for time-consuming services by treating a nodes group as a resource unit.

To support auto-scaling cloud-based web services, the cloud platform needs not only the basic functions mentioned above but also special functions such as auto-configuration, and optimised resource allocation. The platform needs to support multiple kinds of web services. If each server is configured to support all kinds of services, namely "one-server-all-services", the burst visits of one service will block other services. The robustness of the system will be a problem with this configuration model. If each server is configured to support only one service, namely "one-server-one-service", consider a general scenario: within a special time period, there may be many visits requesting for one kind of service but few visits requesting other services. In this scenario, the servers configured for the numerous visits service will be very busy, while other servers will be idle, and system utilisation will be very low. With the given workload, a good configuration should be able use minimal resources to satisfy the QoS of SLA. To achieve an optimal solution for configuration, this chapter proposes an auto-configuration and optimised resource allocation method at PaaS level.

This chapter aims to investigate the configuration problem in the cloud computing environment, considering multiple conflicting SLA and supporting dynamic reconfiguration when workloads change. A self-adaptive configuration optimisation method is proposed to solve this problem. By using queuing theory and statistic techniques, in particular, the problems of modelling and computation for SLA metrics are addressed based on utility function for a given workload

distribution. A multi-objective genetic algorithm is also developed for this optimisation system to realise the self-adaptive configuration. This method can guide cloud customers to purchase appropriate resources and make decisions about deployment configuration such as scale, scheduling and capacity.

The deployment of cloud service problem is a combinatorial optimisation problem which ensures the optimal mapping between each service of an application and the cloud computing resource. Since there are a large number of various types of cloud computing servers and many services in one application, a huge number of possible deployment plans can be generated. An example deployment configuration is shown in Figure 4.1. Each deployment configuration has an arbitrary number of deployment plans and each deployment plan can process arbitrary number of services as a result of which the operation of service throughput and latency is improved. This research work supports the idea that multiple like services are not operated on the same deployment plan, since multiple like services deployed on one machine do not contribute to an improvement in performance .

4.1 Methodology

In this section, the architecture of the proposed optimisation system is presented and the modelling of the system is described.

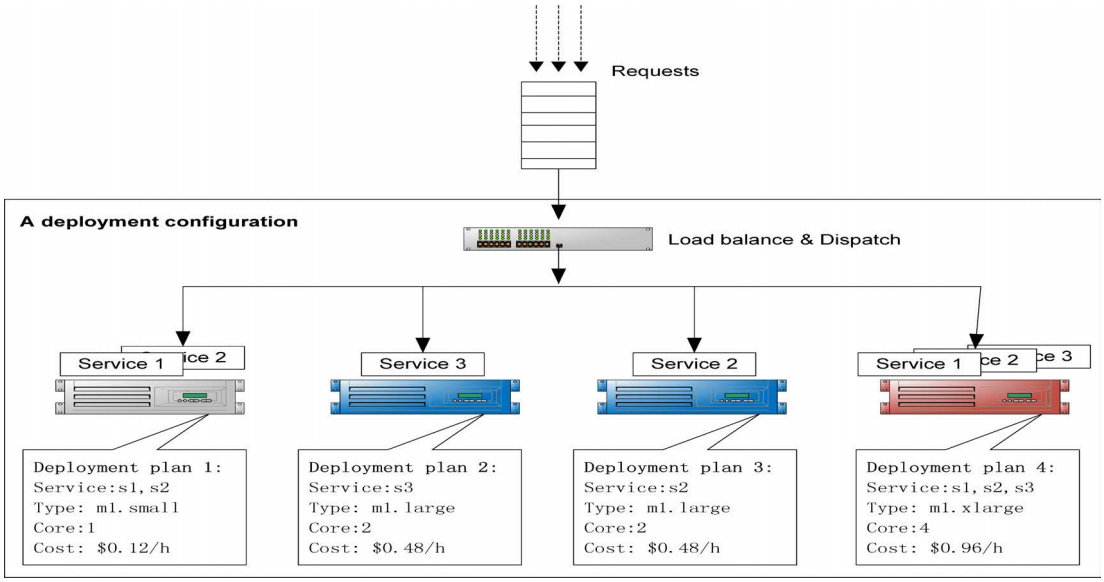


Figure 4.1: An Example of Deployment Configuration

4.1.1 Architecture of Self-adaptive Configuration Optimisation System

The architecture of the proposed self-adaptive configuration optimisation system, its components and their interaction are shown in Figure 4.2. The system has four main components: workload analysis and prediction model, SLA computation model, optimisation and scheduling model, and resource configuration model. This system is a closed loop self-adaptive control model. When the request workload changes, the optimisation and scheduling model executes the optimisation algorithm to determine the best configuration for the system at regular intervals (i.e. reconfiguration intervals), considering the previous execution results and cloud computing resources information (as shown in Figure 4.3).

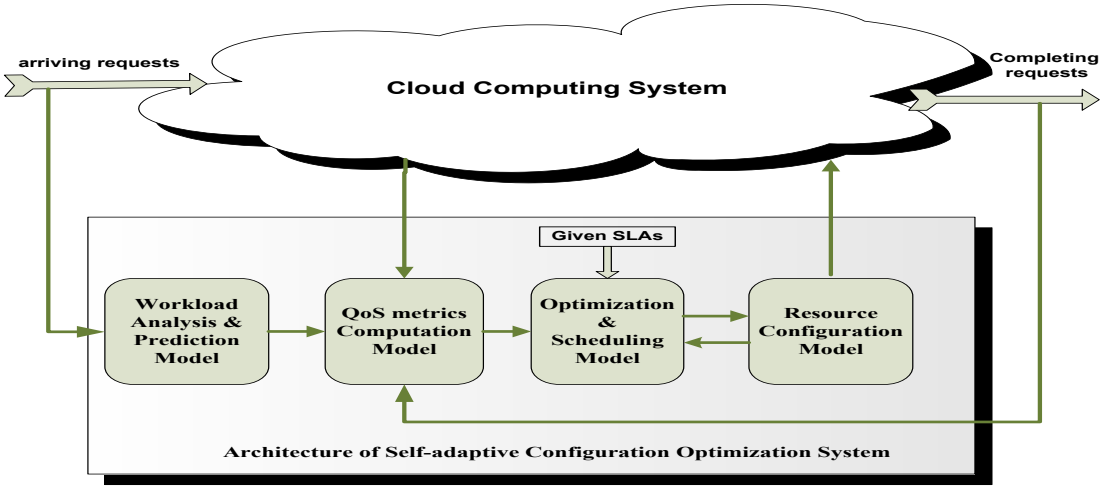


Figure 4.2: Architecture of Self-adaptive Configuration System in Cloud Computing

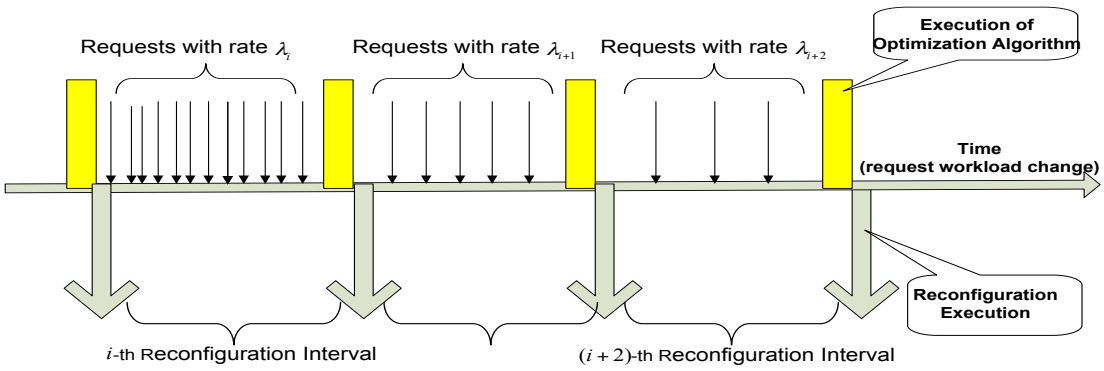


Figure 4.3: Self-adaptive Reconfiguration Interval

The SLA computation component collects utilisation data from the cloud computing system (e.g., CPU and disks) as well as the account of completed requests, which allows the component to compute the SLA objectives (e.g. throughput, latency, cost). The service demand of a request, i.e., the total average service time of a request at a resource, can be computed as the ratio of the resource utilisation and system throughput. The values of SLA objectives computed by

this component are used as one of the input parameters to the optimisation and scheduling model.

The workload analysis and prediction model analyses the stream of arriving requests, computes and stores statistics for the workload intensity, such as the average arrival rate, and applies statistical methods [12] [108] to model and predict the intensity of the workload in the next reconfiguration execution interval. The current or predicted values of workload intensity computed by this component are also used as input parameters of SLA computation model.

At the start point of each reconfiguration interval (see Figure 4.3), the optimisation and scheduling model runs the optimisation algorithm. This algorithm takes into account the desired SLA objectives, the arrival and departure processes, and performs a multi-objective genetic algorithm to search a trade-off configuration in the state space of possible deployment configurations. Once the resource configuration model determines the best configuration for the workload intensity level provided by the workload analysis and prediction model, it will send reconfiguration commands to the cloud computing systems. The system is then realised with self-adaptive configuration optimisation.

4.1.2 SLA Metrics Modelling using Utility Function

In this section, a scenario in which a given workload distribution with the same rate request in the reconfiguration interval is considered, the SLA metrics are modelled and a computational method of Stochastic SLA Metrics is studied. A corresponding optimisation algorithm (genetic algorithm) is then developed in Section 4.1.4.

The SLA are defined as performance metrics (throughput, latency, CPU usage) and cost. These four objectives are conflicting and are variously weighted by different clients, therefore the optimisation value is usually a trade-off of optimal SLA rather than a single optimal result. To handle this multi-objective optimisation problem, the notion of utility function is applied to determine the combined usefulness of cloud services as a function of the various SLA. Utility functions are used quite often in economics and have been extensively used in autonomic computing [37]. However, the use of utility functions to determine the optimal mix of SLA in cloud computing as presented here is novel. The following notations are expressed to formalise the problem of optimal selection of SLA.

Let objective tuple in SLA be: $SLA = \{T, L, U, C\}$, where

- T (throughput): SLA on transaction throughput; $T = (T_1, T_2, \cdots, T_n)$, T_i is the $i-th$ interval throughput.

- L (latency): SLA on response time or waiting time of per service instance

processing; $L = (L_1, L_2, \cdots, L_n)$, L_i is the $i-th$ interval average response time.

- U (CPU Usage): SLA on CPU utilisation; $U = (U_1, U_2, \cdots, U_n)$, U_i is the $i-th$ interval CPU utilisation.

- C (cost): SLA on resource utilisation cost; $C = (C_1, C_2, \cdots, C_n)$, C_i is the $i-th$ interval resource utilisation cost.

For the Utility Function formulas of $SLA = \{T, L, U, C\}$ metric, we use the following sigmoid curve [99] [110] [98]:

$$Uti = K\frac{e^{\alpha(\beta-\varepsilon)}}{1+e^{\alpha(\beta-\varepsilon)}} \tag{4.1}$$

where K is the normalising factor. α is a sensitivity parameter that defines the sharpness of the curve and β is the associated SLA goal. The sign of α determines whether the sigmoid decreases ($\alpha > 0$) with ε or increases ($\alpha < 0$) with ε. The maximum value of the utility function defined above is 1. These values occur at $\varepsilon = 0$ for latency and cost, $\varepsilon = 1$ for CPU usage, and $\varepsilon \to \infty$ for throughput. A decreasing utility function is used for latency and cost, and an increasing function is used for throughput and availability. The utility function of SLA is as follows:

$$Utility(SLA) = w_1 Uti(T) + w_2 Uti(L) + w_3 Uti(U) + w_4 Uti(C) \tag{4.2}$$

The first term $Uti(T)$ in Equation (4.2) is the throughput utility function, the subsequent terms are latency, CPU usage and cost utility function, respectively. Each value of these individual utility functions belongs to the [0, 1] range. w_1, w_2, w_3 and w_4 represent the weights associated with throughput, latency, CPU usage, and cost, respectively, which are used to indicate the preference of objectives in SLA for cloud clients, and $w_1 + w_2 + w_3 + w_4 = 1$, $Utility(SLA) \in [0, 1]$.

Now, a cloud client is faced with the problem of selecting the appropriate resource deployment that will maximises the utility function subject to SLA constraints. This non-linear constraint optimisation problem is shown below:

$$
\begin{aligned}
& maxUtility(SLA) = f(T, L, U, C) \\
& \textit{subject to} \\
& T_{min} \leqslant T \leqslant T_{max} \\
& L_{min} \leqslant L \leqslant L_{max} \\
& U_{min} \leqslant U \leqslant U_{max} \\
& C_{min} \leqslant C \leqslant C_{max}
\end{aligned}
\tag{4.3}
$$

4.1.3 Stochastic SLA Metrics Computation

This section describes the stochastic SLA computation method used in the modelling of the utility function optimisation of SLAs. To estimate the performance

of distributed systems such as cloud system, queuing theory is a well-established method which has been studied in a number of research works [82]. This research work uses queuing theory and statistical techniques to estimate the SLA objectives that are throughput, latency (i.e. waiting time), CPU usage and cost. The cloud server cluster can be modelled as a M/G/m queuing system which indicates that the randomly arrived requests is approximated modelled as a Poisson process. The Poisson distribution is shown as:

$$f(k;\lambda) = \sum_{k=0}^{m} \frac{\lambda^k e^{-\lambda}}{k!} \tag{4.4}$$

where,

- e is the base of the natural logarithm ($e = 2.71828\ldots$);
- k is the number of occurrences of an event;
- $k!$ is the factorial of k;
- λ is a positive real number, equal to the expected number of occurrences during the given interval;
- μ is the maximum number of processed requests per unit time for single cores;
- n is the number of cores;

- s_i is the cost computation in ith interval time.

From Equation (4.4), the Expectation is $E(f(k;\lambda)) = \lambda$, that is the average number of arrival requests per reconfiguration interval. Since the given distribution with the same rate of requests is supported and μ is known, the throughput can be computed as:

$$T = \begin{cases} \lambda, & \lambda \leqslant n\mu \\ n\mu, & \lambda > n\mu \end{cases} \tag{4.5}$$

and the approximation of CPU usage is :

$$U = \rho = \frac{\lambda}{n\mu} \tag{4.6}$$

The probability function of the latency in M/G/m queuing system is [51] [96] :

$$P_r(W \geq t) = (\frac{np_n}{n-\rho})e^{-(n\mu-\lambda)t}, t \geq 0 \tag{4.7}$$

where,

$$p_n = p_0(\frac{\rho^n}{n!})$$

and

$$p_0 = \left[\sum_{k=0}^{n-1} \frac{\rho^k}{k!} + \frac{\rho^n}{(n-1)!(n-\rho)}\right]^{-1}$$

Therefore, the calculation of the latency is the expectation shown as

$$L = E(P_r(W \geq t)) = \frac{np_n}{(n-\rho)(n\mu-\lambda)} \tag{4.8}$$

Then the ith interval resource utilisation cost is

$$C_i = ns_i \tag{4.9}$$

4.1.4 Optimisation Algorithm

In this section, a multi-objective genetic algorithm is developed for configuration optimisation in cloud computing. The detailed algorithm is specified in Algorithm 4.1.4. This algorithm is designed to seek individuals that satisfy given SLA and exhibit optimal utility value considering trade-off among QoS objectives in SLA. At each generation, two parents, $p1$ and $p2$, are selected with binary tournaments. They reproduce two offspring by performing the crossover operator with one-point crossover on genes. Then off-spring's genes are then mutated. A mutation operator is designed to increase or decrease resources. To generate the ability to dynamically change a deployment plan to a deployment configuration, the mutation of operator first adds an empty deployment plan (randomly selected), for example, no service deployed on it. After executing mutations, the mutation operator examines each deployment plan and releases empty deployment plans

Algorithm 4.1 A multi-objective genetic algorithm for cloud resource configuration optimisation

Initial:
MaxGeneration $\leftarrow$ maximal generation count ;
$u \leftarrow$ population size ;
$P \leftarrow$ random u individuals ;
$g \leftarrow 0$;

1: **while** $(++g > MaxGeneration)$ **do**
2: **for** each p in P **do**
3: $t \leftarrow$ throughput with configuration p;
4: $l \leftarrow$ latency with configuration p;
5: $u \leftarrow$ CPU usage with configuration p;
6: $c \leftarrow$ cost with configuration p;
7: $u \leftarrow utility(t, l, u, c)$;
8: $p.fitness \leftarrow u$;
9: **end for**
10: **if** (size of $(P) > u$) **then**
11: $P \leftarrow$ Top u individuals of P based on fitness value;
12: **end if**
13: $Q \leftarrow$ new empty population ;
14: **while** (size of $(Q)! = u$) **do**
15: $r, s \leftarrow$ random two individuals from P ;
16: $p1 \leftarrow max(r, s)$ based on fitness value ;
17: $r, s \leftarrow$ random two individuals from P ;
18: $p2 \leftarrow max(r, s)$ based on fitness value;
19: $tmp1, tmp2 =$ Crossover $(p1, p2)$;
20: **if** $(tmp1, tmp2)$ are not better than $(p1, p2)$ **then**
21: $tmp1, tmp2 \leftarrow p1, p2$;
22: **end if**
23: $tmp1 \leftarrow$ mutation $(tmp1)$;
24: **if** $tmp1$ is better than $p1, p2$ and $tmp1$ does not exist in Q **then**
25: add $tmp1$ to Q;
26: **end if**
27: $tmp2 \leftarrow$ mutation$(tmp1)$;
28: **if** $tmp2$ is better than $p1, p2$ and $tmp2$ does not exist in Q **then**
29: add $tmp2$ to Q;
30: **end if**
31: **end while**
32: $P \leftarrow P + Q$;
33: **end while**

to form a deployment configuration. In this way, it realises the self-adaptive on-demand cloud service deployment. This reproduction process is repeated until MaxGeneration is reached. To fulfil both requirements, this multi-objective genetic algorithm for configuration optimisation in cloud computing uses the utility function of SLA (shown in Equation (4.3)) as the fitness function which satisfies the given SLA constraints. The fitness function for feasible individuals is designed to encourage them to improve their SLA values in all objectives and maintain diversity in their population of SLA values.

