Lifetime Quality Management for Multi-Tenant Service-Based Systems

by

Yanchun Wang

M.Eng. (BUAA)

A thesis submitted to

Swinburne University of Technology

for the degree of

Doctor of Philosophy

November 2017

Abstract

The proliferation of cloud computing has fuelled the rapid growth of multi-tenant service-based systems (SBSs), which serve multiple tenants simultaneously by composing existing services in the form of business processes. It has been a growing challenge to provide quality guarantee across the whole lifetime of an SBS, including service selection, service monitoring and service adaptation. In the cloud environment, the tenants that share a multi-tenant SBS often have differentiated quality requirements, quality preferences and different business scenarios, which has significantly complicated the quality management for multi-tenant SBSs at different lifetime stages.

In this thesis, we propose a systematic solution named LQM4MTS (Lifetime Quality Management for Multi-Tenant SBSs), which consists of a set of techniques to address the major issues of quality management that arise at different stages of the lifetime of multi-tenant SBSs. Specifically, an innovative service selection approach using service recommendation based on clustering techniques is proposed to facilitate efficient service selection for multi-tenant SBSs at build-time; a new service selection approach is proposed to support tenants' correlated quality requirements; a novel approach for service monitoring is proposed to formulate criticality-based cost-effective monitoring strategies for multi-tenant SBSs; an innovative service recommendation approach based on Locality Sensitive Hashing (LSH) is proposed to achieve quick service adaptation at runtime upon anomalies; and a novel fault tolerance approach is proposed to improve the ability of multi-tenant SBSs in handling runtime anomalies by formulating cost-effective fault tolerance strategies based on criticality. Experimental results show that our approaches 1) guarantee high effectiveness whilst significantly improve the efficiency of service selection for multi-tenant SBSs with clustering-based service recommendation at build-time, especially in large-scale scenarios; 2) effectively and efficiently select services to compose multi-tenant SBSs that fulfil the tenants' correlated quality requirements; 3) remarkably improve the effectiveness especially cost-effectiveness of service monitoring for multi-tenant SBSs whilst retaining high efficiency; 4) facilitate effective and efficient runtime service adaptation based on Locality Sensitive Hashing; 5) effectively and efficiently reduce the risk of system quality degradation upon runtime anomalies with the fault tolerance strategies formulated for multi-tenant SBSs.

The major contribution of this research is to propose a systematic solution to lifetime quality management for multi-tenant SBSs in the cloud environment. With the new approaches proposed, the quality of the multi-tenant SBSs can be guaranteed at different stages of the lifetime of the SBSs.

The Author's Publications

Published:

[1] **Y. Wang**, Q. He, D. Ye and Y. Yang, "Formulating Criticality-Based Cost-Effective Fault Tolerance Strategies for Multi-Tenant Service-Based Systems," *IEEE Transactions on Software Engineering (TSE)*, accepted on Mar. 5, 2017, http://dx.doi.org/10.1109/TSE.2017.2681667.

[2] **Y. Wang**, Q. He, X. Zhang, D. Ye and Y. Yang, "Efficient QoS-Aware Service Recommendation for Multi-Tenant Service-Based Systems in Cloud," *IEEE Transactions on Services Computing (TSC)*, accepted on Sept. 27, 2017, http://dx.doi.org/10.1109/TSC.2017.2761346.

[3] **Y. Wang**, Q. He, D. Ye, Y. Yang, "Formulating Criticality-Based Cost-Effective Monitoring Strategy for Multi-Tenant Service-Based Systems," in *Proceedings of the 24th IEEE International Conference on Web Services (ICWS2017)*, Honolulu, Hawaii, USA, 2017, pp. 325-332.

[4] **Y. Wang**, Q. He, D. Ye and Y. Yang, "Service Selection based on Correlated QoS Requirements," in *Proceedings of the 14th IEEE International Conference on Services Computing (SCC2017)*, Honolulu, Hawaii, USA, 2017, pp. 241-248.

[5] **Y. Wang**, Q. He, Y. Yang, "QoS-Aware Service Recommendation for Multi-

Tenant SaaS on the Cloud," in *Proceedings of the 12th IEEE International Conference on Services Computing (SCC2015)*, New York, USA, 2015, pp. 178-185.

[6] D. Ye, Q. He, **Y. Wang** and Y. Yang, "An Agent-based Decentralised Service Monitoring Approach in Multi-Tenant Service-based Systems," in *Proceedings of the 24th IEEE International Conference on Web Services (ICWS2017)*, Honolulu, USA, June 2017, pp. 204-211.

[7] Q. He, X. Xie, **Y. Wang**, D. Ye, F. Chen, H. Jin and Y. Yang, "Localizing Runtime Anomalies in Service-Oriented Systems," *IEEE Transactions on Services Computing (TSC)*, vol. 10, no. 1, pp. 94-106, 2017

[8] D. Ye, Q. He, **Y. Wang** and Y. Yang, "An Agent-based Integrated Self-Evolving Service Composition Approach in Networked Environments," *IEEE Transac-tions on Services Computing (TSC)*, accepted on Nov. 17, 2016, http://dx.doi.org/10.1109/TSC.2016.2631598.

[9] Q. He, R. Zhou, X. Zhang, **Y. Wang**, D. Ye, F. Chen, J. Grundy and Y. Yang, "Keyword Search for Building Service-Based Systems," *IEEE Transactions on Software Engineering (TSE)*, vol. 43, no. 7, pp. 658-674, 2017.

[10] Q. He, J. Han, F. Chen, **Y. Wang**, R. Vasa, Y. Yang, H. Jin, "QoS-Aware Service Selection for Customisable Multi-Tenant Service-Based Systems: Maturity and Approaches," in *Proceedings of the 8th IEEE International Conference on Cloud Computing (CLOUD2015)*, New York, USA, 2015, pp. 237-244.

[11] Q. He, X. Xie, F. Chen, **Y. Wang**, R. Vasa, Y. Yang, H. Jin, "Spectrum-Based Runtime Anomaly Localisation in Service-Based Systems," in *Proceedings of the 12th IEEE International Conference on Services Computing (SCC2015)*, New York, USA, 2015, pp. 90-97.

[12] Q. He, R. Zhou, X. Zhang, **Y. Wang**, D. Ye, F. Chen, S. Chen, J. Grundy, Y. Yang, "Efficient Keyword Search for Building Service-based Systems-based on

Dynamic Programming", in Proceedings of the 15th International Conference on Service-Oriented Computing (ICSOC2017), Malaga, Spain, 2017.

Acknowledgements

My foremost thank goes to my principle coordinating supervisor, Prof. Yun Yang, and my coordinating supervisor, Dr. Qiang He. I benefit a lot from their enthusiasm to research and positive attitude to life. They are also my friends and mentors. Their guidance and keen insight, along with their thirst for new knowledge have been always stimulating and inspiring.

With a special mention to Dr. Dayong Ye. It was fantastic to work with him as a team at SUT. He has been a diligent and helpful friend.

My wife, Juan Zhou, and my two beloved kids, Ben and Ray, have always been there. Their love and understanding have given me the courage to overcome any difficulty on my way.

I am grateful to my parents, who have provided me emotional and moral support in my life. Without them, I would not be able to complete this work. I owe it all to them.

A very special gratitude goes to Prof. Shuzhen Yao and A/Prof. Guangyan Lin at Beihang University (BUAA), who had been my supervisors during my candidature as a master student. They helped me open the door for my academic journey in Australia.

Thanks to A/Prof. Jean-Guy Schneider, who has provided constructive advice on my thesis.

I am also grateful to the following friends: Dr. Feifei Chen, Dr. Dong Yuan, Dr. Miao Du, Dr. Rui Zhou, Dr. Xuyun Zhang, Ru Jia, A/Prof. Xuejun Li, Futian Wang, Dr. Rongbin Xu, Dongwei Li, Xiaodong Zhu and many others in and out of the university, who have supported me along the way. Thanks to Prof. Jun Han, Dr. Wei Lai and Dr. Caslon Chua - the panel members for my Ph.D. candidature review - for their valuable advice and support.

And finally, last but by no means least, also to all the staff in the faculty, for helping me figure out how to apply for the trips to conferences, how to organise my teaching schedules, and so much stuff that I was not familiar with. Thanks to SUT, who provided me financial support with the SUPRA scholarship. It is such a great place to work. I also gratefully acknowledge the funding received from Australian Research Council Discovery Project DP150101775, which supported me for the scholarship top-ups, conference travels, registrations, etc.

Thanks for all your encouragement.

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma, except where due reference is made in the text of the thesis. To the best knowledge, this thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

Yanchun Wang

王艳春

November 2017

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis addresses the issue of lifetime quality management for multi-tenant service-based systems (SBSs). The major contribution of this research is to propose a systematic solution that consists of a set of techniques to support quality management at different lifetime stages of multi-tenant SBSs, including service selection, service monitoring and service adaptation. To support quality-aware service selection, we propose an innovative approach for effective and efficient service selection using service recommendation based on clustering techniques to fulfil tenants' differentiated quality requirements. An approach that handles the tenants' correlated quality requirements is also presented. To support quality-aware service monitoring, we design a novel approach for formulating cost-effective monitoring strategies for multi-tenant SBSs based on service criticality. To support quality-aware service adaptation, we propose an efficient service adaptation approach based on Locality Sensitive Hashing (LSH) to implement quick service replacement for adaptation upon runtime anomalies. An innovative approach for formulating cost-effective fault tolerance strategies based on service criticality is also designed to significantly reduce the adverse impacts to the quality of multi-tenant SBSs caused by runtime anomalies.

This chapter is organised as follows: First, Section 1.1 introduces multi-tenant SBSs as background. Then Section 1.2 introduces the key issues of this research. Finally, Section 1.3 presents the overview of the structure of this thesis.

1.1 Introduction to multi-tenant SBSs

The service-oriented paradigm has become an effective way to engineer service-based systems (SBSs) by composing network-accessible (and often distributed) Web services [1][2], which collectively offer the functionality of the SBS and fulfil its quality requirements, such as cost, response time, reliability, and throughput [3]. A Web service or service is a standardised way to integrate interoperable Web-based applications over a network. It is built on top of open standards such as Simple Object Access Protocol (SOAP) [4], Web Service Description Language (WSDL)[5] and Universal Description, Discovery and Integration (UDDI) [6][7]. Existing Web services can be orchestrated in the form of business process to build an SBS. The development and popularity of e-business, e-commerce, especially the pay-as-you-go business model promoted by cloud computing, have fuelled the rapid growth of services and SBSs, which is indicated by the statistics published by ProgrammableWeb, an online Web service and Web API directory [8].

As an important characteristic of cloud computing, *multi-tenancy* refers to a software architecture that uses a single software instance to serve multiple tenants. A tenant of a multi-tenant system is an organisational entity that hosts a number of end users and subscribes to the multi-tenant system according to the pay-as-you-go business model. For example, a company that subscribes to a multi-tenant CRM (Customer Relationship Management) system provided by Salesforce[1] is one of the tenants that share the system. The employees of that company are the end users of the system. A multi-tenant SBS provides multiple tenants with similar and yet customised functionalities with potentially different

[1]`https://www.salesforce.com`

quality values [9]. Multiple tenants can share all or part of the services in an SBS. By sharing the cloud resources across tenants, multi-tenancy allows SBS vendors to achieve economies of scale and optimisation in terms of speed, security, availability, disaster recovery, and maintenance [10].

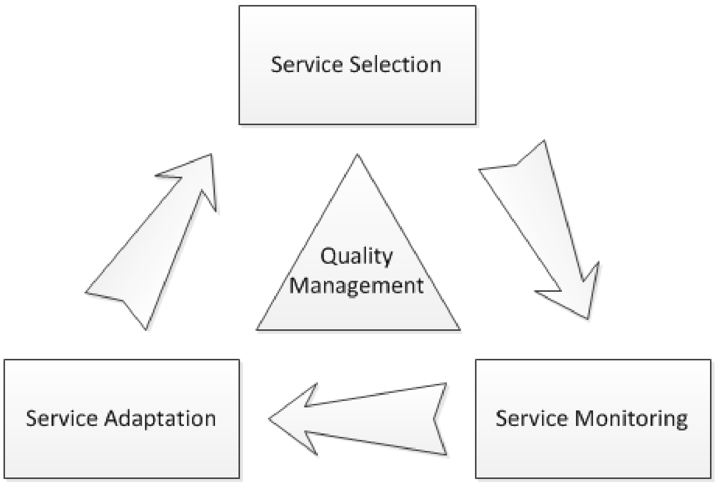


Figure 1.1: The lifetime of service-based systems (SBSs).

As shown in Fig. 1.1, the whole lifetime of an SBS consists of three stages: build time service selection for composition, runtime service monitoring and, if needed, service adaptation for delivery [11]. In the cloud especially the multi-tenant environment, these stages are briefly introduced as follows:

1. **Service selection**: SBS vendors build an SBS by selecting from existing services (referred to as *candidate services*) to create a service composition, which can fulfil tenants' functional and multi-dimensional quality requirements. Moreover, there are often optimisation goals for the SBS, e.g. minimised cost or maximised performance, which must also be achieved in the process of service selection [12].

2. **Service monitoring**: After an SBS is formulated by service selection and put into operation, the selected services in the SBS (referred to as *component services*) must be monitored to provide substantial quality guarantee to all tenants by timely detecting runtime anomalies.

3. **Service adaptation**: Once the runtime anomalies are detected, service adaptation actions, e.g., service replacement, service re-optimisation, etc., can be taken immediately to reduce the risk of quality violation for all tenants.

Improving effectiveness and efficiency of lifetime quality management of SBSs are of paramount importance. First, due to the proliferation of cloud computing, more and more cloud services with equivalent functionality are emerging, characterised by different multi-dimensional quality values [13]. Selecting component services to compose an SBS to fulfil its quality requirements while achieving the optimisation goal is a multi-criteria decision problem, which is NP-complete and often very complex and time consuming, especially in large-scale scenarios [14]. In addition, the quality requirements for an SBS is not always deterministic and correlations may exist between different quality dimensions of the requirements, which has made the service selection more complicated. Second, with the increasing deployment of public, private and hybrid clouds, various cloud services are provided by different vendors in different geographical locations. Due to many uncertain factors, e.g., techniques used to implement the services, the network condition, etc., the cloud environment in which the SBSs operate tends to be dynamic and volatile [15]. Unexpected runtime anomalies may occur in its component services [16]. If a runtime anomaly cannot be fixed in a timely manner, it often leads to degradation and end-to-end violation in multi-dimensional system quality, incurring penalties and losses caused by Service Level Agreement (SLA) breach. Amazon reports that by adding only 100ms in response time, sales drop by 1% [17]. An e-commerce system can lose thousands of dollars for every minute of its unavailability [18]. In such context, an SBS must be monitored to timely detect runtime anomalies, which, once occur, must be handled effectively and efficiently with service adaptation to guarantee the quality of the SBS.

In spite of the benefits of multi-tenancy, such as high scalability, maximised performance, easy system upgrade, etc., new challenges and issues in lifetime quality management have been identified when engineering multi-tenant SBSs in the cloud. On one hand, tenants often have diverse quality preferences and requirements [19]. For example, a tenant of an SBS may have strong preference for response time, while another may prefer high system throughput. Thus, the SBS must provide differentiated quality to satisfy the tenants. Taking Dropbox for example, as a popular application on the cloud that provides services of data storage, synchronisation and sharing, it offers free services with limited quality to light-weight users, while for the users who have strict requirements on quality, the business versions with premium features are available at an extra charge. On the other hand, the tenants share all or part of the component services due to their different business scenarios. The number of tenants that a component service serves simultaneously may vary dramatically. It is possible that one component service in an SBS is shared by a large number of tenants while another service in the same SBS only serves for a few tenants. In such a context, SBS vendors have to consider the diversity in tenants' quality preferences and requirements as well as the service sharing in an SBS in quality management across all lifetime stages of the SBS, and endeavour to guarantee the quality of the SBS for all tenants. Thus, the transition from single-tenancy to multi-tenancy has complicated the quality management of SBSs significantly.

Therefore, it is critical and challenging to manage the quality of multi-tenant SBSs across the entire lifetime for business competitiveness. The lifetime quality management is a fundamental and urgent requirement for the wide application of multi-tenant SBSs in the cloud.

1.2 Key issues of this research

In order to achieve lifetime quality management of multi-tenant SBSs, the following key issues need to be addressed:

- **For service selection**: A lot of research efforts have been devoted to the problem of quality-aware service selection, and various approaches have been proposed to address this issue, such as the work in [13][20][21]. However, some critical issues still remain to be addressed. In this thesis, we focus on two of them: improving the efficiency of service selection whilst retaining high effectiveness and handling the correlations in tenants' quality requirements.

First, as discussed in Section 1.1, the aim of service selection for building a multi-tenant SBS is to fulfil the tenants' diverse multi-dimensional quality requirements, while achieving the optimisation goal for the SBS in the meantime [22]. This is an NP-complete problem [14] that can be computationally expensive, especially in large-scale scenarios. Thus, how to improve the efficiency whilst retaining high effectiveness of build-time service selection for building multi-tenant SBSs has become a critical issue. Service recommendation, as a preferred way to address this issue, has attracted considerable attention in recent years [23][24]. However, most of the approaches for service selection and recommendation are used in single tenant environment, and the differentiation of quality preferences and requirements of the tenants that share the same SBS is not sufficiently considered.

Second, tenant's quality requirements can be correlated, i.e., there is tradeoff between different quality dimensions of a tenant's requirement for the SBS. For example, a tenant of an SBS may accept a lower throughput at a lower cost, but is willing to pay more if a higher throughput is provided. In this way, a tenant's multi-dimensional quality requirements become dynamically varied and correlated. This is a real-world scenario but not sufficiently considered. It is significantly different from the scenarios handled by most existing works in service selection [20][21][25], where all the quality requirements are deterministic and different quality dimensions in the requirements are independent of each other. How to select proper services for compositions to fulfil multiple tenants' correlated quality requirements has become a new challenge, which makes the existing approaches impractical.

- **For service monitoring**: In order to facilitate timely adaptation upon anomalies to guarantee the quality of an SBS, the component services of the SBS must be monitored. Many approaches for service monitoring have been proposed during the past decade [2][3]. However, most of them treat all component services in an SBS equally and constantly monitor all services whilst cost-effectiveness has not been paid enough attention when formulating monitoring strategies. Service monitoring produces benefits (*monitoring benefits* for short) because adaptation actions can be taken timely to fix the anomalies before they cause tenant-perceived quality degradation. However, constantly monitoring all services in an SBS is often impractical. On one hand, hardware, software and sometimes human resources are needed to implement monitoring strategies. Monitoring all services constantly introduces excessive *resource cost*. On the other hand, service monitoring often incurs *system overhead* and impacts the quality of the SBS. For example, the retrieval of system log, the sniff of network traffic, etc., can introduce up to 70% performance overhead [3]. In such a context, an approach for service monitoring must be able to formulate monitoring strategies by comprehensively considering the trade-off between monitoring benefits, resource cost and system overhead. Moreover, the monitoring budget of a system vendor, i.e., the acceptable maximum resource cost for monitoring, is often limited. In order to achieve cost-effectiveness in monitoring, the component services which are critical to the quality of the SBS must be prioritised. Therefore, the identification of component services with high criticality is also an important issue to be addressed.

  Several approaches have been proposed for the identification of critical services in an SBS [3][20][26]. However, those approaches can handle only one quality dimension (e.g., reliability), which is insufficient in real-world scenarios where multiple quality dimensions are involved. In addition, these approaches are not capable of capturing the important characteristic of cloud computing: multi-tenancy, which introduces significant challenges to the formulation of cost-effective monitoring strategies for SBSs. As discussed in Section 1.1, the tenants of a multi-tenant SBS often have different quality preferences [19] for the SBS, and the number of tenants that share different component service often varies. In this context, a component service critical to some tenants may not be of the same criticality to the others. When formulating a monitoring strategy for a multi-tenant SBS, the system engineer must consider the diversity in tenants' differentiated quality preferences and endeavour to reduce the risk of quality violations upon runtime anomalies for all tenants. Therefore, multi-tenancy significantly complicates the identification of critical services and the formulation of monitoring strategies for SBSs.

- **For service adaptation**: In order to fix the runtime anomalies detected, a multi-tenant SBS in the cloud must achieve the ability to adapt at runtime. A number of approaches has been proposed in recent years, such as fault removal [20][27][28] and fault tolerance techniques [29][30][31]. Service re-optimisation, as a representative approach of fault removal, is often used to reduce the impacts of runtime anomalies by selecting alternative candidate services to formulate a new service composition when a system change is detected [32][27][33]. However, this approach often suffers low efficiency in most cases and causes unacceptable business interruptions due to the fact that solving the NP-complete service selection problem satisfactorily is often very time consuming [14], especially in large-scale scenarios. Thus, a quick service re-selection that replaces only the anomalous ones with alternative services with equivalent functionalities is much more practical than an overhaul of the entire SBS. Therefore, how to select the proper services for rapid runtime system adaptation is an important issue to be addressed for the service adaptation of multi-tenant SBSs.