4.2 Experimental Evaluation

In this section, the performance of the proposed optimisation algorithm is investigated.

4.2.1 Experiment Setup

In this section the proposed methods of Genetic Optimisation based Service Configuration (GOSC) are evaluated on the simulator - CloudSim [21]. The number of **server types** is three with the relative capability 1:5:20, and the price of different servers maintains the same ratio of capability. Unlimited number of each server type are available.

Ten different **service types** are considered. On the slowest server, the running

time for each service type is a random variable following Gaussian distribution whose mean value is randomly selected. On different servers, the running time is decreased based on server capability.

This work simulates the **workload** from a real-world datasets which is Sogou [1] search log dataset, the request scale is shown in Figure 4.4. In the time-varying workload scenario, the scale of requests are changed hourly in a different time slot, because the dataset only records the statistical number of visits for a time period. To simulate the visiting arrival sequence, the arrival time is modelled as a random process with the mean value from the real-world datasets.

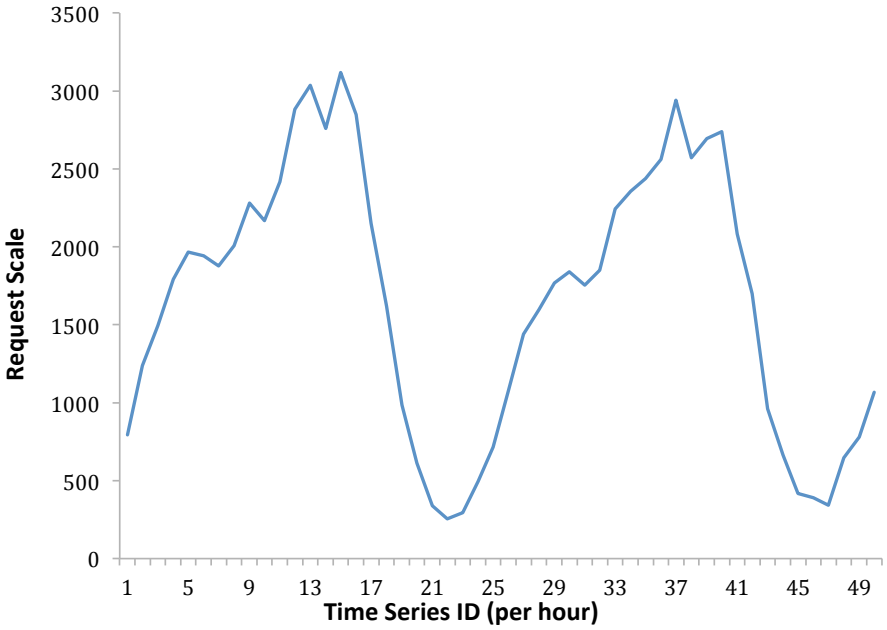


Figure 4.4: Request Scaling for Time-varying Workload

The Threshold Detection Method (TDM) is treated as the **baseline method**.

[1] http://www.sogou.com/labs/dl/q-e.html

Most public cloud platform, e.g. Amazon, provide users with a threshold detection tool for auto-scaling. With this tool, the user can specify the threshold for several indicators of system performance. Once the threshold is broken, the cloud system will automatically scale up or scale down based on the specified action rules of users. This method is simple and straightforward. It is a reactive resource allocation method which adjusts the volume of resources by reacting to the changes in the workload.

4.2.2 Performance Evaluation

With the given time-varying workload series from the real-world dataset, the proposed method is separately evaluated with the baseline schema. The performance between these two methods is compared on four indicators: Utilisation, Cost, Latency, and Throughput.

In Figure 4.5, two curves are drawn to represent the cost for time-varying workload. Our proposed method (GOSC) costs less than the baseline method (TDM - Threshold Detection Method). The cost is the volume of the capacity for allocated resources. It treats the weakest server's capacity as one, and other power servers can be represented as another number based on the capacity comparison. Then a numerical value can be applied to represent the volume of the capacity for all allocated resources. Figure 4.5 demonstrates that the cost increases or

decreases according to the changes in workload. The demand volume of a resource is the ideal optimal value. The cost curve of our proposed method is closer to the ideal optimal value than the cost curve of the baseline method.

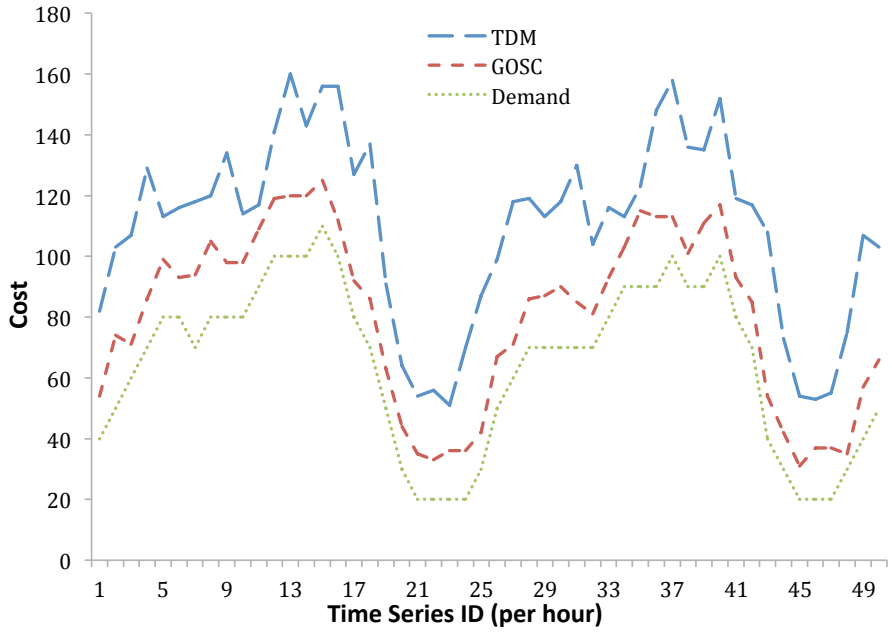


Figure 4.5: Cost for Time-varying Workload

At the same time, our proposed method achieves higher CPU utilisation, as shown in Figure 4.6. To calculate the average CPU utilisation of the system, the mean value of utilisation is calculated by also considering the capacity as well, because different servers have different capacities and utilisation.. The calculation can be represented by Equation 4.10 below.

$$U = \frac{\sum_{i=1}^{N} C_i * U_i}{\sum_{i=1}^{N} C_i} \tag{4.10}$$

where C_i represents the capacity of the ith server by comparing with the weakest server, and U_i is the observed CPU utilisation of the $i$the server. In Figure 4.6, our proposed method has higher utilisation than TDM due to the optimisation of the allocated resource and service configuration.

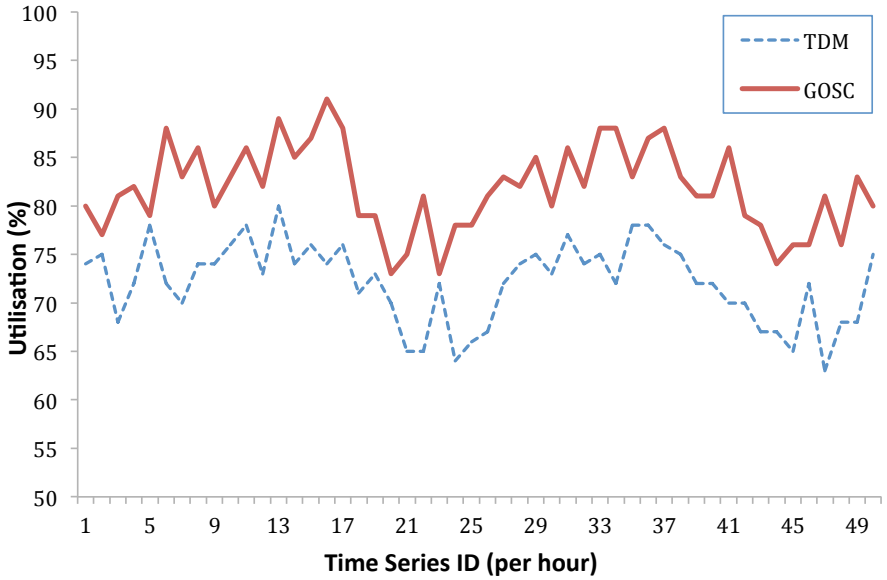


Figure 4.6: Utilisation for Time-varying Workload

To measure the latency of the system, the latency of all requests are recorded within each hour, then the average latency is obtained for each hour.

$$L = \frac{\sum_{i=1}^{M} L_i}{N} \tag{4.11}$$

where L_i is the latency of the Mth request within the given one-hour-time-slot. As shown in Figure 4.7, our proposed method has lower latency because the configuration for different kinds of services is optimised while TDM only adjusts

the volume of the resource when the performance decreases to a certain threshold.

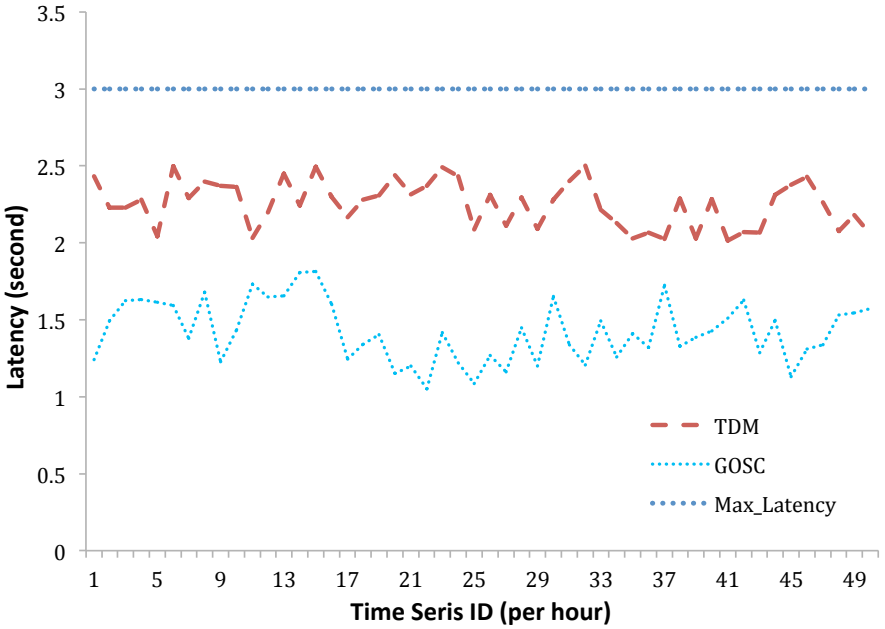


Figure 4.7: Latency for Time-varying Workload

The throughput for each service type in the system is evaluated. Because the service will be configured on only some of allocated servers, the throughput is only calculated on those configured servers rather than all allocated servers. In Figure 4.8, the throughput results for one service type are illustrated, showing that the baseline method has bigger throughput than our proposed method because the based line method applies more servers than our method.

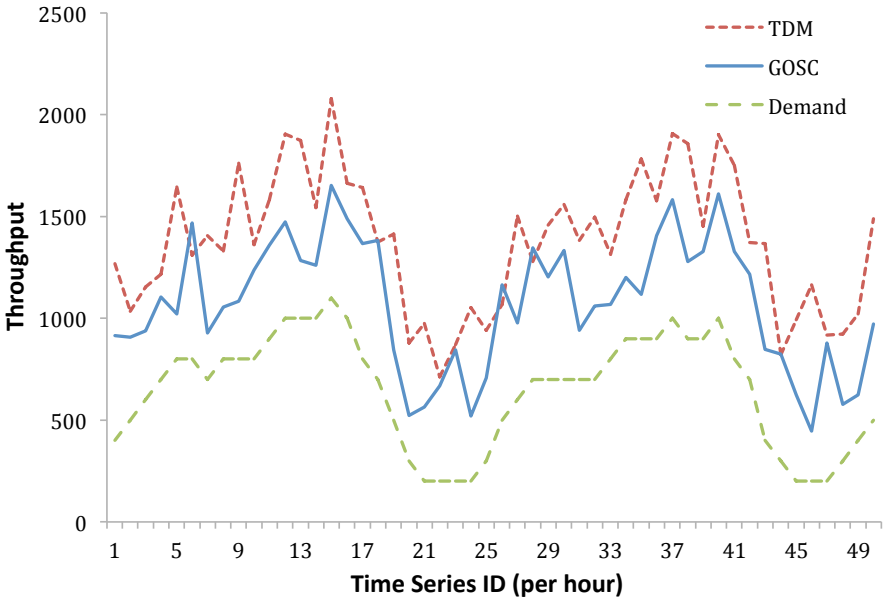


Figure 4.8: Throughput for Time-varying Workload

4.3 Summary

This chapter discusses the PaaS level optimisation for auto-scaling cloud-based web services. To support the auto-scaling of cloud-based web services at the PaaS level, a self-adaptive method is developed with auto-configuration and optimised resource allocation functions. This chapter particularly studies the cloud service configuration optimisation considering system performance, resource provision and scheduling, multiple conflicting SLA, and changing workloads. The self-adaptive cloud service configuration optimisation is developed, considering multiple conflicting SLA and supporting dynamic reconfiguration when workloads change. The designed cloud service configuration optimisation system and algorithm can be applied to any cloud computing platform.

As the experiment results show, our proposed method save server cost, increases server utilisation and reduces average latency in a real-world request visiting environment compared to the baseline method,.

Chapter 5

ACWS Optimisation on IaaS for VMs

Web service providers need to purchase computation resource from cloud providers, and cloud providers typically charge IaaS services by a utility computing basis: cost reflects the amount of resources allocated and consumed. To save cost for web service providers, this chapter propose an optimisation cloud resource allocation method for the web services. Particularly, in this chapter, we focus on the computation resource (e.g. VMs) for computation-intensive web service. Moreover, due to space limitation, the proposed method targets fast-complete service only. Because a time-consuming task can be divided to several fast-complete tasks, and the proposed method is able to be used for this scenario as well.

In most cloud platforms, computation resources are charged with an entire

time slot, such as an hour, a day or a month. To balance the cost of resource rescheduling and running cost on current pricing models, this work adjusts the required resource hourly. Particularly, time-varying workload is firstly predicted by combining time series and predictive analysis. Then the arrival pattern of requests will be modelled by using Queueing Theory. Thirdly, an optimal volume of resource will be is required by satisfying both cost and QoS. Lastly, to handle unpredictable request burst, a real-time reactive method is developed to apply additional cloud resource with higher price.