Fault tolerance, by offering service redundancy with functionally equivalent services, is another intuitive and promising technique to guarantee the quality of the SBS in a volatile environment. Compared to service reselection at runtime upon runtime anomalies, fault tolerance can achieve higher performance, e.g., shorter business interruption time, but consume more resources due to the deployment of service redundancy. First, redundant services are selected for the component services from the corresponding group of candidate services with equivalent functionality, then, the selected redundant services run with the corresponding component services in a certain redundancy mode, such as sequence or parallel. In this way, the redundant services can immediately replace the faulty component services at runtime. It is theoretically ideal to prepare redundancy for all component services of the SBS. However, allocating service redundancy for every component service of an SBS is often impractical because the excessive cost may probably exceed a system vendor's limited budget, especially in large-scale scenarios. Therefore, it is much more cost-effective to provide service redundancy for the critical component services in an SBS, which is an important issue in the formulation of fault tolerance strategies for SBSs. Moreover, similar to the monitoring strategies, the fault tolerance strategies must be able to cater for tenants' diverse multi-dimensional quality requirements and preferences as well as their different business scenarios. This is of tremendous importance in the formulation of cost-effective fault tolerance strategies for multi-tenant SBSs.

1.3 Overview of this thesis

In this thesis, we propose a systematic solution to address the issues discussed in Section 1.2, which is named LQM4MTS (Lifetime Quality Management for Multi-Tenant SBSs). The structure of this thesis is shown in Fig. 1.2 and introduced as

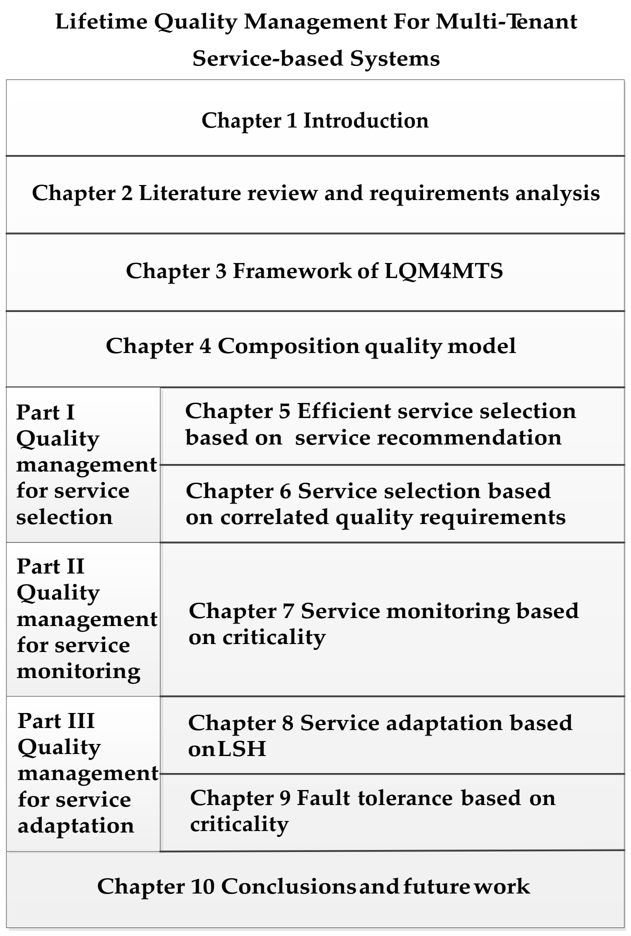

Lifetime Quality Management For Multi-Tenant Service-based Systems
Chapter 1 Introduction
Chapter 2 Literature review and requirements analysis
Chapter 3 Framework of LQM4MTS
Chapter 4 Composition quality model
Part I Quality management for service selection
Chapter 5 Efficient service selection based on service recommendation
Chapter 6 Service selection based on correlated quality requirements
Part II Quality management for service monitoring
Chapter 7 Service monitoring based on criticality
Part III Quality management for service adaptation
Chapter 8 Service adaptation based on LSH
Chapter 9 Fault tolerance based on criticality
Chapter 10 Conclusions and future work

Figure 1.2: The structure of this thesis.

follows: In Chapter 2, the literatures related to the quality management for each lifetime stage of an SBS is reviewed respectively. Then, the requirements for the solution to the research problem are analysed based on a motivating example.

In Chapter 3, we introduce the framework of LQM4MTS, which involves a set of techniques that addresses the above-mentioned issues in each lifetime stage of multi-tenant SBSs.

In Chapter 4, we present the composition quality model for multi-tenant SBSs, which is employed throughout the lifetime quality management for multi-tenant SBSs.

Then we organise the proposed set of approaches in three parts: Part I (Chapter 5 and Chapter 6), Part II (Chapter 7) and Part III (Chapter 8 and Chapter 9), which introduce the approaches for quality management at each lifetime stage of multi-tenant SBSs, namely service selection, service monitoring and service adaptation respectively.

In Chapter 5, we present a novel approach for build-time service selection using service recommendation based on clustering techniques [34], which innovatively explores the similarity between tenants' quality requirements and services' quality values to realise effective service selection for multi-tenant SBSs with high efficiency. We firstly introduce the procedure of recommending representative services for selection based on tenant clustering and service clustering, based on which, the model for service selection is presented. Finally, we evaluate the effectiveness and efficiency of the proposed service selection approach based on some experimental results.

In Chapter 6, we propose an innovative approach for service selection based on tenants' correlated quality requirements. Firstly, the scenario where different dimensions of tenants' quality requirements are correlated is introduced in detail. Then, the procedure of formalising the quality correlations in tenants' requirements is presented, based on which, the models for service selection are proposed. Finally, we conduct extensive experiments to evaluate the proposed approach in terms of effectiveness and efficiency.

In Chapter 7, we describe a novel approach for formulating cost-effective monitoring strategies based on criticality for multi-tenant SBSs. Firstly, we introduce the calculation of service criticalities of the component services in an SBS, based on which, a model is proposed to formulate cost-effective monitoring strategies by comprehensively considering the monitoring benefits, resource cost and incurred system overhead. Finally, the effectiveness especially cost-effectiveness and efficiency of the proposed approach is assessed based on extensive experiments.

In Chapter 8, we propose a new approach for runtime service adaptation based on Locality Sensitive Hashing (LSH), which can achieve quick service replacement for adaptation upon runtime anomalies. The principles of LSH are firstly introduced as a preliminary, and then the procedure of the proposed LSH-based service adaptation is illustrated. Finally, a series of experiments is conducted to evaluate the effectiveness and efficiency of the proposed approach.

In Chapter 9, we present an innovative approach for formulating cost-effective fault tolerance strategies based on criticality for multi-tenant SBSs. Firstly the calculation of service criticality is briefly introduced, which is the same as that in Chapter 7. Then we describe the procedure of formulating fault tolerance strategies in detail. Finally, we evaluate the proposed approach in terms of effectiveness especially cost-effectiveness and efficiency with some experimental results.

Finally, in Chapter 10, we conclude this thesis, summarise the major contributions of this research and outline the future research directions.

Chapter 2

Literature review and requirements analysis

In this chapter, the existing work related to our research is introduced. We conduct an extensive and comprehensive literature review on quality management for SBSs. We present the state-of-the-art literatures and analyse their limitations, based on which, the research requirements for lifetime quality management for multi-tenant SBSs is proposed.

This chapter is organised as follows: Section 2.1 reviews the related work on different lifetime stages of quality management for SBSs respectively, namely service selection, service monitoring and service adaptation. Then, with a motivating example, Section 2.2.2 analyses the research requirements for lifetime quality management for multi-tenant SBSs. Finally, Section 2.3 summaries this chapter.

2.1 Literature review

2.1.1 Quality management for service selection

2.1.1.1 Service selection models and algorithms

Back to a decade ago, the authors in [20] present AgFlow, a middleware platform for Web service composition, which can maximise client satisfaction with utility over multiple quality dimensions, while meeting clients' multi-dimensional quality requirements. Two selection approaches are investigated: local selection and global selection. Integer Programming (IP) is used to find the solution to the constraint optimisation problem (COP). When anomalies occur at runtime in an SBS built with global selection, re-planning procedure may be triggered to ensure that the end-to-end quality remains optimal. The work in [21] proposes an architecture in which quality-aware service selection is modelled respectively as a 0-1 knapsack problem (MMKP) in the combinatorial model and a multi-constrained optimal path (MCOP) problem in the graph model. Heuristic algorithms are adopted to find near optimal solutions for different composition structures in polynomial complexity time. The work in [35] proposes a heuristic approach to find the close-to-optimal solution for service selection by decomposing global quality constraints into local quality constraints, which aims at applications with dynamic changes and real-time requirements. In [13], the authors present CASS, a model that selects services based on iterative multi-attribute combinatorial auction. The complementarities between services are taken into account. The authors in [36] propose an approach for skyline discovery and composition of multi-cloud mashup services. MapReduce paradigm is used to solve the skyline selection problem on mashup cloud platforms, which is accelerated by exploring distributed parallelism. The mashup composition process is optimised based on the MapReduce skyline selection guided by the quality constraints. The authors in [37] propose an approach to compute a quality-optimised selection of service clusters that includes a sufficient number of backup services for each service employed. The backup services should be sufficiently distributed to prevent a task failure. The possible repair costs related to a service in case of failure are taken into consideration in a multi-objective approach for service selection. In [38], the authors propose an approach for service selection based on service clusters. A model is proposed to solve the optimal quality problem of Web Service composition using dynamic programming techniques. In [39] the authors propose an approach for service selection for services with probabilistic quality. Services' quality values are represented as discrete random variables with probability mass functions.