This proposed VM auto-scaling method is developed for cloud tenants who have time-varying resource demands. In general, a cloud tenant wants to make accurate decisions on how many VMs should be used and when to scale up or down. However, it is not a trivial thing for cloud tenants to make such a decision on time-varying demands scenario. The scaling decision is decided by the following three main aspects:

(1) optimal VM consumption in varying time units (the biggest impact);

(2) boot-up time of VMs (scaling up);

(3) task migration on VMs (scaling down).

According to current related work, there are two major methods for facilitating the VM auto-scaling decisions of cloud tenants:

- Reaction-based auto-scaling method: which reacts to the current system based on not only a set of monitored variables (e.g. CPU utilisation) offered by cloud infrastructure vendors or third-party tools but also some auto-scaling rules set by human operator (usually a cloud tenant). This kind of method has often been referred as rule-based or trigger-based auto-scaling method.

- Prediction-based auto-scaling method: which treats the future demand as a predictable value by utiliszing the past demands as a time series or mathematical model. The demand of next period is predicted or estimated by the constructed model. Then, an auto-scaling decision will be made based on the predicted demands.

The Reaction-based method is easy for understanding and operating. But it requires cloud tenants to select logical combination of variables which is hard to achieve cost-effective results on time-varying workload. The Prediction-based method does not require extra efforts of cloud tenants. However, it is difficult for cloud tenants to allocate suitable VMs using the predictive-based method when the workload changes suddenly or significantly.

Considering the pros and cons of the above two methods, this chapter proposes a hybrid method, namely data-driven VM (**DD-VM**) auto-scaling method, to solve the time-varying VM consumption for cloud tenants (e.g. web applica-

tions provider). A data-driven dynamic prediction model is developed to forecast demands volume by using machine learning techniques to train a model with both history and current data. The model will be updated in each time unit according to the changes of demands volume. To solve under-estimation and over-estimation problems of prediction, an adaptive prediction error handling method is designed. Further, an optimisation method is described to obtain the optimal number of VMs required to achieve the minimal resource consumption cost while satisfying the performance level (e.g. latency) defined in the SLA (service level agreement). In addition, a reactive method is operated to handle unpredictable bursts. According to the optimal value, an auto-scaling decision is executed in each time unit. The proposed method is implemented on Amazon AWS and is evaluated by applying three real-world web log datasets. Experiment results demonstrates that the proposed method operate VM auto-scaling with few prediction errors, as well as optimal resource allocation with scalar cost-latency trade-off and few SLA violations. The proposed method is able to be extended easily to large-scale systems, as it does not have to construct and maintain a complex global structure.

The main contributions of this chapter are summarized as follows:

1. It proposes an efficient and robust method, that enables cloud tenants to adaptively and quickly achieve optimal VM consumption under time-varying workloads by utilising the advantages of reactive and predictive-

based methods.

2. It develops a data-driven decision making method for auto-scaling VM consumption, which is dynamically determined by modelling and analysing the data stream while not being restricted by specific rules.

3. It proposes a combined method in which a dynamic predictive-based method deals with the situation that the time-varying consumption has periodical pattern and a reactive method addresses the circumstance that the consumption is unpredictable.

5.1 Methodology

In this section, the overview of the proposed method is presented and the modelling of system is described.

5.1.1 Overview of the Method

The proposed method scales the cloud resource up or down (or NOP : do-nothing operation) by time-unit re-allocation based on predicted optimal resource demands. Web application providers can specify their budgetary constraints and SLA in respect of latency for their applications. In each time-unit, web application providers can be notified of the total cost, SLA violations and re-allocation

state (e.g. scaling up or down or NOP) by using the optimal resource auto-scaling method. Notice that in practical applications, an unpredictable burst of number of requests will happen as a similar situation as which the Animoto experienced. To tackle this unpredictable scenario, this method monitors the waiting queue of requests to be processed in real-time. Once the length of the queue is bigger than a threshold, the method could dynamically append VMs to process the exceeding number of requests. Figure 5.1 illustrates the overview execution paradigm facilitated by the method.

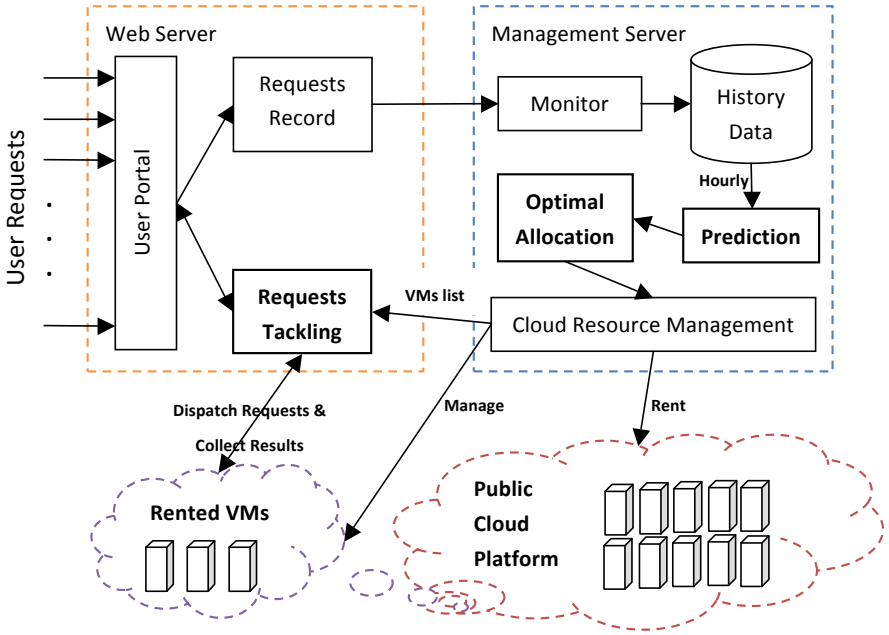


Figure 5.1: Overview of Optimised Cloud Resource Auto-scaling

As shown in Figure 5.1, the main steps of the proposed method are outlined as follows.

(1) to collect request records as the history data;

(2) to analyse the history visiting data hourly and predict the number of requests for the next time-unit (Sub-section 5.1.2);

(3) to discover the optimal number of VMs by utilising the Optimisation Model (Sub-section 5.1.3);

(4) to scale the resource (VMs) up or down or NOP from a public cloud platform.

5.1.2 Prediction Model

Because launching a VM instance takes several tens of seconds to minutes, a predictive-driven resource scaling method is proposed. As the parameters of a VM underlying resource, such as CPU, memory, I/O or network bandwidth, are not necessarily dependent [109], it is not trivial to model resource demand prediction directly within these parameters (i.e. resource-level). This work predicts the web request distribution in each time-unit. Subsequently, the resource demand is modelled based on the predicted web request distribution at a VM-level (that is, by considering the number of VMs, rather than the number of configuration parameters as the resource demand quantity).

(1) Definition

To predict the number of web requests, the web request data is generally denoted as a time series [113]:$\{X(t); t \in T\}$, where T is an index of the time fragment, and $X(t)$ is the random variable, representing the total number of requests that arrive

in the t time fragment. The prediction problem can be defined as follows: given the current and past observed values $(X(t-k), X(t-k+1), ...X(t-1), X(t))$, predict the future value $X(t+p)$, where k is the length of the history data used for prediction and p is the predictive time fragment.

(2) Key Features Identification

Most online web requests have a seasonal or periodical behaviour to some extent, we design a novel Linear Regression method for prediction by using an auto-correlation function to identify the key features.

For instance, a mail server usually experiences the highest web traffic volume every Monday morning, while at midnight, web requests drop to a low level; also, the number of requests will be much higher on weekdays than on weekends. Many more users may request a mail service on festivals and holidays than on other days. Therefore, the web requests behaviour pattern can be established and key features such as hourly, daily, weekly, monthly, seasonally, etc., can be identified by analysing the history data.

We represent a web requests time series as $(X(t-k), X(t-k+1), ...X(t-1), X(t))$, where $X(t)$ is the total arrived requests within the (t)th time fragment. The web request in the $(t+1)$th time fragment is depend on the request volume in the (t)th fragment and other previous fragments. A linear model is implemented

below to model the relationship between different time fragments.

$$X(t) = \sum_{i=1}^{N} w_i X(t-i) \tag{5.1}$$

where w_i is the important factor of the (i)th previous fragment, and N is the total number of previous time fragments.

Based on Equation (5.1), $w_i (i = 1, ...N)$ can be estimated by utilising a linear regression method to obtain the prediction model. If the related time fragments are too many, an over-fitting problem will occur and prediction accuracy may be reduced. The top key features that mainly determine the predicted value should be identified. In this work, the auto-correlation function is applied to identify the key features [107]. For the request state in each time fragment $X(t)$, its auto-correlation with another request $X(t-i)$ is calculated by

$$\rho_{t,t-i} = \frac{E\{[X(t) - \mu][X(t-i) - \mu]\}}{\sigma(t)\sigma(t-i)} \tag{5.2}$$

For different i, a vector $V = \{\underset{t}{mean}(\rho_{t,t-i}) | i \in [1, N]\}$ is obtained. K elements are selected with top values from the sorted vector V as the K key correlated features. These selected elements are composed of a new vector $\bar{N}$. For $t' \in [1, \bar{N}]$,

the linear regression model can be estimated as follows:

$$X(t) = \sum_{t'=1}^{K} w_{t'} X(t') \tag{5.3}$$

(3) Modelling the Relationship between Cost and Latency

To estimate the relationship between web request volume, cost and latency, the following are taken into consideration in the proposed method: (1) cost (C) prediction depends on the number (M) of VMs changing, e.g C=$f(m)$; (2) latency (L) consists of execution time (T_s) and waiting time for executing (T_q); (3) the arrivals of requests to be processed on VMs obey a Poisson distribution with rate λ , and the executed requests on VMs are also considered as a Poisson distribution with rate μ. For convenience, each web server is installed on one VM and all VMs belong to the same type of instance with the same process capacity. These allocated VMs come from an infinite cloud-based resource pool.

The queueing theory technique is applied to model this relationship and the arrival-execution of requests on VMs is considered as a birth-death process, which is a special Markov chain [46], as shown in Figure 5.2.

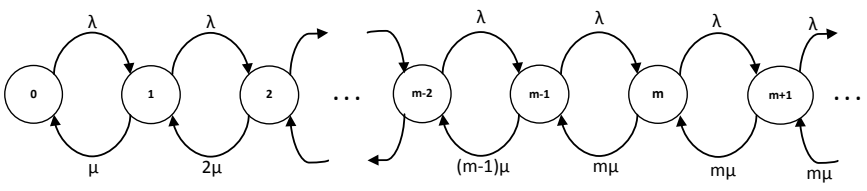


Figure 5.2: Transition Rate for the Web Requests Process on VMs

Due to the allocation of multiple servers (or VMs) in the method, this process of the arrival-execution of requests on VMs is modelled as $M/M/m$ queueing: the arrivals are Poisson distribution with rate λ ($\lambda = E(X(t))$, and each VM has an independent and identical distribution exponential execution-time distribution with mean μ . Since the execution-time of the web request on a given type of VM can be obtained by experiments, the execution-time T_s is known and $\mu = 1/E(T_s)$.

There are m VMs in the system and one web request is allocated to be processed on each one VM in parallel at every instant time-point with a constant rate, so that the "birth" rate is $\lambda_n = \lambda$ for all n. The rate of request completions (or "deaths") depends on the number of VMs in the system. If there are m or more requests in the system, then all m servers must be busy at the instant time-point. Since each VM processes requests with rate μ, the combined process-completion rate for the system is $(m\mu)$. When there are fewer than m customers in the system, e.g. $i < m$, only i of the m VMs are busy and the combined service-completion rate for the system is $(i\mu)$. Hence μ_i may be written as

$$\mu_i = \begin{cases} i\mu & 1 \leqslant i < m, \\ m\mu & i \geqslant m \end{cases} \tag{5.4}$$

Based on the Markov chain in Figure 5.2, the steady-state probabilities p_i is

obtained

$$p_i = \begin{cases} \frac{\lambda^i}{i!\mu^i} p_0 & 1 \leqslant i < m, \\ \frac{\lambda^i}{m^{(n-m)} m!\mu^i} p_0 & i \geqslant m \end{cases} \tag{5.5}$$

To obtain the value of p_0, the probabilities must add to 1 ($\sum_{i=0}^{\infty} p_i = 1$).

$$p_0 = (\sum_{i=0}^{m-1} \frac{\lambda^i}{i!\mu^i} + \sum_{i=m}^{\infty} \frac{\lambda^i}{m^{(i-m)} m!\mu^i})^{-1} \tag{5.6}$$

With the steady-state probabilities p_i, the expected queue size L_q can be calculated. L_q equals zero when the request number i is no more than VM number m, and is equal to $(i - m)$ when the request number i is more than the VM number n , and thus,

$$L_q = \sum_{i=m+1}^{\infty} (i - m) p_i \tag{5.7}$$

Based on Little's Formula [46] $L_q = \lambda T_q$, where $T_q = E(t_q)$ is the expected length of the waiting time in queue t_q.

$$T_q = \frac{L_q}{\lambda} = p_0 (\frac{(\lambda/\mu)^m}{m!(mu)(1 - (\frac{\lambda}{m\mu})^2)}) \tag{5.8}$$

With the expected waiting time T_q, the expected response time (average la-

tency) L can be calculated by

$$L(\lambda, \mu, m) = T_q + T_s \tag{5.9}$$

5.1.3 Optimisation Model

(1) Objective Function

Recall that the web application provider's greatest concern is to maximize profit (e.g. by minimizing cost) while providing high quality service (e.g. by minimizing latency) with lower SLA violation. However, these two factors are in conflict. As in this cloud-based web system with the cost demand on the number of allocated VMs, the number of VMs can be reduced to keep the cost as low as possible when there are insufficient VMs to process requests, but the waiting time in the queue will be too long. To solve this problem, the cost-latency trade-off optimisation is exploited as follows:

$$arg\min_{m,\lambda,\mu} \Gamma(\lambda, \mu, m) = \alpha * f(m) + (1 - \alpha) * L(\lambda, \mu, m) \tag{5.10}$$

where $\alpha \in [0, 1]$ reflects the importance ratio of cost and latency.

Due to the different scale of the number of VMs and latency, the latency can be normalised by

$$G = L/T \tag{5.11}$$

where T is the latency threshold, which is defined in SLAs.

To normalize the number of VMs, the equation below can be considered by

$$C' = F(m) = \frac{f(m) - f_{max}(m)}{f_{max}(m) - f_{min}(m)} \tag{5.12}$$

where $f(m)$, $f_{max}(m)$ and $f_{min}(m)$ refer to the VMs cost per time-unit, the least possible cost per time-unit and the maximum possible cost per time-unit, respectively. Then, the following objective function can be derived for the optimisation.

$$arg\min_{m,\lambda,\mu} \Gamma(\lambda, \mu, m) = \alpha * F(m) + (1 - \alpha) * G(\lambda, \mu, m) \tag{5.13}$$

Based on predicted requests in unit time t ,λ and μ are given, and the latency function in unit time t can be written as

$$L_t(\lambda, \mu, m) = L_t(m) \tag{5.14}$$

Considering the need to satisfy web application providers' cost constraints and SLAs violation in respect of latency, the final objective function of the cost-latency

trade-off in unit time t is obtained by the following:

$$\begin{aligned} &arg\min_{m}\Gamma_t(m) = \alpha * F_t(m) + (1-\alpha) * G_t(m) \\ &subject\ to\ \forall t:\ C_t(m) \leq C_t(max); Pr\{L_t(m) > T\} \leqslant K\% \end{aligned} \tag{5.15}$$

where the SLAs violations constraint K is usually defined as $K \in [2, 5]$ for web applications.

Assuming each server could tackle k requests within time T, the m VMs could tackle mk requests. This means that the SLA will be satisfied when the queue length is less than mk, because all requests could be tackled within time T. By referring to Figure 5.2, it demonstrates that only the previous mk steady-state can satisfy the SLA, and others will violate the SLA. So the equation of SLAs violations constraint can be written by

$$Pr\{L_t(m) \leqslant T\} = \sum_{i=0}^{mk} p_i > (1 - K\%) \tag{5.16}$$

(2) Solving the Optimisation Problem

To minimize the objective function, an optimal number of VMs m is expected to be calculated to obtain the cost-latency trade-off values, satisfying all constraints. Clearly, Equation 5.15 is a complex nonlinear function and hard to simplify by mathematical methods. Considering that the number of VMs the web application

Algorithm 5.1 Computing Optimal Number of VMs

input

λ - arrival rate, μ - process rate per VMs,
α - priority of cost, $\mathcal{K}$ - threshold of SLA violation

output

m - optimal number of VMs

```
1: minV = ∞
2: for (n = 1..N) do
3:    l_t = L(λ, μ, n); //Equation (6.9)
4:    l_t' = normalize(l_t); //Equation (5.11)
5:    n' = normalize(n); //Equation (5.12)
6:    newV = α * n' + (1 − α) * l_t'; //Equation (5.15)
7:    if ( (n satisfy constraint(K)) // Equation (6.20)
      && (newV < minV) ) then
8:       m = n;
9:       minV = newV;
10:   end if
11: end for
```

provider purchased is limited, an exhaustive search algorithm is exploited to calculate the Γ with different m, and to find the lowest $Cost$ and the related m, as shown in Algorithm 5.1.

5.2 Experimental Evaluation

In this section, the performance of the proposed method is evaluated. The experiment setup and datasets used in the experiments is first expressed, followed by the analysis and discussion of the evaluation results.

5.2.1 Experiment Setup and Datasets

The performance of the proposed method is evaluated by using three kinds of well-know real-world datasets as following: AOL [1] and Sogou [2] search log dataset, as well as another real-world dataset collected by the UTS (The University of Technology, Sydney) library, to evaluate the performance. Because most VMs instances in public clouds are charged hourly, the time-unit of re-allocation in this work is the hour-unit. Therefore, the length of time fragment is set as one hour, and the number of requests is aggregated for each hour.

The experiment is organised by steps as follows:

(1) investigate how the seasonal characters affect the selection of features for prediction modelling (Sub-section 5.2.2);

(2) evaluate the prediction model through three datasets (Sub-section 5.2.3);

(3) visualize the performance of the prediction model (Sub-section 5.2.4);

(4) evaluate the allocation performance for the given number of requests (Sub-section 5.2.5);

(5) compare the proposed method with other methods (Sub-section 5.2.6).

[1] http://www.infochimps.com/datasets/aol-search-data
[2] http://www.sogou.com/labs/dl/q-e.html

5.2.2 Features Selection Evaluation

To measure the seasonal characters, the difference between two time periods is compared. The number of requests in each hour is represented as a vector $<v_1, ..., v_i, ...v_{60}>$, where v_i is the requests volume within one minute. Each vector is considered as a distribution, and the Kullback-Leibler (KL) divergence is applied to measure the difference between two distribution probabilities.

$$D_{KL}(P||Q) = \sum_i \ln(\frac{P(i)}{Q(i)})P(i) \tag{5.17}$$

Because the KL divergence is a non-symmetric measure, a variant Symmetrizing KL (SKL) divergence [58] is utilised to evaluate as

$$SD_{KL}(P||Q) = \frac{D_{KL}(P||Q) + D_{KL}(Q||P)}{2} \tag{5.18}$$

By taking the hourly number of requests as an element, the requests volume in a day can be considered as a 24-length vector. Each vector is treated as a distribution, and can be calculated by the SKL divergence with another hourly vector. Similarly, the hourly vector can be extended to a weekly or monthly vector. Table 5.1 shows the average SKL divergences on hourly, daily and weekly vectors with three datasets.

For the SKL divergence, 1 represents the greatest distance and 0 describes

Table 5.1: Average SKL Divergence on Different Period Vectors

Dataset	Hour	Day	Week
AOL	0.0324	0.0283	0.0229
UTSlib	0.0667	0.0302	0.0577
Sogou	0.0054	0.0086	0.0110

the smallest distance. All the SKL divergences in Table 5.1 are small, which demonstrates that the three datasets have highly seasonal characters and the number of requests can be predicted by using the history data.

Before learning the prediction model, the key features need to be selected for the linear regression model. Table 5.2 represents the top 10 correlated features. The results in Table 5.2 show that the request volume in time-unit t is most correlated to that of the first previous unit-time $t-1$.

Table 5.2: Top 10 Correlated Lags

Period	Correlated lag (ordered by correlation descent)
AOL	1,2,3,4,5,145,144,146,6,143
UTSlib	1,2,169,25,168,24,170,145,26,3
Sogou	1,25,2,49,24,26,73,48,50,97

5.2.3 Evaluation Methods

Several common measurements are applied for the regression model, such as Root Mean Squared Error (RMSE), Relative Squared Error (RSE), Mean Absolute Error (MAE), Relative Absolute Error (RAE), and coefficient of determination

(R^2).

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - a_i)^2}{n}} \tag{5.19}$$

$$RSE = \frac{\sum_{i=1}^{n}(p_i - a_i)^2}{\sum_{i=1}^{n}(\bar{a_i} - a_i)^2} \tag{5.20}$$

$$MAE = \frac{\sum_{i=1}^{n}|p_i - a_i|}{n} \tag{5.21}$$

$$RAE = \frac{\sum_{i=1}^{n}|p_i - a_i|}{\sum_{i=1}^{n}|\bar{a_i} - a_i|} \tag{5.22}$$

$$R^2 = \frac{\sum_{i=1}^{n}(p_i - \bar{p_i})^2}{\sum_{i=1}^{n}(a_i - \bar{a_i})^2} \tag{5.23}$$

where a is the actual value, p is the predicted value.

10-fold cross validation are utilised as the evaluation method. Table 6.1 shows the performance of the regression model on three datasets.

Table 5.3: Performance of Regression Model

Data	Avg Req	RMSE	RSE	MAE	RAE	R^2
AOL	1.6*10^3	191	0.05	140	0.19	0.98
UTSlib	2.9*10^4	4582	0.10	3082	0.25	0.96
Sogou	7.6*10^4	5617	0.02	3555	0.10	0.99

5.2.4 Prediction Model Evaluation

In a practical application, a padding is added to the predicted value as the cap (U) of prediction.

$$U = (1 + padding) * prediction \tag{5.24}$$

The prediction accuracy is evaluated by utilising the confidence interval $Pr(x < U)$, which represents the probability that real demands (x) are less than the cap (U) of the prediction. To select a good padding value, the relationship between the padding value and the confidence interval is measured, as shown in Table 5.4.

Table 5.4: The Confidence Interval with Different Paddings

Padding (%)	Confidence Interval Pr{x⩽U}		
	AOL (%)	UTSlib (%)	Sogou (%)
5	69.27	68.63	86.84
10	82.96	81.16	95.44
15	91.46	89.67	98.60
20	95.49	94.43	99.30
25	97.57	97.10	99.30
30	98.50	98.17	99.65
35	99.13	99.05	100
40	99.42	99.46	100
45	99.56	99.64	100
50	99.76	99.79	100

Figure 5.3 shows that the proposed method achieves good prediction on both number of requests and resource demands, and that the padding value can be dynamically adjusted well in each time interval.

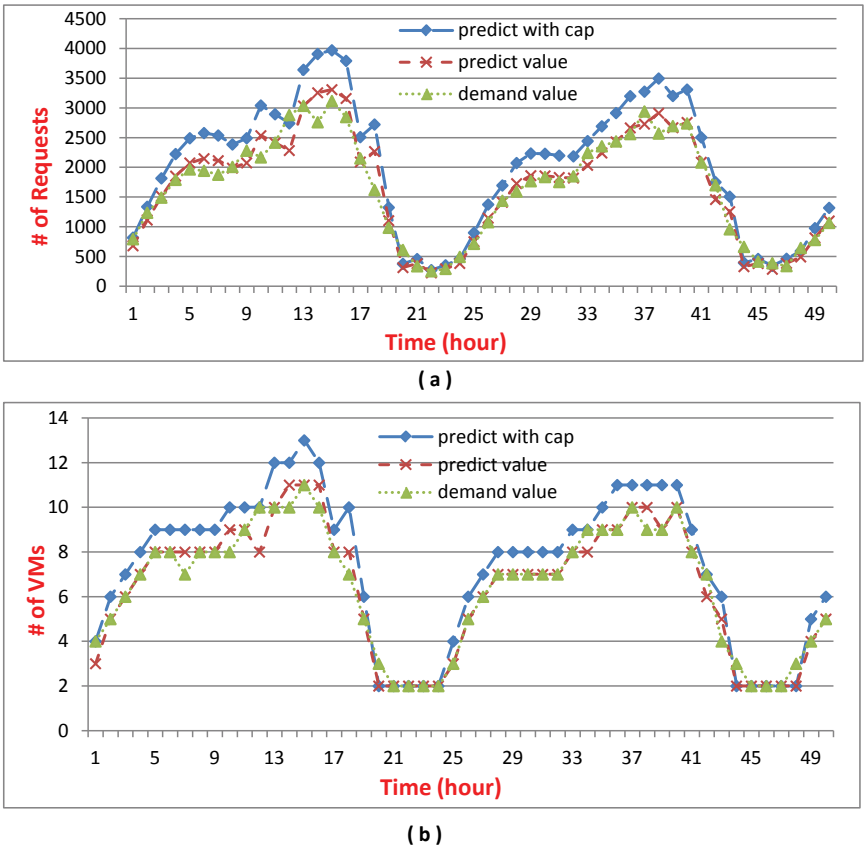


Figure 5.3: Prediction and Allocation with a Dynamic Cap

5.2.5 Allocation Evaluation

This allocation method is related to the arrival rate λ (per minute), process rate μ (per minute), maximal process time T (s), SLA violation ratio threshold K, and cost priority α. $r = \lambda/\mu$ is defined as the minimal required number of VMs, and consider m as the optimal number of VMs allocated by the proposed method.

With the given $\mu = 10$, $T = 60$, $K = 2\%$ and $\alpha = 0.8$, the λ is changed from 10 to 300. Figure 5.4 (a) shows that a bigger padding $(m - r)$ should be allocated when the number of requests increases. Meanwhile, Figure 5.4 (b) shows that the

relative ratio between m and r (i.e. m/r) decreases to be close to 1, which means the method achieves good cost-effectiveness when the number of requests rises.

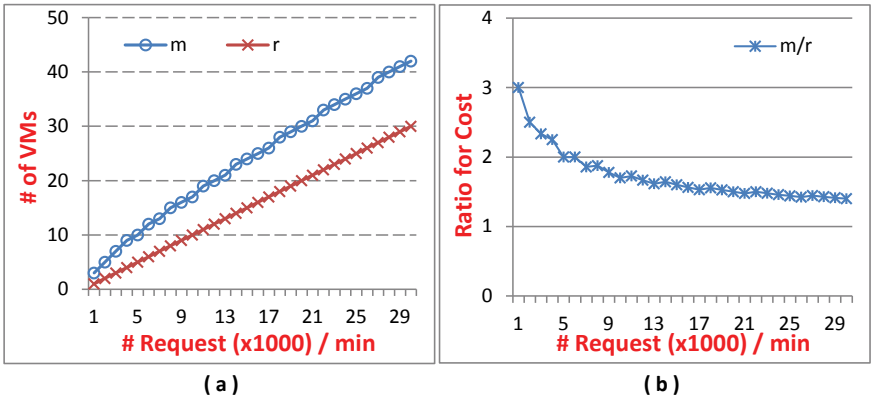


Figure 5.4: Allocation with Queuing Theory.

5.2.6 Performance Evaluation for a Web Application

The proposed method is implemented on Amazon AWS with a Web application. This method can rent or lease VM instances automatically from Amazon EC2. To simplify the problem, the experiment only considers the cost of VMs with same type of instances. The frequency of requests is simulated based on the real datasets, and the process time of requests obeys Poisson distribution ($\frac{1}{\mu} = 6$ seconds).

The proposed method (QT) is compared with another three methods: PEAK, PEAK($\times 3/4$) and Cap($\times 2$). For Peak method, the number of VMs is always allocated based on the peak value, while PEAK($\times 3/4$) is an method to reduce cost by allocating the number of VMs as 3/4 of peak value. For the CAP($\times 2$)

method, the resource cap is set as two times of the minimal number of VMs that can satisfy the predicted the number of requests by considering all requests that arrived with an average rate.

As Figure 5.5 and Table 5.5 shows, the proposed method allocates less resources, while achieving better performance compared to the PEAK(×3/4) and Cap(×2). Compared to the PEAK, the proposed method reduces much less numbers of VMs, although with slightly higher SLA violation rate.

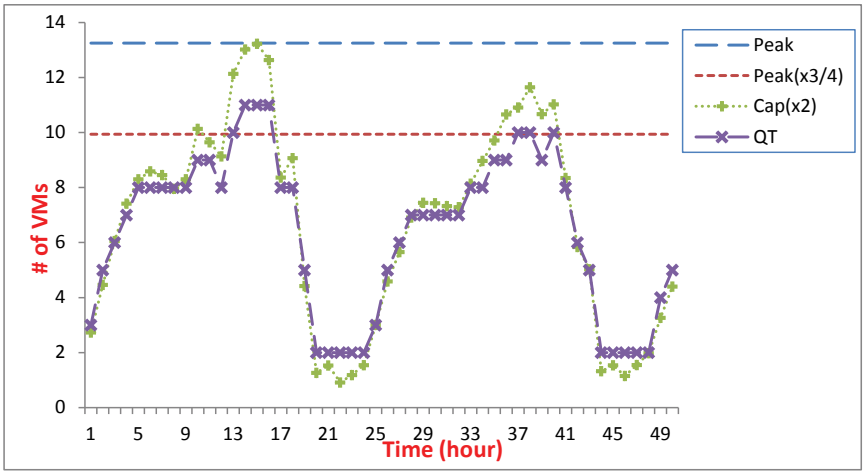


Figure 5.5: Allocation Comparison for Different Methods

5.3 Summary

In this chapter, an innovative auto-scaling method from IaaS level is proposed , an automatic decision suppport testbed is developed for cloud tenants (e.g. web application providers), who have time-varying resource (e.g. VMs) demand, to

Table 5.5: Performance Comparison for Different Methods

Dataset	Method	# Req /h	# VMs /h (avg)	Violate (%)	Avg Tq /h (s)
AOL	PEAK	1916	13.25	0.03	6.15
	PEAK (×3/4)	1916	9.94	0.63	16.15
	CAP(×2)	1916	7.33	0.57	15.61
	QT	1916	**7.21**	**0.18**	**9.96**
UTSlib	PEAK	2165	21.00	0.02	7.46
	PEAK (×3/4)	2165	15.75	0.45	19.15
	CAP(×2)	2165	8.25	0.24	13.73
	QT	2165	**7.75**	**0.20**	**11.87**
Sogou	PEAK	2954	25.67	0.06	8.32
	PEAK (×3/4)	2954	19.23	1.02	26.15
	CAP(×2)	2954	10.67	0.76	18.35
	QT	2954	**9.70**	**0.54**	**13.54**

auto-scale optimal VM consumption. The method consists of four key components:

(1) a dynamic prediction model is learned to forecast the workload in the next unit time;

(2) an adaptive error feedback learning method to add an increment to the prediction value is proposed to solve the problem of underestimation and overestimation, which is one of the most important issues in cloud resource prediction;

(3) an optimisation method is presented to obtain the minimal cost with low SLA violation to maximize cloud tenants' profits;

(4) a reactive method is proposed to add an extra number of VMs to the predicted optimal value, in order to deal with the unpredictable workload burst.

Based on the optimal number of VMs that need to be allocated, the system automatically scale up or down the capability of cloud platform. The experimental results demonstrate that the proposed method can balance the cost and desired latency. Compared to other methods, the proposed method presents a superior price-performance ratio across three real-world datasets. This research will potentially accelerate the migration of web applications to cloud systems.

Chapter 6

ACWS Optimisation on IaaS for Bandwidth

Bandwidth-intensive cloud online application providers, such as Video-on-Demand (VoD) providers, tend to lease computation and network resources from commercialised public cloud service providers like Amazon, instead of investing in their own server clusters. For example, Netflix, an online VoD provider, moved its streaming servers, encoding software, data storage and other customer-oriented APIs to Amazon Web Services (AWS) in 2010 [2]. One of the main interests in applying cloud services for these bandwidth-intensive cloud application providers is that they can scale resources automatically according to demand to achieve cost-efficiency.