However, the common and critical limitation of these existing approaches when applied in cloud computing is that they only support single-tenant SBS - they try to optimise the quality for only one end user. These approaches can be adopted to create service compositions for multiple tenants one after another. The result is that, although the created service compositions can locally fulfil the quality constraints of corresponding tenants, the overall quality of the SBS is usually sub-optimal, i.e., the optimisation goal of service selection for the SBS cannot be achieved. Furthermore, applying these approaches to compose multi-tenant SBSs is very computationally expensive in large-scale scenarios. To address this issue, in work [22], the authors propose an approach that facilitates quality-aware service selection for multi-tenant SaaS by modelling the problem as a constraints optimisation problem, and again, IP is used to find an optimal solution. In large-scale scenarios where the SaaS optimisation problem is computationally expensive, a greedy algorithm is proposed to find a near-optimal solution. This work does not consider the diversity and similarity between tenants' quality requirements and quality values of the services, which can be used to recommend suitable services for service composition to fulfil the tenants' diverse quality requirements. By doing so, the effectiveness of service selection for multi-tenant SBSs can be guaranteed and its efficiency can be further improved.

Quality correlations in service selection has received considerable attention and made a lot of progress. The correlations between different services have been well investigated in service selection by many works [13][40][41]. This type of quality correlation means that some quality dimensions of a service are not only dependent on the service itself but also correlated to other services [40]. The authors in [40] propose CASP, which is a correlation-aware service pruning method for service selection. CASP manages quality correlations by accounting for all services that may be integrated into optimal composite services and prunes services that are not the optimal candidate services. The work in [42] proposes an approach for computing the composite service skyline in the presence of quality correlations. Pruning techniques are used to accelerate the computation. The authors in [43] propose an approach for quality dependency-aware service composition considering multiple quality attributes. Pareto-based techniques are combined with Vector Ordinal Optimisation techniques to search for Pareto optimal solutions. A candidate pruning algorithm is proposed to remove the unpromising candidate services. The authors in [44] propose an approach for quality-aware Web Service selection with internal complementarity (WSS-IC). This problem is handled by an iteratively improving framework for deriving the solution iteration by iteration while taking into consideration both solution structure and quality constraints. In each iteration, the current solution is improved by solving a disjunctively constrained knapsack problem. The work in [45] proposes a cloud service composition framework that selects the optimal composition based on an end user's long-term quality requirements. Multivariate quality analysis method is used to predict the long-term quality provisions based on service providers' historical quality data and short-term advertisements represented using time series. Quality dimensions' intra correlations and quality time series' inter correlations are taken into consideration in the multivariate analysis and the selection of optimal service composition. In [41], the authors present an alliance-aware service composition method, which takes into consideration the Alliance Relation

(AR) between services when formulating service composition. The fundamental properties of the AR are given based on a multi-granularity service composition model, and then alliance relation granularity is coarsened into a relation granulation quotient space and the domain elements are matched reversely with service composition. A relation Granularity-aware Particle Swarm Optimisation Algorithm (RG-PSO) is used to solve the alliance-aware service composition optimisation problem. However, none of the above works and the like take into consideration the correlation between different quality dimensions of a tenant's quality requirements for a multi-tenant SBS.

Fuzzy logic can be used to handle users' correlated and imprecise quality requirements. For example, in [46], the authors model the service selection as a fuzzy constraint satisfaction problem. The constraint levels on each quality dimension are represented with multiple fuzzy sets. The authors in [47] present a fuzzy model for ranking real-world Web services. A ranking algorithm is proposed, which is based on the objective weighting technique that leverages the distance correlation metrics between quality dimensions. The authors in [48] propose an approach for service selection based on fuzzy logic that considers users' personalised trade-off strategies. However, these service selection and ranking approaches based on fuzzy logic are used in single-tenant environment. They mostly assume that the quality requirements can be represented with a set of fuzzy expressions based on users' quality trade-off preferences, which is insufficient to handle the dynamically varied and correlated quality requirements.

2.1.1.2 Service filtering and recommendation

Service recommendation and filtering, as an effective way for improving the effectiveness and efficiency of service selection, has been discussed in a number of research investigations.

In [49], the authors apply skyline techniques to quality-aware service composition to reduce the number of candidate services. If the size of skyline is still large, representative services are then selected after a clustering and ranking process. The work in [50] presents a statistical clustering approach that supports retrieving Web services to match a given natural language-based enquiry. Web service clusters are calculated based on the proximity between WSDL documents, which is measured with Euclidean distance extended by means of a multi-dimensional angle produced by a vector space search engine. In [51], the authors propose the ranking and clustering of Web services based on the notion of dominance, which is obtained by comparing the degree of matching in all parameters and according to all criteria between services. The produced service clusters reveal and reflect the different trade-offs between the matched parameters. In [52] the authors propose a quality-driven component ranking framework named CloudRank for cloud computing by taking advantage of the past component usage experiences of different component users, which facilitates the optimal component selection from a set of functionally equivalent component candidates. A greedy method is used to compute the component ranking. In [53], the authors present FTCloud, a component ranking framework for fault-tolerant cloud applications. Algorithms based on system structure information and prior knowledge are employed to identify and rank the significant components in a cloud application. Significant value is used to indicate the importance of a component service. In [23], the authors combine user-based and item-based collaborative filtering algorithms to facilitate Web service recommendation. In [54], the authors present AWSR, a Web service recommendation system based on users' usage history to actively recommend Web services to users. A hybrid new metric of similarity is developed to combine functional similarity measurement and non-functional similarity measurement based on comprehensive quality of Web services. In [55], the authors propose a semantic content-based recommendation approach that analyses the context of intended service use to provide recommendations in conditions of scarce user feedback. In [24], the authors present LoRec, a system that supports service recommendation by predicting Web service quality values based on user locations and historical Web service quality records. Users and services are clustered respectively according to the user similarity and service similarity. The authors in [56] propose a Web service recommendation approach that incorporates a user's potential quality preferences and diversity feature of user interests on Web services, which are mined by exploring the Web service usage history. Then the scores of candidate services are calculated by measuring their relevance with historical and potential user interests, and their quality utility. A diversity-aware Web service ranking algorithm is proposed to rank the candidate services based on their scores and diversity degrees derived from a Web service graph, which is constructed based on the functional similarity between services. The authors in [57] propose a temporal tag-based and social-based (TTS) service recommendation algorithm, which is a hybrid method by considering tag, time and users' social relations information at the same time for service recommendation. The work in [58] presents a location-aware personalised CF method for service recommendation, which leverages both locations of users and services when selecting similar neighbours for the target user or service. The personalised influence of users and services are taken into account in the similarity measurement. In [59], the authors propose a probabilistic approach to predict the popularity of services to facilitate service recommendation performance. A method is presented that extracts service evolution patterns by exploiting Latent Dirichlet Allocation (LDA) and time series prediction. A time-aware service recommendation framework is established for mashup creation that conducts joint analysis of temporal information, content description and historical mashup-service usage in an evolving service ecosystem.

However, the above service filtering and recommendation approaches are proposed mainly in the single-tenant environment, where multiple tenants' diverse multi-dimensional quality requirements on the same SBS are not considered. In addition, among the approaches of service recommendation with similarity measurement between the users' quality requirements and services' quality values, most of them assume that the quality requirements to a single service are known and do not take the end-to-end quality requirements into account. Thus, the quality requirements for the entire SBS need to be explored further in service recommendation to improve the effectiveness and efficiency of service selection, especially in the multi-tenant environment.

2.1.2 Quality management for service monitoring

After service selection, the component services of an SBS must be monitored to facilitate timely adaptation upon anomalies to guarantee the quality of the SBS. Monitoring is the basis of most work in service adaptation, such as [27][60][61].

Many efforts have been devoted to the monitoring of SBSs. The authors in [62] propose ReqMon, a two-level monitoring system that consists of distributed individual monitor servers and a centralised global integrative monitor. The authors in [63] propose a framework to verify the requirements for service compositions at runtime, which supports monitoring the runtime behaviours of component services by intercepting the events exchanged in the SBS. The work in [64] presents an approach for runtime monitoring of WS-BPEL processes, which can weave the external monitoring rules into the service composition. In [65], the authors propose an assertion language named ALBERT, with which both the functional and non-functional properties of service compositions can be specified. Dynamo [66], a proxy-based monitoring infrastructure is used at runtime to check the assertions. The authors in [67] propose an architecture named Astro aiming at separating the business logic of a Web service from its monitoring functionality. It can monitor both a single instance and multiple instances of a BPEL process in a class. A language is also defined to specify the properties to be monitored. The authors in [68] propose an integrated approach for BPEL monitoring based on Dynamo [66] and Astro [67]. The integration happens both for the monitoring languages used and the monitoring frameworks. In [69], the authors propose a general monitoring language named SECMOL based on three existing monitoring languages, namely ECAssertion [70], SLANG [71] and WSCoL [64]. In SECMOL, data collection, data computation, and data analysis are considered separately. In [72] the authors employ timed automata to monitor the timeliness, reliability and throughput of the system according to the SLA. The work in [1] adopts WSCoL as a means to enhance service compositions with monitoring capabilities.

However, none of the existing work has properly considered the cost effectiveness of monitoring strategies. Most of them treat all component services in an SBS equally and constantly monitor all services whilst cost-effectiveness is not sufficiently considered when formulating monitoring strategies. The trade-off between monitoring benefit, resource cost and system overhead should be taken into consideration and the critical services which impact the quality of an SBS more seriously upon anomalies must be given higher priorities in the formulation of cost-effective monitoring strategies.

In order to identify the critical services in an SBS, some approaches have been studied. Recent years, in the area of software engineering, various reliability-based and structure-based importance measures have been proposed [73][74][75], which can be used to identify and rank the important components in a software system. These works have shed light on the issue of important service identification in the SBSs. In [20] the authors use the critical path in terms of execution duration to identify the critical services in a service composition. In [26] the authors present FTCloud, a component ranking framework for fault-tolerant cloud applications. Algorithms based on system structure information and prior knowledge are employed to identify and rank the significant components in a cloud application. In [3] and [76], the authors propose CriMon, an approach that evaluates the criticalities of both execution paths and component services based on the concept of probabilistic critical path. The trade-off between monitoring benefit, resource cost and system overhead is described by the Value of Monitoring (VoM), which is integrated into the model for formulating cost-effective monitoring strategies.

However, the existing approaches mostly assess service criticality from one dimension of quality, such as failure rate or response time, which is not suitable for the cloud environment characterised by multiple tenants with preferences for multiple dimensional quality.

2.1.3 Quality management for service adaptation

2.1.3.1 Service re-selection at runtime

The authors of [2] proposed QoSMOS, a tool-supported framework for the development of adaptive service-based systems. QoSMOS first translates high-level quality requirements for the SBS to probabilistic temporal logic formulae, which are then analysed to identify and enforce optimal system configurations. Three mapping patterns of abstract to candidate services are studied: single, sequential one-to-many, and parallel one-to-many mapping. In [20], the middleware platform AgFlow is designed with the ability of adaptation. When exceptions occur at runtime in an execution plan built with global selection, replanning procedure may be triggered to ensure that the end-to-end quality remains optimal. In [27], the authors present an optimisation approach for the composition of Web services, which can meet the local and global quality constraints of the users. Service selection problem is formalised as a Mixed Integer Programming (MIP) problem, and adaptive re-optimisation is adopted to fulfil the variable quality constraints at runtime. When a feasible solution does not exist, negotiation techniques are exploited to enlarge the solution domain. The work in [28] proposes a decentralised self-adaptation mechanism using market-based heuristics for service-based applications in the cloud. Continuous double-auction is used to select services for composition to meet the changing quality requirements. The authors in [60] propose a framework named MOSES that supports quality-driven adaptation of a service-oriented system at runtime. MOSES is able to select and implement adaptation actions based on a combination of both the service selection and coordination pattern selection mechanisms. The optimal service adaptation problem is formulated as a Linear Programming (LP) problem and the computational cost is investigated. However, the existing works mainly focus on single-tenant environment and the computational complexity may be high due to the process of finding optimal solution and negotiation, and hence are not suitable for the runtime adaptation of multi-tenant SBSs.

2.1.3.2 Fault tolerance based on service redundancy

In order to improve the effectiveness and efficiency in fixing runtime anomalies in SBSs, some researchers have looked into the use of service redundancy for fault tolerance to mitigate the impact of service anomalies. Inspired by the success of hardware redundancy for tolerating hardware failures, software redundancy by using multiple versions of independently developed software has become a widely accepted means to improve the reliability and availability of a software system [77]. Service redundancy usually uses services with similar or identical interfaces as redundant replicas aiming at fault tolerance and performance improvement of SBSs. In [30], the authors present a distributed replication strategy evaluation and selection framework for fault tolerant Web services. Time redundancy, space redundancy and the combinations of the two are studied. Similar strategies are used in [60] for service adaptation to obtain quality levels that could not be achievable by single service. The work presented in [78] investigates the inherent redundancy and diversity of services, base on which, the authors propose solutions to improve the dependability of SBSs and two invocation strategies of redundant services are employed: sequential and simultaneous. In [79], a framework is proposed for selecting the optimal fault tolerance strategy for an SBS, which is modelled as an optimisation problem with user requirements as local and global constraints, and a heuristic algorithm is used to find the solution. These works and the like all assume that all the services are equally important to the service providers, and thus probably make them overpay for the services which are not critical, or cannot mitigate the system failures effectively because of the absence of protection to critical services under a budget constraint.

2.2 Motivating example and requirements analysis

After reviewing the state-of-the-art for quality management at different lifetime stages of SBSs, we can see that some critical issues remain unresolved in the scenario of multi-tenant SBSs, which are challenging yet fundamental for the wide application of multi-tenant SBSs in the cloud. Existing approaches are insufficient in aspects such as efficiency, multi-tenancy support, cost-effectiveness, etc. Thus, we advocate that a systematic solution should be provided, which can effectively and efficiently manage the quality of multi-tenant SBSs across their whole lifetime.

2.2.1 Motivating example

This section presents an example multi-tenant SBS, namely *Versatile Online Video Studio (VOVS)* [80] to motivate this research.

As shown in Fig. 2.1, this SBS consists of nine tasks. Similar to [20][21][27][81],

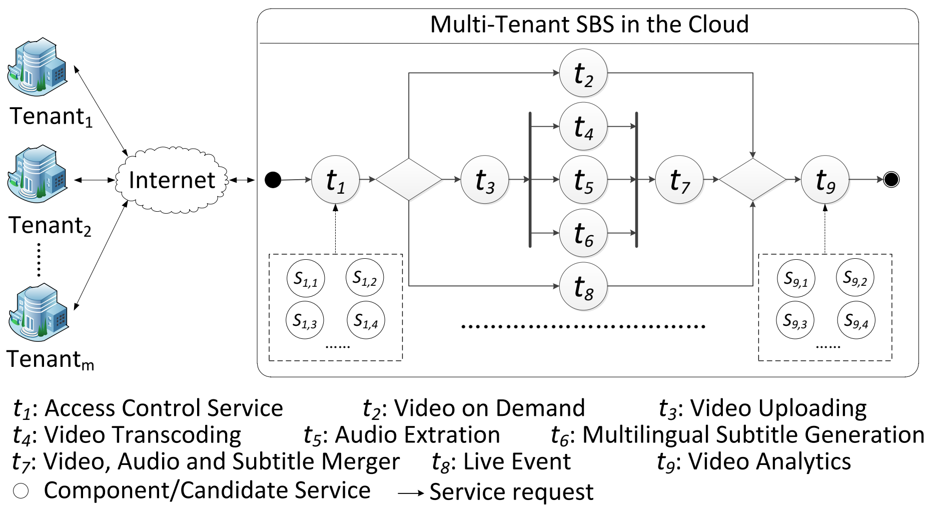

Multi-Tenant SBS in the Cloud
Tenant$_1$
Tenant$_2$
Tenant$_m$
Internet
t_1
t_2
t_3
t_4
t_5
t_6
t_7
t_8
t_9
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,4}$ ......
$s_{9,1}$ $s_{9,2}$ $s_{9,3}$ $s_{9,4}$ ......
t_1: Access Control Service
t_2: Video on Demand
t_3: Video Uploading
t_4: Video Transcoding
t_5: Audio Extration
t_6: Multilingual Subtitle Generation
t_7: Video, Audio and Subtitle Merger
t_8: Live Event
t_9: Video Analytics
○ Component/Candidate Service
→ Service request

Figure 2.1: Business process of *VOVS*.

there is a group of functionally equivalent candidate services with differentiated quality values available for each of the tasks of *VOVS*. The system engineer needs to select component services from the corresponding set of candidate services (referred to as *service class*) for tasks t_1 to t_9 to compose the SBS, which provides three types of online video services to multiple tenants. The notation and acronym summary for this thesis can be found in the appendix.

In *VOVS*, t_1 provides the Access Control Service (ACS) to authenticate tenants when their end users try to gain access to the SBS. After the tenants are authenticated, the end users can use different categories of video services: t_2 is used to view videos on demand (VoD); t_8 offers the live streams services, which are used to broadcast or watch live videos; while t_3 to t_7 provide services for uploading, editing, and publishing the video clips; t_3 hosts the uploaded videos; t_4 transcodes the videos to different formats and resolutions compatible with various end devices; t_5 extracts audio streams from the videos, with which multilingual subtitles are generated by t_6; t_7 merges the videos, audios and subtitles, and publishes the synthetic video clips; t_9 performs video analytics, which carries out measurement and analysis of the videos viewed and uploaded online for user experience optimisation.

This SBS has the following characteristics:

1. In order to reduce the time and cost, the whole or part of the business process can be implemented and deployed by other system vendors in the form of Web services in the cloud. The system engineer selects a component service from the corresponding service class for each task to formulate the SBS in the form of service composition to serve multiple tenants simultaneously.

2. For some tenants, trade-offs may exist between different quality dimensions of their quality requirements. In this way, the tenants' quality requirements are no longer deterministic but becomes dynamically varied and correlated. For example, the cost that a tenant can accept may increase along with the

increase of system throughput. Such quality requirements must be considered in the process of service selection.

3. After the SBS is built, it must process the video stream timely and continuously to ensure low-latency streaming of high quality videos. In the dynamic and volatile cloud environment, various runtime anomalies may happen in the services that constitute the system, such as service unavailability, network failure, etc. If an anomaly cannot be fixed in a timely manner, the tenants sharing the faulty service will suffer from increased latency in the video stream or even denial of service, which can easily lead to customer complaints and attrition. Service monitoring and adaptation are essential for handling anomalies in a dynamic and volatile environment to ensure the quality of its video streaming, such as response time, throughput, etc.

4. The tenants that share the SBS have multi-dimensional quality requirements and preferences, as well as different business scenarios. Firstly, the tenants may contain end users that access the SBS with different kinds of end devices, and thus have different concerns on quality of the SBS. For example, when viewing videos on demand, tenants that offer services for end users using laptops on a home network may prefer high resolution and low latency without worrying about cost incurred by data traffic, while tenants that access the SBS with a mobile network may prefer relatively low resolution to limit the cost incurred by data traffic and be endurable to long buffering time. In addition, the tenants may subscribe to *VOVS* for different purposes, e.g., watching videos on demand or live videos, and thus anomalies of some services may not affect the tenants who do not use them. These need to be taken into account in the quality management at different lifetime stages of the SBS.

For such a multi-tenant SBS, the requirements for the quality management at each lifetime stage are analysed in the following sections.

2.2.2 Requirements analysis

2.2.2.1 Quality management for service selection

In this stage, we focus on ensuring high effectiveness of service selection whilst improving its efficiency, and service selection based on tenants' correlated quality requirements.

Existing approaches for service selection often suffer from high computational overhead by finding the optimal solutions, especially in large-scale scenarios. Service recommendation is a practical way to address this issue. As we have discussed in Section 2.2.1, tenants' quality requirements for the SBS are often characterised by distinctive features, e.g., some prefer higher throughput while others prefer lower cost. The quality values of candidate services, on the other hand, also have the similar features, i.e., the functionally equivalent services (e.g., video transcoding) provide differentiated quality and trade-offs can be found between the multi-dimensional quality values. In such context, an effective and efficient service selection approach using service recommendation should fulfil the requirements as follows:

1. Recommending the "right" candidate services according to the tenants' differentiated quality requirements. For example, a candidate service that can respond service requests quickly should have high potential to participate in the service selection for the tenants that prefer lower response time.

2. Catering for the tenants' varied quality preferences in service recommendation. The quality preferences of the tenants that share the corresponding component service should be taken into consideration when ranking the functionally equivalent candidate services for recommendation.

3. Ensuring high effectiveness of service selection whilst improving its effi-

ciency by reducing the search space of the problem of service selection significantly.

In order to handle the tenants' correlated quality requirements, a successful service selection approach should support the following features:

1. Formalising the correlated quality requirements appropriately, by which, the trade-offs between different quality dimensions of the requirements can be captured.

2. Facilitating effective and efficient service selection to fulfil multiple tenants' correlated quality requirements simultaneously. The optimisation goal in service selection, if any, should also be achieved in the meanwhile.