Unlike storage-intensive and computation-intensive applications, these bandwidth-

intensive applications must be continuously served by the network-bound bandwidth, because a stringent data transfer rate must be maintained to allow end users (e.g.VoD users) to watch online videos smoothly during each full service period. The outgoing bandwidth in current cloud service provision is limited and is shared by multiple tenants; the network bandwidth will thus be a major performance bottleneck when a large number of end user requests are received at the same time. To achieve maximum profits for bandwidth-intensive application providers, a dynamic *bandwidth reservation* is urgently required in which network usage is charged by bandwidth size (i.e. pay-by-bandwidth) rather than by the total amount of bytes transferred (i.e. pay-by-bytes).

No bandwidth capacity guarantees are provided in the current cloud services, but automated bandwidth reservation is a near-term possibility, since the emerging new network virtualisation technology, *Software Defined Network* (SDN), provides a feasible solution. In the SDN-based cloud network, the management functionality, which is performed by software and executed by an external controller in the cloud, and the forwarding functionality, which is implemented by switches and routed in hardware, are separated so that the optimum routing calculation can be executed without physical switches or routers being modified and the virtualised network bandwidth can be performed in software [34], [14], [75]. Cloud network tenants are able to specify guaranteed bandwidth requirements for their applications, which are hosted in the cloud. When dynamic and large-scale requests

occur for bandwidth-intensive applications, a cost-effective network consumption method in the SDN-based cloud network is particularly required to achieve high quality performance.

A novel auto-scaling method for cloud bandwidth consumption in the new scenario of SDN-based cloud network is proposed in this chapter. This will enable bandwidth-intensive application providers to incur minimal cost while at the same time avoiding SLA violations, without the need for manual intervention. The proposed method is based on a scheme in which the dynamic network consumption is charged by bandwidth-usage in a short period (i.e. pay-by-bandwidth in time-unit), in contrast to the traditional network consumption in which a long-term fixed bandwidth fee is paid, for example, for a plan of 1 Gbps per month, or the fee is charged according to the total number of bytes transferred.

There are two main challenges to be addressed to achieve this goal: *Firstly*, how should the dynamic bandwidth demand be calculated in each time-unit? *Secondly*, how should the optimal bandwidth be reserved to minimise the cost but control SLA violations?

To address these two key challenges, a VoD application request requires a virtual tunnel with fixed bandwidth for data transfer in a SDN-based cloud network. The bandwidth provision for each request is continuous and is on a *first come, first serve* (i.e. one request finishes, a new request starts). Thus the bandwidth demand is computed by calculating the number of requests and their correspond-

ing fixed bandwidth in each time-unit. In this work, machine learning techniques are leveraged to forecast the number of requests in the next time-unit so that bandwidth usage in a future time-unit, by learning from the historical data of requests. The service time for consideration of each request of an application will vary, (e.g. one video has a different streaming transfer time from another in VoD applications), thus a $M/G/c$ model in queueing theory is applied to analyse the performance of reserved bandwidth. Taking the cloud bandwidth consumption price model into account, an optimisation algorithm is developed to minimise the cost while satisfying the performance constraint conditions defined in the SLA.

Our scheme scales the cloud resource up or down by time-unit allocation based on predicted optimal resource demands. Web application providers can specify their budgetary constraints and SLAs in respect of latency for their applications. In each time-unit, web application providers can be notified of the total cost, SLA violations and re-allocation state (i.e. scaling up or down) by using the optimal resource auto-scaling scheme. Note that in practical applications, an unpredictable burst of requests may occur, similar to the Animoto experience. To tackle this unpredictable scenario, this scheme monitors the waiting queue of requests to be processed in real-time. Once the length of the queue is bigger than a certain threshold, the scheme can dynamically excess bandwidth for processing the exceed number of requests. Figure 6.1 illustrates the overview execution paradigm facilitated by the scheme.

As shown in Figure 6.1, the main steps of the proposed scheme for system workflow are outlined as follows.

- Service Component: End-users request services from the Web servers of the service provider, and the Web server allocates the cloud data server and network resource for the request. With the information received from the Web servers, the end-user build a network channel with the data server and starts to use the service;

- Management Component: The auto-scaling controller monitors the web server in real time to collect trace data, and makes hourly predictions for the request scale. The auto-scaling controller then runs an optimisation algorithm to obtain the optimal amount of bandwidth. With the desired optimal number of resources, the auto-scaling controller communicates with the SDN controller, updates the configuration of the network and notifies the communication results to the Web server.

6.1 Methodology

In this section, the overview of the proposed method is presented and the modelling of the system is described.

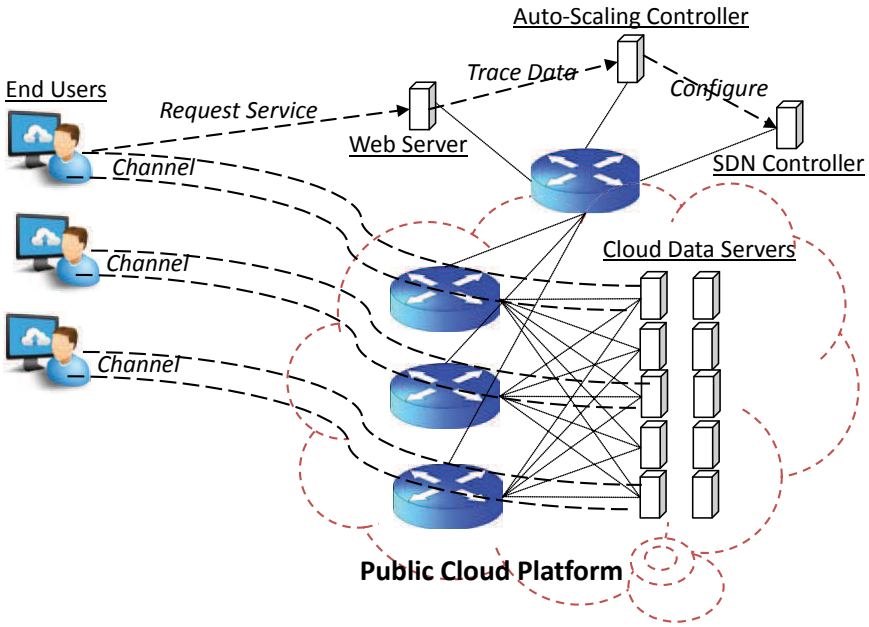


Figure 6.1: Execution Paradigm of SDN-based Bandwidth Auto-scaling

6.1.1 Dynamic Prediction Model

Assume a VoD service provider periodically requires or occupies bandwidth from a SDN-based cloud. A dynamic prediction model best fits the time-varying demand. Real-time prediction is too frequent and unstable for resource requests. An hour is a reasonable period for resource requests, therefore every hour, our method will predict the demand of the next hour, and request the corresponding resources.

There are numerous ways to predict the traffic of an online application, such as the autoregression (AR) model, moving average, or machine learning. In this chapter, a machine learning-based method is chosen due to its powerful ability to utilise the historical data. In particular, neural network is applied as the

prediction algorithm.

A time series $\{R_1, R_2, ...R_i, ...R_n\}$ is defined to represent the request volume of an online application over a long period. Each R_i represent the request volume in the period whose index is i. There is a total of n periods in the time series. The prediction objective is to predict the value R_{n+1}.

In time series analysis, a basic assumption is that the value of the next period will be related to previous periods. There are two challenges for the analysis: one is which period will impact the prediction more, while the other is how to represent the relationship between the prediction and the historical data. Use random variable X_n to represent the nth period's value. The prediction model can be represented by

$$X_n = F(X_{n-1}, ...X_{n-i}, ...) \tag{6.1}$$

There are too many factors on the right side of Equation 6.1. To avoid an overfitting problem, only the important factors which have a high correlation with X_n are selected. After the factor (feature) selection process, a new equation can be obtained:

$$X_n = F(X_{n-k}, ...X_{n-p}, ...) \tag{6.2}$$

Based on the equation above, a new vector $M_n = \{X_n, X_{n-k}, ...X_{n-p}, ...\}$, and a new matrix $M = \{M_n; ...M_i; ...\}^T$ can be constructed.

The relationship can be linear or non-linear. For a linear relation assumption, the AR model, linear regression analysis can be used to obtain the prediction. Considering the non-linear method represents more complex relationship than the linear method and the non-linear method is used to predict the value of the next period R_{n+1}. Regression neural network [94] can be applied as the non-linear prediction method due to its ability to represent more complex relationships with deep architecture.

6.1.2 VN Consumption Model

In this service model, a fixed bandwidth for each request is allocated so that users can utilise the network resources to accomplish their data transfer task, such as on-line video watching, or big data transfer. Assume the arrivals of requests are Poisson with rate λ and that resources are allocated according to the FCFS (First-come, First-served) rule.

For a video-watching task, the service time is decided by the length of the video; for a data transfer task, the service time is decided by the size of data because the bandwidth is fixed for each request. Therefore, assume that the video length or data size follow the general distribution with mean value μ; this means that the service time of each request follows the general distribution as well. Moreover, the network resource can be divided into multiple small fixed band-

width channels (representing the number of channels with random variable c) and each of these can provide service to one request. Based on these considerations, the system can be modelled as a M/G/c queueing system.

(1) Static-transition Probability

The moments of task request arrivals are selected as Markov points. Two successive task request arrivals and the task departures that may occur between them are shown schematically in Figure 6.2. Note that the number of departures may be any integer between 0 and ∞.

Let A_n and A_{n+1} indicate the moment of the n th and $(n+1)$ th arrivals to the system, respectively, while q_n and q_{n+1} indicate the number of tasks found in the system immediately before these arrivals. If v_{n+1} indicates the number of tasks which depart the system between A_n and A_{n+1}, then $q_{n+1} = q_n - v_{n+1} + 1$.

(2)Dynamic Service-time Probability

Due to the ergodicity of the system, an equilibrium probability distribution exists for the number of tasks present at the arrival instants. Let $\pi_k = lim_{n \longrightarrow +\infty} Prob[q_n = k], 0 \leqslant k \leqslant m + r$, denotes the probability of having k tasks in the system immediately before a new task arrival.

(3) M/G/c Queueing System

In respect of the M/G/c queueing system, Tims et al. [100] believe it is "not likely that computationally tractable methods can be developed to compute the exact numerical values of the steady-state probability in the M/G/k queue".

Therefore, the service waiting time can only be approximated. The most popular approximation method is based on the equation below:

$$E[W^{M/G/c}] = \frac{1 + C_B^2}{2} * E[W^{M/M/c}] \tag{6.3}$$

where $C_B^2 = Var[S]/E^2[S]$ is the squared coefficient of the variation of service distribution, and $E[S] = \frac{1}{\mu}$ is the average service-time for all requests. The inference of $E[W^{M/M/c}]$ will be discussed in Section 6.1.2.

(4) M/M/c Queueing System

All settings for the $M/M/c$ queueing system are the same as $M/G/c$ except that the service time distribution is Poisson distribution rather than General distribution. The $M/M/c$ queueing system is a special birth-death process which can be modelled as a Markov chain. As shown in Figure 6.2, the steady=state probabilities p_n can be obtained by equation below:

There are c channels in the system and one channel for each web request is allocated at every instant time-point with a constant rate, so that the "birth" rate is $\lambda_i = \lambda$ for all n. On the other hand, the rate of request completions (or "deaths") depends on the number of channels in the system. If there are c or more requests in the system, then all c servers must be busy at the instant time-point. Since each channel processes requests with rate μ, the combined process-completion rate for the system is $(c\mu)$. When there are fewer than c

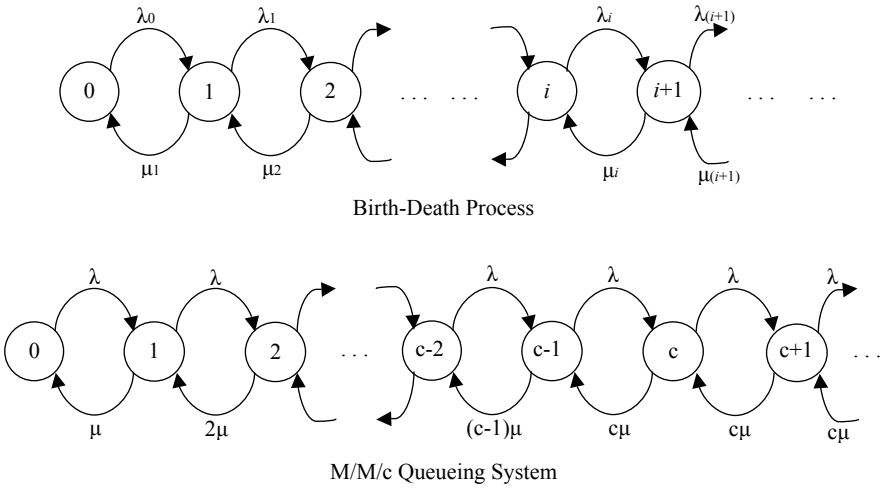


Figure 6.2: Transition Rate for the Web Requests Process on Channels

customers in the system, e.g. $i < c$, only i of the c channels are busy and the combined service-completion rate for the system is $(i\mu)$. Hence μ_i may be written as

$$\mu_i = \begin{cases} i\mu & 1 \leqslant i < c, \\ c\mu & i \geqslant c \end{cases} \tag{6.4}$$

Based on the Markov chain in Figure 6.2, the steady-state probabilities p_i can be obtained:

$$p_i = p_0 \prod_{k=1}^{i} \frac{\lambda_{k-1}}{\mu_k} = \begin{cases} \frac{\lambda^i}{i!\mu^i} p_0 & 1 \leqslant i < c, \\ \frac{\lambda^i}{c^{(n-c)} c!\mu^i} p_0 & i \geqslant c \end{cases} \tag{6.5}$$

To obtain the value of p_0, the condition that the probabilities must sum to 1

$(\sum_{i=0}^{\infty} p_i = 1)$ is applied.

$$p_0 = (\sum_{i=0}^{c-1} \frac{\lambda^i}{i!\mu^i} + \sum_{i=c}^{\infty} \frac{\lambda^i}{c^{(i-c)}c!\mu^i})^{-1} \tag{6.6}$$

With the steady-state probabilities p_i, the expected queue size L_q can be evaluated. L_q equals zero when the request number i is no more than channel number c, and is equal to $(i-c)$ when the request number i is more than the VM number n , and thus,

$$L_q = \sum_{i=c+1}^{\infty} (i-c)p_i \tag{6.7}$$

Based on Little's Formula [46] $L_q = \lambda W_q$, where $W_q = E(t_q)$ is the expected length of the waiting time in queue t_q.

$$W_q = \frac{L_q}{\lambda} = p_0(\frac{(\lambda/\mu)^c}{c!(mu)(1-\frac{\lambda}{c\mu})^2}) \tag{6.8}$$

With the expected waiting time T_q, the expected response time (average latency) $W^{M/M/c}$ can be calculated by the equation below:

$$E[W^{M/M/c}] = W_q + E[S^{M/M/c}] = W_q + \frac{1}{\mu} \tag{6.9}$$

(5)Performance Measurement For any request in the proposed $M/G/c$ queueing system, only the wait-time-in-queue W_q is concerned rather than the total waiting time W which is composed of the wait-time-in-queue W_q and service-

time S. The W_q is calculated by the equation below:

$$\begin{aligned} E[W_q^{M/G/c}] &= E[W^{M/G/c}] - E[S^{M/G/c}] \\ &= \frac{1+C_B^2}{2} * E[W^{M/M/c}] - E[S^{M/G/c}] \\ &= F(\lambda, \mu, Var(S), c) \end{aligned} \tag{6.10}$$

6.1.3 Profit Maximisation of VN Consumption

(1) Probability of SLA Violence

The SLA of wait-time-in-queue is usually defined within one minute while the service time can be tens of minutes. If a request arrives at a busy situation (i.e. all servers are busy), the request's wait-time-in-queue will exceed the SLA definition. Considering the Markov Chain, the preceding c states are considered as idle states while others are treated as busy states. All busy states can result in SLA violence. Therefore, the probability of SLA violence can be defined as the equation below.

$$P_{SLA} = Pr(X_n > c) = 1 - Pr(x_n \leq c) = 1 - \sum_{i=0}^{c} p_i \tag{6.11}$$

where p_i is the steady-state probability of i in the system at an arbitrary point of time.

Let π_n represent the probability of n in the system at a departure point (a point of time just slightly after a customer has completed service) after steady state is reached. Authors in [47] proved that the π_n, the steady-state probability of n in the system at a departure time, equals to p_n, the steady-state probability of n at an arbitrary point in time. Therefore, the $\pi = \{\pi_n\}$ can be used to represent the steady-state probability vector. Moreover, we set $X_n = X(t_n)$ be the number of customers left in the system immediately after the departure of the customer at time t_n.

The Markov-Chain of the M/G/c queueing system can be described as shown in the Figure 6.3. Each steady-state is able to transfer to all other back steady-

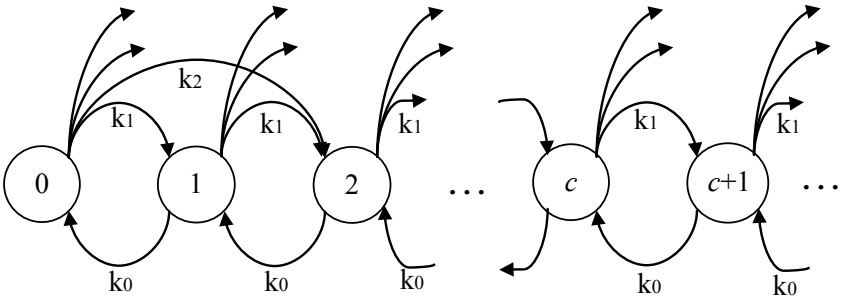


Figure 6.3: Rate Transition Rate for the Web Requests Process on Channels

states. Because there are multiple arrivals during one service time. Moreover, each state only can transfer to the previous one steady-state. Because each state represents the departure time of one customer. Accordingly, we define the tran-

sition matrix P_{tran} as

$$P_{tran} = \{p_{ij}\} \begin{pmatrix} k_0 & k_1 & k_2 & k_3 & \cdots \\ k_0 & k_1 & k_2 & k_3 & \cdots \\ 0 & k_0 & k_1 & k_2 & \cdots \\ 0 & 0 & k_0 & k_1 & \cdots \\ 0 & 0 & 0 & k_0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{6.12}$$

where k_i =Pr$\{i$ arrivals during a service time$\}$.

Based on the Markov Chain's steady state characters $\pi P = \pi$, the equation for π is calculated as

$$\pi_i = \pi_0 k_i + \sum_{j=1}^{i+1} \pi_j k_{i-j+1} \tag{6.13}$$

where $\pi_0 = 1 - \rho$.

The Equation (6.13) contains two power series. Two generating functions correspond to the power series π_i and k_i are defined as

$$\Pi(z) = \sum_{i=0}^{\infty} \pi_i z^i \tag{6.14}$$

$$K(z) = \sum_{i=0}^{\infty} k_i z^i \tag{6.15}$$

Finally, we can get an equation containing only one power series k_i in one variable z.

$$\Pi(z) = \frac{(1-\rho)(1-z)K(z)}{K(z)-z} \tag{6.16}$$

With Equation (6.16) and the given distribution of the service time, the probability vector π [47] can be calculated. Thus the probability of SLA violence can be represented as the equation below.

$$P_{SLA} = G(\lambda, \mu, D_s, P_{tran}, c) \tag{6.17}$$

where D_s denotes the parameters for the distribution of service time.

(2) Objective Function

To minimise the cost of network consumption while considering satisfying the SLA, the optimisation problem is represented as the equations below:

$$\arg\min_{c} M_{cost} = H(c) \tag{6.18}$$

subject to:

$$E[W_q^{M/G/c}] \leqslant W_{SLA} \tag{6.19}$$

and

$$P_{SLA} < d\% \tag{6.20}$$

where, $H(c)$ is the cost function of the network resource, c is the number of channels, and W_{SLA} is a constant representing the maximal waiting-time in queue defined in SLA and d is the threshold of the probability of the SLA violence.

(3) Search-based Optimization Algorithm

The proposed objective function and conditional function in Equations (6.18), (6.19) and (6.20) are composed of complex mathematic expression, and it is hard to find the optimal result by mathematic tools. To solve the optimization problem, we need to search for the optimal value from the solution space. Consider the reality limitation, our optimization problem's solution space is a finite integer interregional. Particularly, the result of optimal c is a finite integer, and we can an utilize greedy algorithm to find the optimal solution from the finite integer solution space. The pesudo-code is listed in the Algorithm 6.1.

6.2 Experimental Evaluation

In this section, a SDN-based cloud system is simulated, and the performance of the proposed scheme is evaluated. The experiment setup and datasets used in the experiments are first described, followed by the analysis and discussion of the evaluation results.

Algorithm 6.1 Optimal Search-based Algorithm for Cloud Bandwidth Computing

input

N - maximal number of resource rate, μ - process rate per VMs,
α - priority of cost, N - maximal number of resource,
Other related parameters for Queueing theory

output

m - optimal bandwidth

1: $minV = \infty$
2: **for** (c = 1..N) **do**
3: $W_q = F(\lambda, \mu, Var(S), c)$ // Equation (6.10)
4: $P_{SLA} = G(\lambda, \mu, D_s, P_{tran}, c)$ // Equation (6.17)
5: $newV = H(c)$; //Equation (6.18)
6: **if** ((W_q satisfy constraint 1)) // Equation (6.19)
&& (P_{SLA} satisfy constraint 2) // Equation (6.20)
&& ($newV < minV$)) **then**
7: $m = c$;
8: $minV = newV$;
9: **end if**
10: **end for**

6.2.1 Experiment Setup and Datasets

The request arrival model is simulated from three real-world trace datasets: AOL [1] and Sogou [2] search log datasets, and another real-world dataset collected by the UTS (The University of Technology, Sydney) library. To simplify the representation, these AOL, Sogou and UTS-Library datasets are named DS-1, DS-2 and DS-3 respectively. To simulate the service time, a Gaussian distribution (u=10 minutes and $\sigma^2 = 5.0$) is applied to generate the number of arrived requests in each time-unit. Because most resource instances in public clouds are

[1] http://www.infochimps.com/datasets/aol-search-data
[2] http://www.sogou.com/labs/dl/q-e.html

charged hourly, the time-unit of re-allocation in this work is the hour-unit. Therefore, the length of time fragment is set as one hour, and the number of requests is aggregated for each hour.

The experiment is organised in steps as follows:

1. evaluate the prediction model through three datasets (Sub-section 6.2.2);
2. evaluate the proposed schema for Network Auto-scaling (Sub-section 6.2.3).
3. investigate how performance is impacted by various parameters.(Sub-section 6.2.4).

6.2.2 Evaluation of Neural Network-based Prediction Model

The number of requests in the current time-unit is treated as the prediction target, and the number of requests in the previous several time-units are treated as training features. A training matrix can be generated in which a row represents each time-unit and a column is the number of requests in the corresponding time-unit. To avoid the over-fitting problem resulting from numerous features, a feature selection method is implemented based on the auto-correlation of time series data. With the selected features and targeted values, a neural network regression model is trained. This feature selection based neural network regression model is evaluated on common criteria compared with benchmark algorithms. The chosen benchmark algorithms and our proposed method are shown below:

- AR method (benchmark): Basic Autoregression (AR) [18] method with previous 100 fragments of time.

- FAR method (benchmark): Basic AR model with selected features. Particluarly, we treat the workload value of previous time fragment as a feature and then select a few most related features as the input of AR model.

- FNN method: Our proposed feature selection-based Neural Network prediction method. Particularly, we treat the workload of previous time fragment as a set of features, and then apply Neura Network to do the prediction of current workload;

Several common measurements for the regression model are chosen, such as Root Mean Squared Error (RMSE), Relative Squared Error (RSE), Mean Absolute Error (MAE), Relative Absolute Error (RAE), and coefficient of determination (R^2).

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - a_i)^2}{n}} \tag{6.21}$$

$$RSE = \frac{\sum_{i=1}^{n}(p_i - a_i)^2}{\sum_{i=1}^{n}(\bar{a_i} - a_i)^2} \tag{6.22}$$

$$MAE = \frac{\sum_{i=1}^{n}|p_i - a_i|}{n} \tag{6.23}$$

$$RAE = \frac{\sum_{i=1}^{n} |p_i - a_i|}{\sum_{i=1}^{n} |\bar{a}_i - a_i|} \tag{6.24}$$

$$R^2 = \frac{\sum_{i=1}^{n} (p_i - \bar{p}_i)^2}{\sum_{i=1}^{n} (a_i - \bar{a}_i)^2} \tag{6.25}$$

where a is the actual value, p is the predicted value.

Table 6.1 shows the performance of the regression method on three datasets. The results in Table 6.1 demonstrate that the FNN method has better performance than the AR and FAR baseline methods.

Table 6.1: Performance of Regression Method

DataSet	Avg Req	Model	RMSE	RSE	MAE	RAE	R^2
		AR	221	0.13	172	0.29	0.87
DS-1	$1.6*10^3$	FAR	203	0.07	156	0.23	0.93
		FNN	190	0.05	138	0.18	0.99
		AR	6103	0.11	4021	0.21	0.91
DS-2	$7.6*10^4$	FAR	5617	0.06	3647	0.14	0.96
		FNN	5603	0.02	3419	0.09	0.99
		AR	5275	0.18	3311	0.33	0.86
DS-3	$2.9*10^4$	FAR	4600	0.13	3082	0.25	0.93
		FNN	4517	0.09	3039	0.24	0.97

6.2.3 Evaluation of Network Consumption Auto-scaling

A simulator for the M/G/c queueing system is built to simulate the cloud network resource allocation and consumption procedure. In the experiment, we sequen-

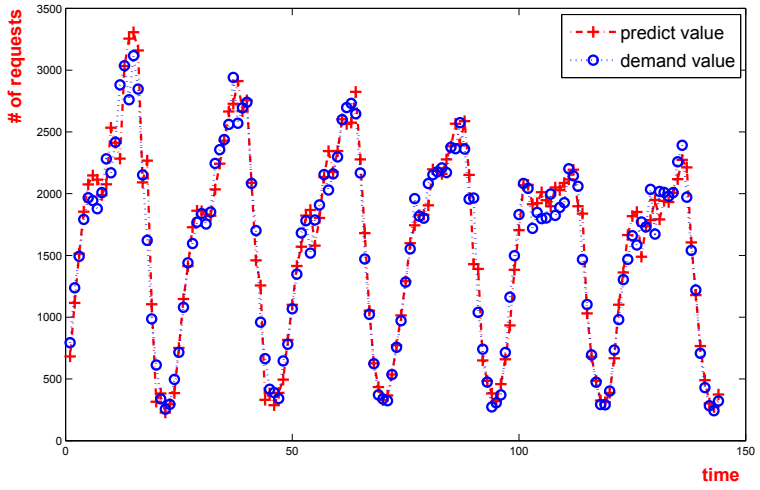


Figure 6.4: The Prediction Curve for DS-3

tially input the request and corresponding service-time based on arrival time, and the simulator allocates the servers (channels) for each request following the FCFS rule. The queueing system automatically adjusts the number of servers hourly based on the proposed prediction model's result and optimisation algorithm's outcomes. Three methods are compared on the simulator:

- PEAK: allocate resources based on peak hour requirements.
- Mean($\times K$): allocate K times the mean value with the required resource. K is an empirical coefficient, and is set as 2 in the experiment.
- Dynamic Auto-Scaling (DAS): allocate resources based on Queueing Theory and the optimisation algorithm proposed in this chapter.

For the experiments, four performance indicators are measured for the system.

- N_c: the average number of allocated network channels;
- W_q: the average waiting time in queue for web requests;
- $R_{violate}$: the ratio of SLA-violation for all time-units;
- Utilisation: the time utilisation of network resources;

The experiment result is shown in Table 6.2.

Table 6.2: Performance of Regression Model

Model	Data	$\bar{N_c}$	$\bar{W_q}$(Sec)	$R_{violate}$	Utilisation
DS-1	PEAK	$3.2*10^3$	3	0.01	0.45
	Mean($\times$2)	$2.1*10^3$	7	0.05	0.60
	DAS	$0.6*10^3$	12	0.09	0.85
DS-2	PEAK	$13.3*10^4$	3	0.01	0.41
	Mean($\times$2)	$8.7*10^4$	6	0.06	0.55
	DAS	$1.7*10^4$	9	0.09	0.79
DS-3	PEAK	$6.1*10^4$	2	0.01	0.43
	Mean($\times$2)	$4.2*10^4$	6	0.07	0.57
	DAS	$0.8*10^4$	11	0.13	0.82

As Table 6.2 shows, the proposed DAS method uses fewer resources to achieve almost the same level wait-time in queue as the PEAK and Mean methods. The Mean($\times$2) method has the highest violation ratio because the most peak hour visits exceed twice the number of the mean value. The DAS method has the highest utilisation of network resources while satisfying the SLA constraint. In summary, DAS is a cost-effective method for Cloud network resource auto-scaling.

6.2.4 Performance Study on Various Parameters

In the proposed optimisation algorithm, different parameters will impact the performance. Our allocation method is related to the arrival rate λ, and the service time S which is decided by Gaussian distribution with parameters μ' and σ^2. The value of these three parameters is adjusted on the DS-3 dataset, and the changes to c are observed- the number to allocated resources (network channels). As shown in Figure 6.5, the required resource is increased while the number of requests per time-unit is raised. In the left figure, the three curves represent different σ^2 values, and the system requiring more resources shows a bigger variation. In the right figure, with the same σ^2 values, the three curves demonstrate the system require more resources with a smaller μ' value, and the system experiences in a big increase of resource demand if the value of μ' equals to one.

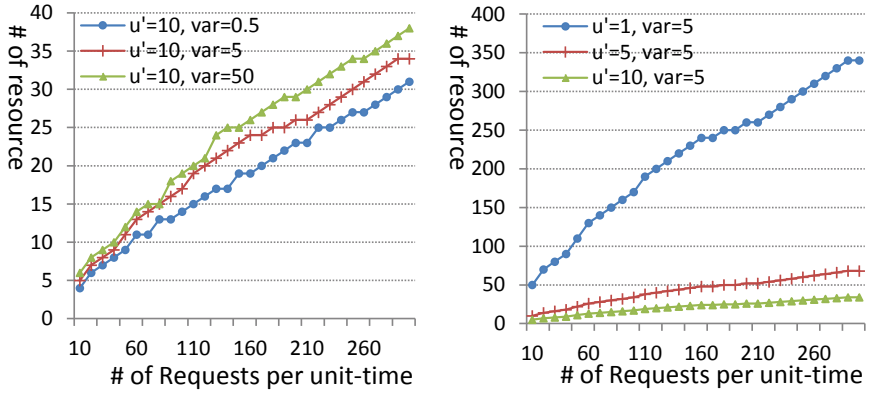


Figure 6.5: Performance Impact on Various Parameters with DS-3

6.3 Summary

In this chapter, a novel optimisation method of auto-scaling for bandwidth-intensive cloud-based web services is developed at the IaaS level. With large-scale outgoing bandwidth requirements for data transfer, the scale of bandwidth is updated in each bandwidth-charged time-unit according to the predicted workload (the number of requests for data transfer). Learning from the historical request data, a neural network tool is proposed to predict the expected future demand requests. The prediction is operated at each bandwidth-charge time-unit and the prediction model is also updated correspondingly. Leveraging the predicted values, a $M/G/c$ queueing model is applied to quantify the relationship between the resource volume and the QoS of the SLA. The optimal volume of resources is assigned hourly depending on the predicted workload and resource allocation is reactively appended in real time for unpredicted request bursts.

The proposed method is evaluated through extensive simulation based on three real-world trace datasets. The experiment results show that the method achieves cloud bandwidth auto-scaling with a low rate of prediction errors, as well as optimal consumption with scalar cost-latency trade-off and low SLA violations.

Chapter 7

Conclusions and Future Research

7.1 Conclusions

This research aims to solve the optimised resource allocation problem for auto-scaling cloud-based web services. With the proposed methods, cloud-based web service providers can provide high quality auto-scaling web services to users at minimal cost. Moreover, the saved energy consumption benefits the global environment. In particular, this study solves various optimisation problems at three levels of cloud computing: SaaS, PaaS and IaaS.