2.2.2.2 Quality management for service monitoring

In this stage, we focus on the effectiveness especially cost-effectiveness and efficiency of service monitoring.

Service monitoring must be implemented to guarantee the quality of the SBS after it is composed by service selection. As discussed in Section 2.2.1, cost-effectiveness is a critical issue in service monitoring and limited resources should be allocated to those component services which are critical to the quality of the SBS. However, the impacts of different component services on the quality of the SBS upon runtime anomalies are different, and the severity of the impact of an anomalous component service on the SBS is also varied to different tenants due to their different business scenarios [82]. In order to reduce the quality degradation and quality violation caused by runtime anomalies within the system vendor's monitoring budget, an approach for cost-effective monitoring should meet the requirements as follows:

1. Identifying critical component services based on their multi-dimensional quality, the tenants' quality preferences and their service sharing across the SBS.

2. Formulating monitoring strategies by determining the key monitoring parameters for the component services to be monitored such as number of monitors, monitoring frequency, monitoring granularity.

3. Achieving high cost-effectiveness in monitoring the critical component services. The trade-off between monitoring benefit, resource cost and system overhead within the system vendor's monitoring budget should be considered systematically.

2.2.2.3 Quality management for service adaptation

When the runtime anomalies are detected by service monitoring, it is of tremendous importance to fix the anomalies effectively in a timely manner to reduce the risk of quality violation. In this stage, we focus on improving the effectiveness and efficiency of service adaptation.

Re-optimising the entire SBS upon runtime anomalies is potentially time consuming and is often impractical. Thus, the efficiency of system adaptation must be given a higher priority over the optimality of the system quality. Efficient service adaptation at runtime need to fulfil the following requirements:

1. Efficiently finding appropriate candidate services in terms of quality values for quick replacements of the anomalous component services.

2. If the quality violation cannot be eliminated after service replacements and system re-optimisation is unavoidable, service selection for re-optimisation must still be of high efficiency.

3. After service adaptation, the quality requirements of all tenants for the SBS must still be satisfied.

As another promising way to ensure the quality of the SBS, fault tolerance can alleviate or mitigate impacts of runtime anomalies on the system quality of the SBS by deploying redundant services for all or some component services.

The redundant services coordinate with the corresponding component service with certain redundancy mode, such as sequence and parallel. An approach for formulating cost-effective fault tolerance strategies should support the following requirements:

1. Identifying critical component services in the SBS, which is the same as that in service monitoring.
2. Prioritising the critical component services in the formulation of fault tolerance strategies to achieve cost-effectiveness.
3. Effectively and efficiently reducing the risk of quality degradation and quality violation upon runtime anomalies.

2.3 Summary

In this chapter, the literatures in relation to the research problems have been reviewed intensively. The major issue of existing research is the lack of a systematic solution that supports the quality management for SBSs in a multi-tenant environment at different lifetime stages, namely service selection, service monitoring and service adaptation. Based on a motivating example, the research requirements for addressing this issue have been analysed in detail.

Chapter 3

Framework of LQM4MTS

In this chapter, we introduce the framework of LQM4MTS, the proposed solution for lifetime quality management for multi-tenant SBSs.

3.1 Overview of the framework

LQM4MTS consists of a set of approaches that cover the whole lifetime of a multi-tenant SBS to address the key issues of quality management discussed in Section 1.2 and fulfil the research requirements discussed in Section 2.2.2. Its framework is shown in Fig. 3.1, in which each module represents an approach for the quality management at corresponding lifetime stage of a multi-tenant SBS.

At the stage of service selection, we propose two approaches as follows:

- SSR4MTS (Service Selection based on service Recommendation for Multi-Tenant SBSs): SSR4MTS aims at effective yet (especially) efficient service selection using service recommendation based on clustering techniques [34]. The similarity between tenants' quality requirements and services' quality values are innovatively explored. It firstly builds tenant clusters based on the diversity and similarity in their multi-dimensional quality require-

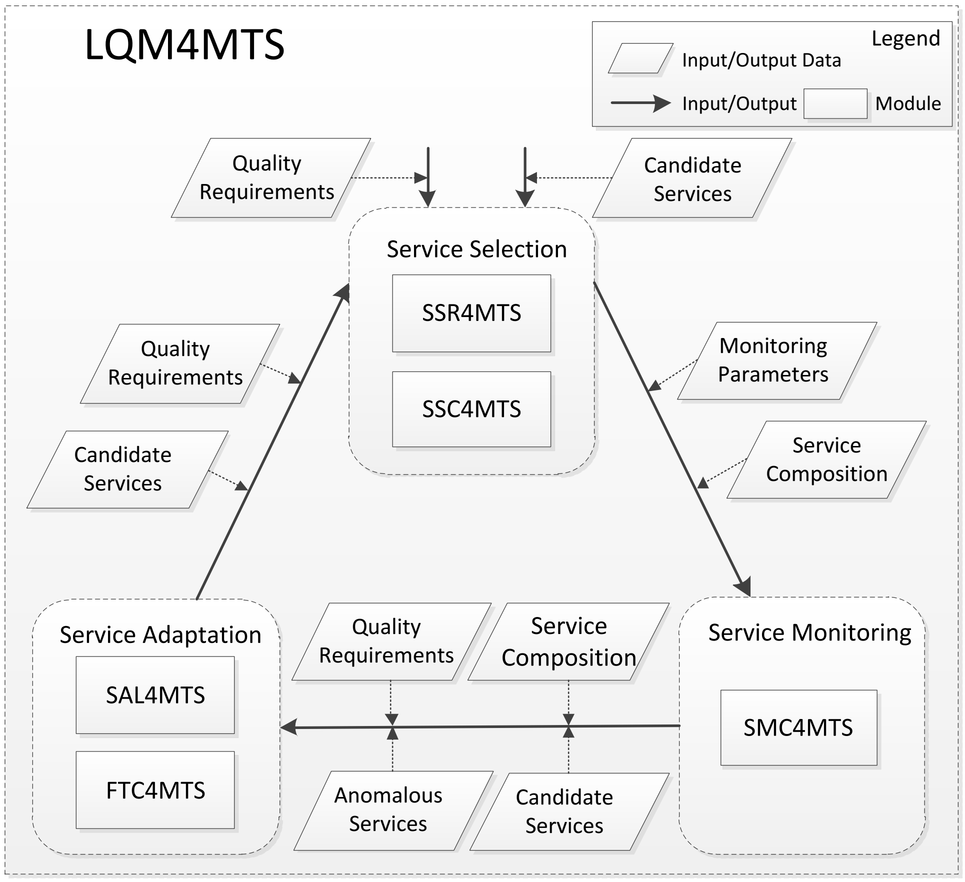


Figure 3.1: Framework of LQM4MTS.

ments. After that, the common quality features of each tenant cluster are extracted, based on which, the candidate services are grouped into different clusters accordingly. By doing so, candidate services with similar quality features to the corresponding tenants' quality requirements can be effectively and efficiently identified and recommended as representatives for service selection. Thus, SSR4MTS is particularly suitable for the scenario where both the tenants' quality requirements and services' quality values have distinctive features.

- SSC4MTS (Service Selection based on Correlated quality requirements for Multi-Tenant SBSs): SSC4MTS is designed to handle the tenants' quality requirements with correlations. It innovatively formalises a tenant's correlated quality requirement with a quality correlation function, which from the perspective of spatial geometry can be represented with a graph, e.g., a (straight or curved) line or a surface, in a Euclidean space. With the available candidate services, the service selection based on a correlated quality requirement is modelled as a Constraint Optimisation Problem (COP), in which the quality correlations are used as quality constraints for service selection.

At the stage of service monitoring, we propose an approach as follows:

- SMC4MTS (Service Monitoring based on Criticality for Multi-Tenant SBSs): SMC4MTS cost-effectively monitors the critical component services in service composition of a multi-tenant SBS. The criticality of a component service is evaluated based on its impacts on the quality of the SBS upon runtime anomalies and the tenants sharing the service. The services with higher criticalities in an SBS are given higher priorities in monitoring. Multiple monitoring parameters and the corresponding monitoring benefit, monitoring resource cost and incurred system overhead are systematically considered in the formulation of cost-effective monitoring strategies.

At the stage of service adaptation, we propose two approaches as follows:

- SAL4MTS (Service Adaptation based on LSH for Multi-Tenant SBSs): SAL4MTS is a runtime service adaptation approach based on Locality Sensitive Hashing (LSH), which, upon a runtime anomaly, rapidly finds the replacement service for the anomalous component service based on the quality similarity between the anomalous service and the corresponding candidate services. The replacement service can then be used to locally adapt the multi-tenant SBS to the runtime anomaly in a timely manner to reduce the adverse impacts on the tenants sharing the anomalous component service. If tenants' quality requirements cannot be satisfied after service replacement, SSR4MTS can be used for service re-selection of the entire SBS.

- FTC4MTS (Fault-Tolerance based on Criticality for Multi-Tenant SBSs): FTC4MTS formulates cost-effective fault tolerance strategies for multi-tenant SBSs by providing redundancy for the critical component services in service composition. The criticality of a component service is the same as that in SMC4MTS. Redundant services for a component service are selected from the corresponding candidate services. A fault tolerance strategy specifies the allocation of redundant services for the critical component services and their redundancy mode, such as sequence or parallel. Based on the formulated fault tolerance strategy, redundant services can replace anomalous component services at runtime to significantly reduce the risk of system quality violation. In contrast to SAL4MTS, FTC4MTS can protect the SBS from runtime anomalies with higher performance but consume more resources due to the deployment of redundant services.

After a multi-tenant SBS is built with service selection based on SSR4MTS and SSC4MTS, it is monitored using SMC4MTS to detect the runtime anomalies to guarantee the quality of the SBS. Once the runtime anomalies are detected, service adaptation based on SAL4MTS or FTC4MTS can be used to adapt the SBS to ensure that the SBS remains operational and available. If re-optimisation of entire SBS is unavoidable upon runtime anomalies, service re-selection process is carried out to build a new SBS based on the candidate services to fulfil the tenants' quality requirements.

3.2 Summary

In this chapter, the framework of LQM4MTS is presented. The proposed approaches in LQM4MTS for the quality management for multi-tenant SBSs at different lifetime stages, namely service selection, monitoring and adaptation, are introduced. The fundamentals and application scenarios of these approaches are also discussed.

Chapter 4

Composition quality model

Quality evaluation of a service composition is the basis for quality management for SBSs. In this chapter, we introduce the composition quality model adopted in this research, including composition pattern, execution path, execution plan, characteristics of numerical quality, and utility evaluation. This chapter is based on my paper [80].

This chapter is organised as follows. Section 4.1 introduces the composition patterns of services. Then Section 4.2 presents the definitions of execution path and execution plan. Section 4.3 introduces the characteristics of the quality dimensions adopted in this thesis. The evaluation of utility is presented in Section 4.4 . Finally, Section 4.5 summarises this chapter.

4.1 Composition pattern

A system engineer selects services from a number of sets of candidate services (also known as service classes) to compose an SBS with certain composition patterns to serve multiple tenants. Composition patterns describe the order in which the services are executed in the service composition of an SBS. Similar to other

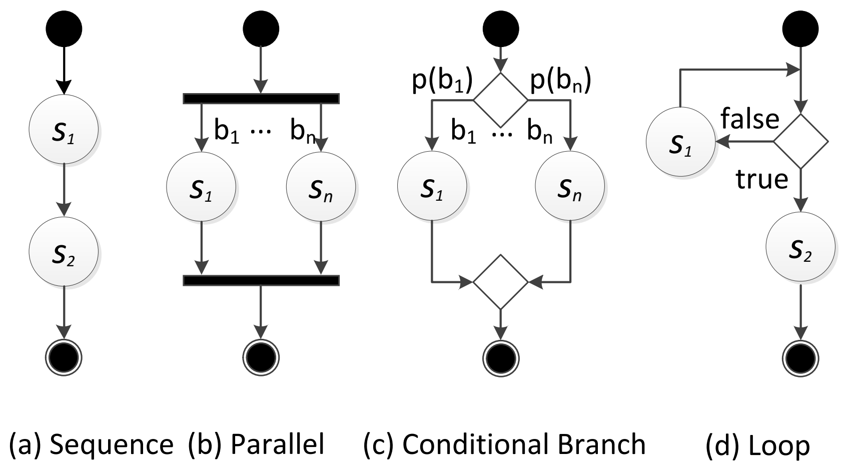


Figure 4.1: Composition patterns.

work [13][20], as Fig. 4.1 shows, four basic composition patterns are included in our model: *Sequence*, *Parallel*, *Conditional Branch* and *Loop*.

1. *Sequence*: The services are executed sequentially.

2. *Parallel*: All the branches $\{b_1, b_2, \ldots, b_n\}$ are executed at the same time.

3. *Conditional Branch*: There is a set of branches $\{b_1, b_2, \ldots, b_n\}$ in this structure, and only one of them can be selected to execute at one time with a probability $p(b_i)(0 \leq p(b_i) \leq 1, \sum_{i=1}^{n} p(b_i) = 1)$.

4. *Loop*: In this structure, every loop iterates for i times with a probability $p(l_i)$, where $0 \leq i \leq MNI$, $\sum_{i=0}^{MNI} p(l_i) = 1$ and MNI is the expected maximum number of iterations for the loop.

$p(b_i)$ in *Conditional Branch* and $p(l_i)$ in *Loop* can be obtained by the system engineer through the analysis of tenants' business requirements as well as the corresponding SLA.

Fig. 4.2 shows an instance of the example SBS *VOVS* presented in Fig. 2.1. Component services s_1 to s_9 have been selected to create the service composition. We can identify three types of composition patterns in *VOVS*: *Sequence*, *Parallel*, and *Conditional Branch*. For example, in response to each service request, s_1, s_2, s_9

are executed in sequence; s_4, s_5 and s_6 are performed in parallel; s_2, s_3 and s_8 are located on conditional branches, only one of which is selected each time with a probability that depends on the distribution of tenants' functional requirements.

In this thesis, we mainly focus on *Sequence*, *Parallel* and *Conditional Branch*. The loop structure is transformed to the conditional branch structure with the loop peeling method described in [13]. Fig. 4.3 shows an example of the loop peeling process where the expected maximum number of iterations is 2. This loop structure is transformed into a conditional branch structure that contains three branches b_1, b_2 and b_3, which are selected to execute with the probabilities of $p(b_1)$, $p(b_2)$ and $p(b_3)$ respectively.

4.2 Execution path and execution plan

Due to the branches in a service composition, execution paths and execution plans can be identified, which are defined as follows:

Definition 4.1. ***Execution Path***. Denoted by *ep*, an execution path is a flattened sequence of services from the initial service to the final service in a service composition. It does not contain conditional branch and parallel structures. A service can belong to multiple execution paths.

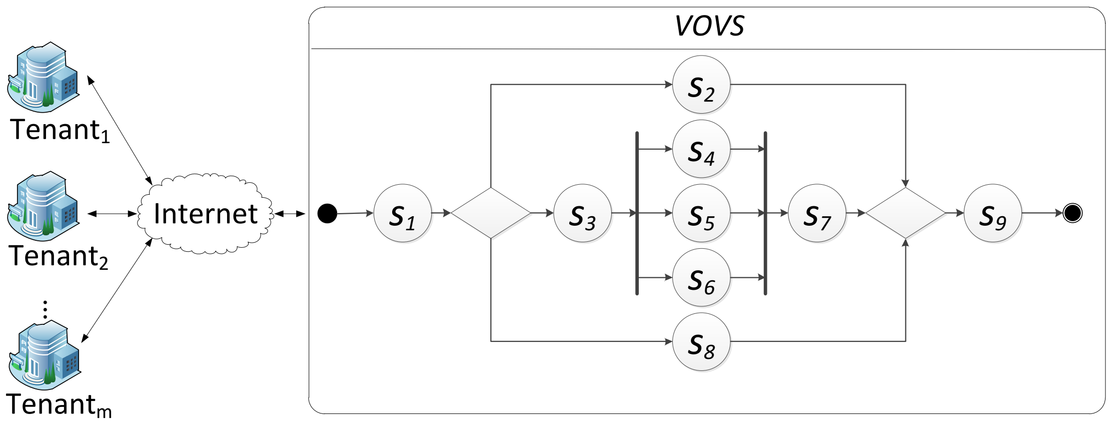


Figure 4.2: An SBS instance of *VOVS*.

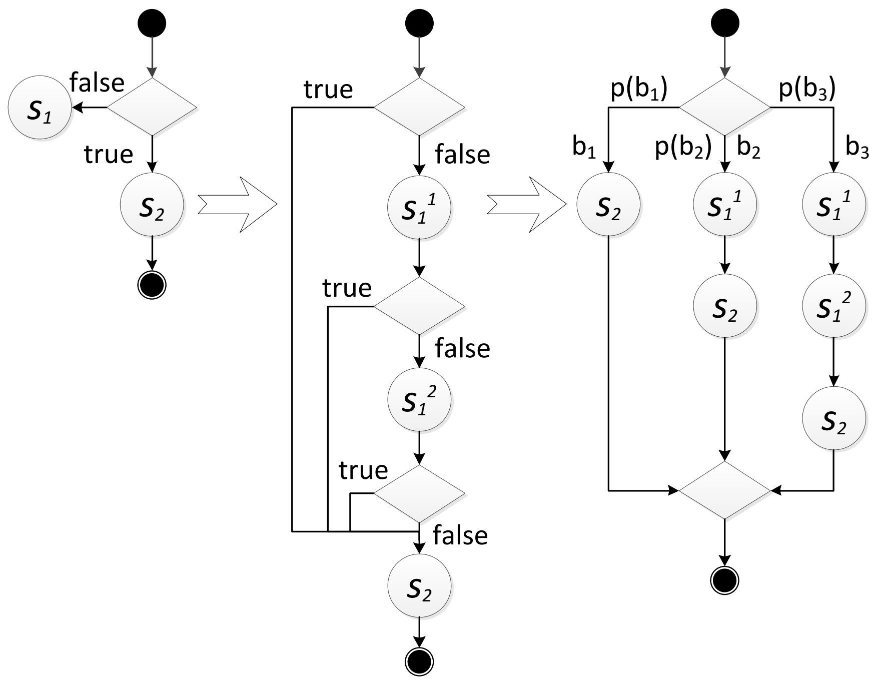


Figure 4.3: A loop peeling example.

Definition 4.2. ***Execution Plan***. Denoted by *epl*, an execution plan is a combination of execution paths with conditional branch or parallel structures, which accomplishes certain functional requirements. It serves multiple tenants in an SBS. A service can also belong to multiple execution plans.

As depicted in Fig. 4.4, five execution paths and three execution plans are identified for the instance of *VOVS* in Fig. 4.2. epl_1 is used for Video on Demand service, epl_2 is for the tenants to upload, edit and publish their video clips, and epl_3 streams live videos.

4.3 Quality characteristics

The quality of an SBS is evaluated based on all its possible execution plans according to their execution probabilities, which is calculated based on the execution probabilities of their execution paths [13]. Table 4.1 shows the aggregation functions for evaluating the overall quality of an SBS $\mathbb{S}$ with four numerical qual-

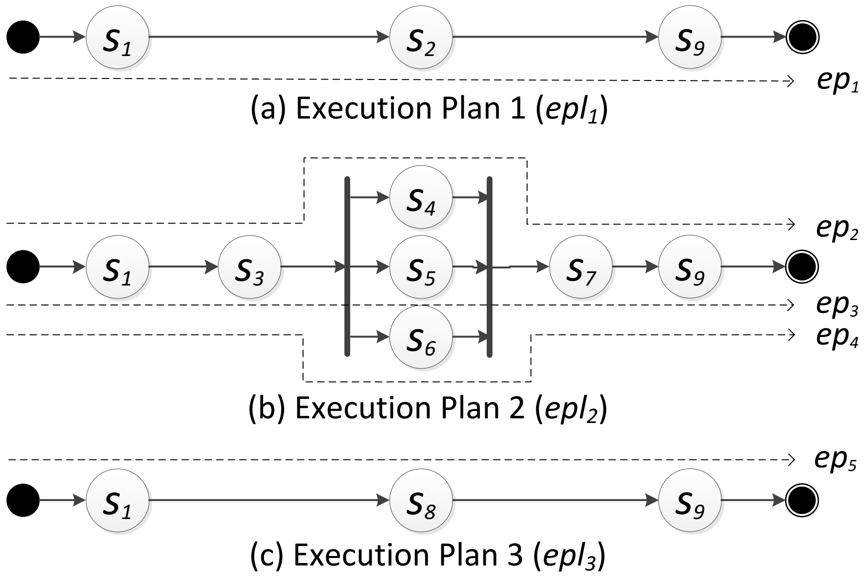

s1
s2
s9
ep1
(a) Execution Plan 1 (epl1)
s4
s1
s3
s5
s7
s9
s6
ep2
ep3
ep4
(b) Execution Plan 2 (epl2)
ep5
s1
s8
s9
(c) Execution Plan 3 (epl3)

Figure 4.4: Execution paths and execution plans of *VOVS*.

ity dimensions: cost, response time, reliability and throughput, which are typical quality dimensions considered by many works [20][21][60]. Their quality values are denoted by q_{ct}, q_{rt}, q_{re} and q_{tp}, respectively, and $s_{i,j}$ represents the j^{th} service in the i^{th} service class. ep_k is the k^{th} execution path and $p(epl_e)$ is the execution probability of the e^{th} execution plan. Take response time of $VOVS$ as an example, it can be calculated by $q_{rt}(VOVS) = p(epl_1)\times q_{rt}(epl_1)+p(epl_2)\times q_{rt}(epl_2)+p(epl_3)\times q_{rt}(epl_3)$, and the response time of each execution plan is determined by its execution path with the longest execution time.