At the SaaS level, the parallelised programming model of cloud-based web services is developed. This parallelized model achieves auto-scaling web service provision with high performance. The optimisation problem of cloud-based time-consuming web services is also solved by considering the speedup curve and SLAs. Moreover, the program can achieve auto-scaling indeed due to the computation complexity of programs are highly depended on the number of cloud computation

resource.

At the PaaS level, a self-adaptive platform for cloud-based web services is proposed to achieve optimised auto-scaling. To support multiple kinds of web services, the platform auto-configures web services for allocation on cloud servers. The proposed method obtains optimal outcomes by leveraging the number of allocated cloud servers and the configuration of services on servers while satisfying the QoS of SLAs. Moreover, the platform configures various kinds of web severces automatically and assign web requrests to the appropriate computation node and thus to achieve the auto-scaling for cloud-based web services.

At the IaaS level, two novel optimisation methods for two different kinds of infrastructure resource allocation as VMs and VNs are proposed to achieve auto-scaling for two kinds of cloud-based web service: computation-intensive web service and bandwidth-intensive web service.

7.2 Main Contributions

This research develops a deep insight into cloud resource optimisation for auto-scaling cloud-based web services from the three business models of cloud computing of SaaS, PaaS and IaaS. Different problems are studied in each of the models and novel methods are proposed to solve the respective problems.

This research enables cloud-based web service providers to offer optimised

auto-scaling services with high quality performance while incurring minimal cost, and provides new energy-saving solutions to achieve green computing in the big data era.

As shown in the Figure 7.1, this work solves several critical auto-scaling problems at different levels in the cloud paradigm. Particularly, Chapter 3 investigates how to optimise the auto-scaling program for cloud-based web services, and Chapter 4 proposes an optimised service configuration scheme at the level of PaaS. Chapter 5 and 6 solve the infrastructure auto-scaling problems from the view of VMs and bandwidths correspondingly.

7.3 Limitations and Further Studies

Due to the restriction of time, this research work has some limitations.

- **Limitation 1**: The proposed methods are not applied to real cloud service providers to implement real world problems (e.g. Big Data); This limitation will decrease the potential commercial value of our research.

- **Limitation 2**: For IaaS, storage is another very important infrastruture resource in cloud; however, it is not discussed in this study. It has not impact the accomplishment of this thesis because most of Web service caring about VMs.

- **Limitation 3**: A hybrid solution for cloud-based web service optimisation

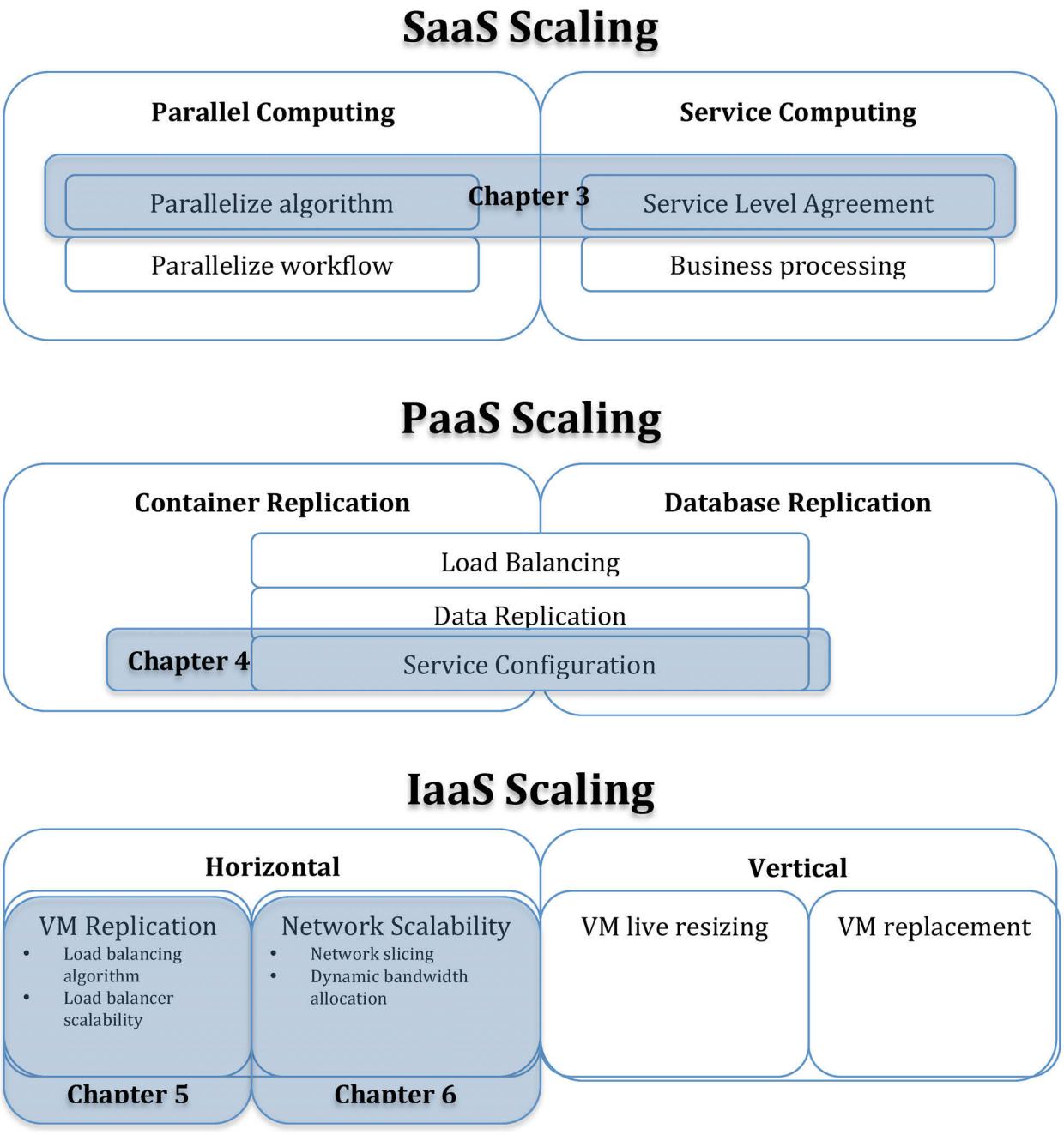


Figure 7.1: Summary of Cloud Scaling Techniques

where there is involvement in multiple cloud business models is not studied. This is a another emerging application scenario for cross-platform trend in cloud computing. Our proposed method focus on solve challenges in single-paltform scenario of cloud computing.

Auto-scaling optimisation for cloud-based web services is a very challenging research task. Many problems still need to be solved. The research can be fully advanced in the following ways:

- Challenges from the real world. Two questions should be answered: How can the method be evaluated in a real cloud-based web service provider? How can the emerging problem of the real world be solved? **(target Limitation 1)**

- In the big data era, large-scale and multiple-source data access and process will be the new challenges of web service auto-scaling. New methods need to be studied. **(target Limitation 1)**

- Storage is an important IaaS resource in cloud. Solving the optimisation of storage problem can be of benefit by not only saving the cost of disks and transmission but also by reducing the response time for data-intensive applications. Therefore, the optimisation problems of storage resource for cloud-based web services need to be defined, to be solved in future study. **(target Limitation 2)**

- Hybrid solutions. The proposed methods in this work focus on one cloud business model only, and the auto-scaling optimisation on multiple cloud business models will be studied in a real world system. Future study will focus on how to combine the current proposed methods to form a hybrid solution. **(target Limitation 3)**

Abbreviations

ACWS Auto-scaling Cloud-based Web Service

AMI Amazon Machine Images

API Application Program Interface

AR AutoRegression

AWS Amazon Web Services

BFD Best Fit Decreasing

CDN Content Delivery Network

CDNI Content Delivery Network Interconnection

CF Collaborative Filtering

CPU Central Processing Unit

DAS Dynamic Auto-Scaling

DFS Distributed File System

DNS Domain Name System

DRAS Distributed and Robust Auto-Scaling

EC2 Elastic Compute Cloud

ERP Enterprise Resource Planning

FAR Feature selection based AutoRegression

FCFS First-Come, First-Served

GOSC Genetic Optimisation based Service Configuration

HDFS Hadoop Distributed File System

I/O Input/Output

IaaS Infrastructure as a Service

ICF Item-based Collaborative Filtering

IETF Internet Engineering Task Force

ISP Internet Service Provider

KL Kullback-Leibler

LLC Limited Lookahead Control

MAE Mean Absolute Error

NN Neural Network

PaaS Platform as a Service

QoS Quality of Service

QT Queueing Theory

RAE Relative Absolute Error

RMSE Root Mean Squared Error

RSE Relative Squared Error

RTT Round-Trip Time

SaaS Software as a Service

SDN Software Defined Network

SKL Symmetrizing Kullback-Leibler

SLA Service-Level Agreement

TDM Threshold Detection Method

VDC Virtual Data Centre

VM Virtual Machine

VN Virtual Network

VoD Video-on-Demand

References

[1] Amazon auto scaling. `http://aws.amazon.com/autoscaling/`. (Cited on pages 23 and 25.)

[2] The netflix blog in 2010: Four reasons we choose amazons cloud as our computing platform. http://techblog.netflix.com/2010/12/four-reasons-we-choose-amazons-cloud-as.html. (Cited on page 113.)

[3] Rightscale. `http://www.rightscale.com/`. (Cited on page 25.)

[4] Youtube statistics. *http://www.youtube.com/yt/press/statistics.html*, 2014. (Cited on page 40.)

[5] Sherif Abdelwahed, Nagarajan Kandasamy, and Sandeep Neema. Online control for self-management in computing systems. In *The 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 368–375, 2004. (Cited on page 38.)

[6] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Volker Hilt, and Zhi-Li Zhang.

A tale of three cdns: An active measurement study of hulu and its cdns. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 7–12, 2012. (Cited on page 41.)

[7] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhi-Li Zhang. Unreeling netflix: Understanding and improving multi-cdn movie delivery. In *Proceedings of IEEE INFOCOM*, pages 1620–1628, 2012. (Cited on page 40.)

[8] Vijay Kumar Adhikari, Sourabh Jain, Yingying Chen, and Zhi-Li Zhang. Vivisecting youtube: An active measurement study. In *Proceedings of IEEE INFOCOM*, pages 2521–2525, 2012. (Cited on page 40.)

[9] Mansoor Alicherry and TV Lakshman. Network aware resource allocation in distributed clouds. In *Proceedings of IEEE INFOCOM*, pages 963–971, 2012. (Cited on page 42.)

[10] FJ Almeida Morais, F Vilar Brasileiro, R Vigolvino Lopes, R Araujo Santos, Wade Satterfield, and Leandro Rosa. Autoflex: Service agnostic autoscaling framework for iaas deployment models. In *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 42–49, 2013. (Cited on page 21.)

[11] Gene M Amdahl. Validity of the single processor approach to achieving

large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, pages 483–485, 1967. (Cited on pages 58 and 62.)

[12] Artur Andrzejak, Derrick Kondo, and Sangho Yi. Decision model for cloud computing under sla constraints. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pages 257–266, 2010. (Cited on page 71.)

[13] Adnan Ashraf. Cost-efficient virtual machine provisioning for multi-tier web applications and video transcoding. In *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 66–69, 2013. doi: 10.1109/CCGrid.2013.24. (Cited on page 21.)

[14] Siamak Azodolmolky, Philipp Wieder, and Ramin Yahyapour. Sdn-based cloud computing networking. In *The 15th International Conference on Transparent Optical Networks (ICTON)*, pages 1–4, 2013. (Cited on page 114.)

[15] Anton Beloglazov and Rajkumar Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *Cluster, Cloud and Grid Computing*

(CCGrid), 2010 10th IEEE/ACM International Conference on, pages 577–578, 2010. (Cited on page 39.)

[16] Milind Bhandarkar. Mapreduce programming with apache hadoop. In *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–1, 2010. (Cited on page 31.)

[17] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic placement of virtual machines for managing sla violations. In *10th IFIP/IEEE International Symposium on Integrated Network Management, IM'07.*, pages 119–128, 2007. (Cited on page 35.)

[18] Peter J Brockwell and Richard A Davis. *Time series: theory and methods*. Springer Science & Business Media, 2009. (Cited on page 132.)

[19] Richard Brown. Report to congress on server and data center energy efficiency: Public law 109-431. *Lawrence Berkeley National Laboratory*, 2008. (Cited on page 30.)

[20] Rajkumar Buyya, Suraj Pandey, and Christian Vecchiola. Cloudbus toolkit for market-oriented cloud computing. In *Cloud Computing*, pages 24–44. Springer, 2009. (Cited on page 18.)

[21] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. Modeling and simulation of scalable cloud computing environments and the cloudsim

toolkit: Challenges and opportunities. In *International Conference on High Performance Computing & Simulation, HPCS'09.*, pages 1–11, 2009. (Cited on pages 36 and 79.)

[22] Rajkumar Buyya, Anton Beloglazov, and Jemal Abawajy. Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. *arXiv preprint arXiv:1006.0308*, 2010. (Cited on page 36.)

[23] Junwei Cao, Kai Hwang, Keqin Li, and Albert Y Zomaya. Optimal multiserver configuration for profit maximization in cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1087–1096, 2013. (Cited on page 28.)

[24] Dukhyun Chang, Junho Suh, Hyogi Jung, Ted Taekyoung Kwon, and Yanghee Choi. How to realize cdn interconnection (cdni) over openflow? In *Proceedings of the 7th International Conference on Future Internet Technologies*, pages 29–30, 2012. (Cited on page 41.)

[25] Ruiqing Chi, Zhuzhong Qian, and Sanglu Lu. A game theoretical method for auto-scaling of multi-tiers web applications in cloud. In *Proceedings of the Fourth Asia-Pacific Symposium on Internetware*, page 3, 2012. (Cited on page 35.)

[26] Trieu C Chieu, Ajay Mohindra, Alexei A Karve, and Alla Segal. Dynamic scaling of web applications in a virtualized cloud computing environment. In *IEEE International Conference on e-Business Engineering, ICEBE'09.*, pages 281–286, 2009. (Cited on page 24.)

[27] Cheng Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19:281, 2007. (Cited on pages 45 and 61.)

[28] Adrian Cockcroft. Netflix cloud architecture. 2011. (Cited on page 40.)

[29] Reginald Cushing, Spiros Koulouzis, Adam SZ Belloum, and Marian Bubak. Prediction-based auto-scaling of scientific workflows. In *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*, page 1, 2011. (Cited on page 31.)

[30] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. (Cited on page 45.)

[31] Sandvine Network Demographics. Global internet phenomena report: Spring 2011, 2011. (Cited on page 40.)

[32] Da Deng, ZhiHui Lu, Wei Fang, and Jie Wu. Cloudstreammedia: a cloud

assistant global video on demand leasing scheme. In *IEEE International Conference on Services Computing (SCC)*, pages 486–493, 2013. (Cited on page 41.)

[33] Brian Dougherty, Jules White, and Douglas C Schmidt. Model-driven auto-scaling of green cloud computing infrastructure. *Future Generation Computer Systems*, 28(2):371–378, 2012. (Cited on pages 27 and 39.)

[34] Frank Dürr. Towards cloud-assisted software-defined networking. Technical report, Institute of Parallel and Distributed Systems, Universität Stuttgart, 2012. (Cited on page 114.)

[35] Jaliya Ekanayake, Shrideep Pallickara, and Geoffrey Fox. Mapreduce for data intensive scientific analyses. In *IEEE Fourth International Conference on eScience, eScience'08.*, pages 277–284, 2008. (Cited on page 45.)

[36] Armando Fox, Rean Griffith, A Joseph, R Katz, A Konwinski, G Lee, D Patterson, A Rabkin, and I Stoica. Above the clouds: A berkeley view of cloud computing. *Technical Report UCB/EECS-2009-28, Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley*, 28:13, 2009. (Cited on page 23.)

[37] Archana Ganapathi, Yanpei Chen, Armando Fox, Randy Katz, and David Patterson. Statistics-driven workload modeling for the cloud. In *IEEE 26th*

International Conference on Data Engineering Workshops (ICDEW), pages 87–92, 2010. (Cited on page 72.)

[38] Anshul Gandhi, Varun Gupta, Mor Harchol-Balter, and Michael A Kozuch. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation*, 67(11):1155–1171, 2010. (Cited on page 23.)

[39] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A Kozuch. Distributed, robust auto-scaling policies for power management in compute intensive server farms. In *Open Cirrus Summit (OCS), 2011 Sixth*, pages 1–5, 2011. (Cited on page 39.)

[40] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A Kozuch. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Transactions on Computer Systems (TOCS)*, 30(4): 14:1–14:25, 2012. (Cited on pages 21 and 27.)

[41] Hamoun Ghanbari, Bradley Simmons, Marin Litoiu, and Gabriel Iszlai. Exploring alternative approaches to implement an elasticity policy. In *IEEE International Conference on Cloud Computing (CLOUD)*, pages 716–723, 2011. (Cited on page 35.)

[42] Hamoun Ghanbari, Bradley Simmons, Marin Litoiu, Cornel Barna, and Gabriel Iszlai. Optimal autoscaling in a iaas cloud. In *Proceedings of the 9th*

International Conference on Autonomic Computing, pages 173–178, 2012. (Cited on page 38.)

[43] Hamoun Ghanbari, Bradley Simmons, Marin Litoiu, and Gabriel Iszlai. Feedback-based optimization of a private cloud. *Future Generation Computer Systems*, 28(1):104–111, 2012. (Cited on page 34.)

[44] Zhenhuan Gong, Xiaohui Gu, and John Wikes Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *International Conference on Network and Service Management (CNSM)*, pages 9–16, 2010. (Cited on page 26.)

[45] Hadi Goudarzi, Mohammad Ghasemazar, and Massoud Pedram. Sla-based optimization of power and migration cost in cloud computing. In *The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 172–179, 2012. (Cited on page 26.)

[46] Donald Gross, John F. Shortle, James M. Thompson, and Carl M. Harris. *Fundamentals of queueing theory*, volume 627. Wiley-Interscience, 2011. (Cited on pages 96, 98, and 124.)

[47] Donald Gross, John F Shortle, James M Thompson, and Carl M Harris. *Fundamentals of queueing theory*. Wiley. com, 2013. (Cited on pages 126 and 128.)

[48] Chuanxiong Guo, Guohan Lu, Helen J Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, and Yongguang Zhang. Secondnet: A data center network virtualization architecture with bandwidth guarantees. In *Proceedings of the 6th International Conference*, page 15, 2010. (Cited on page 41.)

[49] Yang Guo, Alexander L Stolyar, and Anwar Walid. Online algorithms for joint application-vm-physical-machine auto-scaling in a cloud. Technical report, Bell Labs Technical Memo, 2014. http://ect.bell-labs.com/who/stolyar/publications/gswsigm-14-full.pdf. (Cited on page 42.)

[50] Rui Han, Li Guo, Moustafa M. Ghanem, and Yike Guo. Lightweight resource scaling for cloud applications. In *The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 644–651, 2012. (Cited on page 27.)

[51] Alexandru Iosup, Simon Ostermann, M Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick HJ Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):931–945, 2011. (Cited on page 76.)

[52] Waheed Iqbal, Matthew N Dailey, David Carrera, and Paul Janecek. Adap-

tive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, 27(6):871–879, 2011. (Cited on page 33.)

[53] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, 28(1):155–162, 2012. (Cited on page 26.)

[54] Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. Autonomic resource provisioning for cloud-based software. In *SEAMS*, pages 95–104, 2014. (Cited on page 35.)

[55] Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. Optimal cloud resource auto-scaling for web applications. In *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 58–65, 2013. (Cited on pages 21, 28, and 38.)

[56] Yexi Jiang, Chang-shing Perng, Tao Li, and Rong Chang. Self-adaptive cloud capacity planning. In *IEEE Ninth International Conference on Services Computing (SCC)*, pages 73–80, 2012. (Cited on page 27.)

[57] Yang Jie, Jie Qiu, and Ying Li. A profile-based approach to just-in-time scalability for cloud applications. In *IEEE International Conference on Cloud Computing, CLOUD'09*, pages 9–16, 2009. (Cited on page 24.)

[58] Don H. Johnson and Sinan Sinanovic. Symmetrizing the kullback-leibler distance. *IEEE Transactions on Information Theory*, 1(1):1–10, 2001. (Cited on page 104.)

[59] Farhana Kabir and David Chiu. Reconciling cost and performance objectives for elastic web caches. In *Proceedings of the 2012 IEEE International Conference on Cloud and Services Computing (CSC'12)*, 2012. (Cited on page 27.)

[60] Evangelia Kalyvianaki and Steven Hand. Applying kalman filters to dynamic resource provisioning of virtualized server applications. In *Proccedings of the 3rd International Workshop Feedback Control Implementation and Design in Computing Systems and Networks (FeBid)*, page 6, 2008. (Cited on page 37.)

[61] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th International Conference on Autonomic Computing*, pages 117–126, 2009. (Cited on pages 26 and 37.)

[62] A Karve, Tracy Kimbrel, Giovanni Pacifici, Mike Spreitzer, Malgorzata Steinder, Maxim Sviridenko, and A Tantawi. Dynamic placement for clus-

tered web applications. In *Proceedings of the 15th International Conference on World Wide Web*, pages 595–604, 2006. (Cited on page 35.)

[63] Seoyoung Kim, Jik-Soo Kim, Soonwook Hwang, and Yoonhee Kim. An allocation and provisioning model of science cloud for high throughput computing applications. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, page 27, 2013. (Cited on page 36.)

[64] Vipin Kumar and Vineet Singh. Scalability of parallel algorithms for the all-pairs shortest-path problem. *Journal of Parallel and Distributed Computing*, 13(2):124–138, 1991. (Cited on page 63.)

[65] Dara Kusic, Jeffrey O Kephart, James E Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009. (Cited on page 38.)

[66] Horacio Andrés Lagar-Cavilla, Joseph Andrew Whitney, Adin Matthew Scannell, Philip Patchin, Stephen M Rumble, Eyal De Lara, Michael Brudno, and Mahadev Satyanarayanan. Snowflock: Rapid virtual machine cloning for cloud computing. In *Proceedings of the 4th ACM European Conference on Computer Systems*, pages 1–12, 2009. (Cited on page 23.)

[67] Aris Leivadeas, C Papagianni, and Symeon Papavassiliou. Energy aware

networked cloud mapping. In *The 12th IEEE International Symposium on Network Computing and Applications (NCA)*, pages 195–202, 2013. (Cited on page 37.)

[68] Jacob Leverich and Christos Kozyrakis. On the energy (in)efficiency of hadoop clusters. *ACM SIGOPS Operating Systems Review*, 44(1):61–65, 2010. (Cited on page 44.)

[69] Chu-Fen Li. Cloud computing system management under flat rate pricing. *Journal of Network and Systems Management*, 19(3):305–318, 2011. (Cited on page 19.)

[70] Hui Li, Giuliano Casale, and Tariq Ellahi. Sla-driven planning and optimization of enterprise applications. In *Proceedings of the first joint WOSP/SIPEW International Conference on Performance Engineering*, pages 117–128, 2010. (Cited on page 31.)

[71] Jim Li, John Chinneck, Murray Woodside, Marin Litoiu, and Gabriel Iszlai. Performance model driven qos guarantees and optimization in clouds. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 15–22, 2009. (Cited on page 34.)

[72] Jim Zhanwen Li, John Chinneck, Murray Woodside, and Marin Litoiu. Fast scalable optimization to configure service systems having cost and quality

of service constraints. In *Proceedings of the 6th International Conference on Autonomic Computing*, pages 159–168, 2009. (Cited on page 35.)

[73] Harold C. Lim, Shivnath Babu, and Jeffrey S. Chase. Automated control for elastic storage. In *Proceedings of the 7th international conference on Autonomic computing*, pages 1–10, 2010. (Cited on page 26.)

[74] Xue Liu, Xiaoyun Zhu, Pradeep Padala, Zhikui Wang, and Sharad Singhal. Optimal multivariate control for differentiated services on a shared hosting platform. In *The 46th IEEE Conference on Decision and Control*, pages 3792–3799, 2007. (Cited on page 38.)

[75] Muhammad Salman Malik, Mirko Montanari, Jun Ho Huh, Rakesh B Bobba, and Roy H Campbell. Towards sdn enabled network control delegation in clouds. In *The 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–6, 2013. (Cited on page 114.)

[76] Sunilkumar S Manvi and Gopal Krishna Shyam. Resource management for infrastructure as a service (iaas) in cloud computing: A survey. *Journal of Network and Computer Applications*, 41:424–440, 2014. (Cited on page 36.)

[77] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proceedings of 2011 Interna-*

tional Conference for High Performance Computing, Networking, Storage and Analysis, page 49, 2011. (Cited on pages 21, 28, and 36.)

[78] Ming Mao, Jie Li, and Marty Humphrey. Cloud auto-scaling with deadline and budget constraints. In *The 11th IEEE/ACM International Conference on Grid Computing (GRID)*, pages 41–48, 2010. (Cited on page 36.)

[79] Michele Mazzucco and Dmytro Dyachuk. Optimizing cloud providers revenues via energy efficient server allocation. *Sustainable Computing: Informatics and Systems*, 2(1):1–12, 2012. (Cited on page 38.)

[80] Michele Mazzucco, Dmytro Dyachuk, and Ralph Deters. Maximizing cloud providers' revenues via energy aware allocation policies. In *IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 131–138, 2010. (Cited on page 38.)

[81] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011. (Cited on page 18.)

[82] Daniel A Menascé, John M Ewing, Hassan Gomaa, Sam Malex, and João P Sousa. A framework for utility-based service oriented design in sassy. In *Proceedings of the first joint WOSP/SIPEW International Conference on Performance Engineering*, pages 27–36, 2010. (Cited on page 75.)

[83] Bradley N Miller, Istvan Albert, Shyong K Lam, Joseph A Konstan, and

John Riedl. Movielens unplugged: experiences with an occasionally connected recommender system. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, pages 263–266, 2003. (Cited on page 61.)

[84] Amit Nathani, Sanjay Chaudhary, and Gaurav Somani. Policy based resource allocation in iaas cloud. *Future Generation Computer Systems*, 28 (1):94–103, 2012. (Cited on page 25.)

[85] Di Niu, Hong Xu, Baochun Li, and Shuqiao Zhao. Quality-assured cloud bandwidth auto-scaling for video-on-demand applications. In *Proceedings of the 2012 IEEE INFOCOM*, pages 460–468, 2012. (Cited on pages 22 and 41.)

[86] Pradeep Padala, Kai-Yuan Hou, Kang G. Shin, Xiaoyun Zhu, et al. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 13–26, 2009. (Cited on page 25.)

[87] Umakishore Ramachandran, H Venkateswaran, Anand Sivasubramaniam, and Aman Singla. Issues in understanding the scalability of parallel systems. In *Proceedings of the First International Workshop on Parallel Processing, Bangalore, India*, pages 399–404, 1994. (Cited on page 62.)

[88] Luis Rodero-Merino, Luis M Vaquero, Victor Gil, Fermín Galán, Javier

Fontán, Rubén S Montero, and Ignacio M Llorente. From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, 26(8):1226–1240, 2010. (Cited on page 37.)

[89] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *IEEE International Conference on Cloud Computing (CLOUD)*, pages 500–507, 2011. (Cited on page 26.)

[90] Akkarit Sangpetch, Andrew Turner, and Hyong Kim. How to tame your vms: an automated control system for virtualized services. In *Proceedings of the 24th International Conference on Large Installation System Administration*, pages 1–16, 2010. (Cited on page 26.)

[91] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295, 2001. (Cited on page 50.)

[92] Mina Sedaghat, Francisco Hernandez-Rodriguez, and Erik Elmroth. A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, page 6, 2013. (Cited on page 42.)

[93] Ahmed A Soror, Umar Farooq Minhas, Ashraf Aboulnaga, Kenneth Salem, Peter Kokosielis, and Sunil Kamath. Automatic virtual machine configuration for database workloads. *ACM Transactions on Database Systems (TODS)*, 35(1):7, 2010. (Cited on page 34.)

[94] Donald F Specht. A general regression neural network. *IEEE Transactions on Neural Networks*, 2(6):568–576, 1991. (Cited on page 120.)

[95] Moritz Steiner, Bob Gaglianello Gaglianello, Vijay Gurbani, Volker Hilt, William D Roome, Michael Scharf, and Thomas Voith. Network-aware service placement in a distributed cloud environment. *ACM SIGCOMM Computer Communication Review*, 42(4):73–74, 2012. (Cited on page 42.)

[96] Christopher Stewart, Terence Kelly, and Alex Zhang. Exploiting nonstationarity for performance prediction. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 31–44, 2007. (Cited on page 76.)

[97] Alexander L Stolyar. An infinite server system with general packing constraints. *Operations Research*, 61(5):1200–1217, 2013. (Cited on page 39.)

[98] Lik Gan Alex Sung and Johnny W Wong. Autonomic resource management for a cluster that executes batch jobs. In *IEEE International Conference on Cluster Computing*, pages 1–10, 2006. (Cited on page 73.)

[99] Gerald Tesauro. Reinforcement learning in autonomic computing: A man-

ifesto and case studies. *Internet Computing, IEEE*, 11(1):22–30, 2007. (Cited on page 73.)

[100] Hendrik Cornelis Tijms, Michiel Harpert Van Hoorn, and Awi Federgruen. Approximations for the steady-state probabilities in the m/g/c queue. *Advances in Applied Probability*, pages 186–206, 1981. (Cited on page 121.)

[101] Ruben Torres, Alessandro Finamore, Jin Ryong Kim, Marco Mellia, Maurizio M Munafo, and Sanjay Rao. Dissecting video server selection strategies in the youtube cdn. In *The 31st International Conference on Distributed Computing Systems (ICDCS)*, pages 248–257, 2011. (Cited on page 40.)

[102] Dimitrios Tsoumakos, Ioannis Konstantinou, Christina Boumpouka, Spyros Sioutas, and Nectarios Koziris. Automated, elastic resource provisioning for nosql clusters using tiramola. In *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 34–41, 2013. (Cited on page 21.)

[103] Luis M Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011. (Cited on pages 31 and 37.)

[104] Meng Wang, Xiaoqiao Meng, and Li Zhang. Consolidating virtual machines

with dynamic bandwidth demand in data centers. In *Proceedings of the 2011 IEEE INFOCOM*, pages 71–75, 2011. (Cited on page 21.)

[105] XiaoYing Wang, DongJun Lan, Gang Wang, Xing Fang, Meng Ye, Ying Chen, and QingBo Wang. Appliance-based autonomic provisioning framework for virtualized outsourcing data center. In *The Fourth International Conference on Autonomic Computing, ICAC'07*, pages 29–29, 2007. (Cited on page 39.)

[106] Tom White. *Hadoop: The definitive guide*. O'Reilly Media, 2012. (Cited on page 31.)

[107] Wayne A. Woodward, Henry L. Gray, and Alan C. Elliot. *Applied Time Series Analysis*. CRC Press, 2011. 2011. ISBN 9781439818374. (Cited on page 95.)

[108] Linlin Wu, Saurabh Kumar Garg, and Rajkumar Buyya. Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In *The 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 195–204, 2011. (Cited on page 71.)

[109] Cheng-Zhong Xu, Jia Rao, and Xiangping Bu. Url: A unified reinforcement

learning approach for autonomic cloud management. *Journal of Parallel and Distributed Computing*, 2011. (Cited on pages 25, 28, and 93.)

[110] Zhao Yong-quan, Lu Li-bin, and Fang Shu-fen. Stochastic linear optimization for modeling uncertainty in aggregate production planning. In *International Conference on Autonomic and Autonomous Systems*, pages 31–31, 2006. (Cited on page 73.)

[111] Xiaodong Zhang, Yong Yan, and Qian Ma. Measuring and analyzing parallel computing scalability. In *International Conference on Parallel Processing*, volume 2, pages 295–303, 1994. (Cited on page 62.)

[112] Ying Zhang, Gang Huang, Xuanzhe Liu, and Hong Mei. Integrating resource consumption and allocation for infrastructure resources on-demand. In *IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 75–82, 2010. (Cited on page 24.)

[113] Weibin Zhao and Henning Schulzrinne. Predicting the upper bound of web traffic volume using a multiple time scale approach. In *Proceedings of International World Wide Web Conference (WWW)*, page 251, 2003. (Cited on page 93.)

[114] Weizhong Zhao, Huifang Ma, and Qing He. Parallel k-means clustering

based on mapreduce. In *Cloud Computing*, pages 674–679. Springer, 2009. (Cited on pages 45 and 61.)