The numerical quality dimensions in Table 4.1 can be divided into two categories: positive and negative, defined as follows:

Definition 4.3. ***Positive Quality Dimension***. If the quality evaluation increases along with the increase of quality value, the quality dimension is regarded as positive, such as throughput and reliability.

Definition 4.4. ***Negative Quality Dimension***. A quality dimension is negative if its evaluation will decrease as its value increases, such as cost and response time.

The model and experiments in this thesis are mainly based on the quality dimensions presented in Table 4.1. The quality dimensions introduced in other literatures can be included as added dimensions in our model.

Table 4.1: Quality Aggregation Functions

Quality Dimension	Aggregation Function
Cost	$q_{ct}(\$) = \sum_{\substack{s_{i,j} \in epl_e \\ epl_e \in \$}} p(epl_e) \times q_{ct}(s_{i,j})$
Response Time	$q_{rt}(epl_e) = \max_{ep_k \in epl_e} (\sum_{s_{i,j} \in ep_k} q_{rt}(s_{i,j}))$ $q_{rt}(\$) = \sum_{epl_e \in \$} p(epl_e) \times q_{rt}(epl_e)$
Reliability	$q_{re}(\$) = \prod_{s_{i,j} \in \$} q_{re}(s_{i,j})$
Throughput	$q_{tp}(epl_e) = \min_{ep_k \in epl_e} (\min_{s_{i,j} \in ep_k} q_{tp}(s_{i,j}))$ $q_{tp}(\$) = \sum_{epl_e \in \$} p(epl_e) \times q_{tp}(epl_e)$

4.4 Utility evaluation

Services in the same service class are functionally equivalent but usually different in their multi-dimensional quality values. As shown in Fig. 2.1, the Video on Demand service in *VOVS* provided by one service provider may have better performance as well as higher price than a service with the same functionality provided by another provider. Which service is better in service selection depends on the quality preferences of the tenants sharing the service. In this thesis, we rank the services in a service class based on their *utility*, which is defined as follows:

Definition 4.5. ***Service Utility***. The utility of a service indicates tenants' generalised preferences for the service. It is calculated based on the quality values of the service and the tenants' quality preferences by applying the Simple Additive Weighting (SAW) technique [83].

Different quality dimensions can be adopted in the calculation of service utility, depending on the scenarios.

In order to remove the incomparability between the units of measurement for different quality dimensions, the original numerical quality values are normalised first using (4.1) for positive quality dimensions and (4.2) for negative quality dimensions with widely used Min-Max normalisation technique [13][20]:

$$Q_p(s_{i,j}) = \begin{cases} \dfrac{q_p(s_{i,j}) - q_p^{min}(sc_i)}{q_p^{max}(sc_i) - q_p^{min}(sc_i)} & \text{if } q_p^{max}(sc_i) \neq q_p^{min}(sc_i) \\ 1 & \text{if } q_p^{max}(sc_i) = q_p^{min}(sc_i) \end{cases} \tag{4.1}$$

$$Q_p(s_{i,j}) = \begin{cases} \dfrac{q_p^{max}(sc_i) - q_p(s_{i,j})}{q_p^{max}(sc_i) - q_p^{min}(sc_i)} & \text{if } q_p^{max}(sc_i) \neq q_p^{min}(sc_i) \\ 1 & \text{if } q_p^{max}(sc_i) = q_p^{min}(sc_i) \end{cases} \tag{4.2}$$

where $q_p^{max}(sc_i)$ and $q_p^{min}(sc_i)$ are the maximum and minimum values, respectively, for the p^{th} quality dimension in the i^{th} service class; $q_p(s_{i,j})$ is the p^{th} dimensional quality value of the j^{th} service in the i^{th} service class; and $Q_p(s_{i,j})$ is the normalised p^{th} quality value of service $s_{i,j}$. In this thesis, a specific quality dimension can be indicated by both "p^{th}" and the abbreviations of their names, e.g., "rt" for response time, "tp" for throughput, etc.

Similar to [22][19], for a multi-tenant SBS, we use (4.3) to calculate the average preference for each quality dimension of a service across all the tenants:

$$w_p^{ave} = \frac{1}{\tau} \times \sum_{t=1}^{\tau} w_{t,p}, p = 1, \ldots, d \tag{4.3}$$

where τ is the number of tenants that share a given service, and a system engineer can usually obtain tenants' quality preferences from their SLAs. Parameter $w_{t,p}$ indicates the t^{th} tenant's preference for the p^{th} quality dimension, $\sum_{p=1}^{d} w_{t,p} = 1$ where d is the number of quality dimensions.

Then, the utility of a given service $s_{i,j}$ with d quality dimensions across τ tenants is calculated by (4.4):

$$u(s_{i,j}) = \sum_{p=1}^{d} w_p^{ave} \times Q_p(s_{i,j}) \tag{4.4}$$

The utility of an execution plan epl_e can be obtained by (4.5):

$$u(epl_e) = \sum_{s_{i,j} \in epl_e} u(s_{i,j}) \tag{4.5}$$

Finally, the utility of an SBS $\$$ can be calculated by (4.6):

$$u(\$)) = \sum_{epl_e \in \$} p(epl_e) \times u(epl_e) \tag{4.6}$$

where $p(epl_e)$ is the execution probabilities of epl_e. In general, an SBS with higher utility is more preferable by the tenants.

4.5 Summary

In this chapter, the composition quality model used in this research is presented, which is the foundation of lifetime quality management for multi-tenant SBSs. In the following chapters, the quality evaluation in different approaches are all based on this quality model.

Part I

Quality management for service selection

Chapter 5

Service selection based on service recommendation

As discussed in Section 1.2, service selection for multi-tenant SBSs is the process of selecting from a number of candidate services to create service compositions to fulfil the tenants' differentiated quality requirements and achieving the optimisation goals in the meantime, such as the minimised cost, maximised throughput, etc. It is a Constraint Optimisation Problem (COP) and NP-Complete [14], which can be computationally expensive to find an optimal solution especially in large-scale scenarios. Thus, it is of tremendous importance to improve the efficiency of service selection for multi-tenant SBSs whilst ensuring its high effectiveness. In this chapter, we introduce SSR4MTS (Service Selection based on service Recommendation for Multi-Tenant SBSs), which supports effective and efficient service selection using service recommendation based on clustering techniques. Tenants and candidate services are clustered based on the quality requirements and quality values respectively, and the similarity between them is explored, by which, the appropriate candidate services in terms of quality values can be recommended for the service composition to achieve high effectiveness yet especially high efficiency in service selection. This chapter is based on my paper [19][81].

This chapter is organised as follows. Section 5.1 introduces the SSR4MTS approach. Then Section 5.2 presents the experimental evaluation of SSR4MTS in terms of effectiveness and efficiency. Section 5.3 discusses how SSR4MTS can be used for service adaptation at runtime and how SSR4MTS can be enhanced by adopting alternative clustering algorithms. Finally, Section 5.4 summarises this chapter.

5.1 SSR4MTS approach

As discussed in Section 2.2.2, tenants' quality requirements for a multi-tenant SBS are often characterised by distinctive features, based on which, the tenants can be partitioned into different groups, i.e., clusters. Take the SBS *VOVS* introduced in Section 2.2.1 for example, some tenants that use the Video on Demand (VoD) service may require minimised video buffering time in spite of higher cost, whilst others can accept slow response time in order to reduce the overall cost. In this case, the tenants can be grouped into two clusters based on their quality requirements. On the other hand, the functionally equivalent candidate services (e.g., the video transcoding services in *VOVS*) often provide differentiated multi-dimensional quality values and trade-offs can be found between them. For example, a candidate service for video transcoding service in *VOVS* can respond quickly to tenants' service requests at a higher price, while another candidate service with same functionality may show a slower response at a more affordable price. Based on this, the functionally equivalent candidate services in each service class can also be categorised into different clusters.

According to the principles of clustering, tenants are similar in the same cluster but different from those in other clusters in terms of quality requirements, and so do the services with respect to the quality values. Based on such similarity and diversity, we are able to establish the connections between tenant clusters and ser-

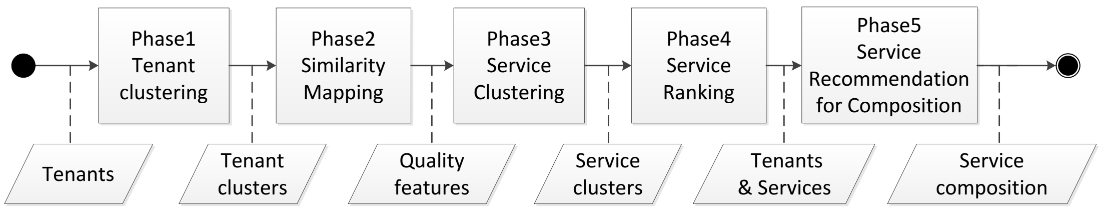


Figure 5.1: The procedure of SSR4MTS.

vice clusters. For example, the services with lower prices potentially are preferred by the tenants that have stringent demand on the lower cost. SSR4MTS is to recommend suitable services for the composition of a multi-tenant SBS by exploring the similarity between tenants' quality requirements and services' quality values.

As shown in Fig. 5.1, SSR4MTS is a five-phase process. We first calculate the similarity between tenants in phase 1 based on their quality requirements and partition them into different tenant clusters. After that, in phase 2, the feature of quality requirements in each tenant cluster is identified and mapped onto the quality space of services. Then, in phase 3, we cluster the candidate services in each service class according to the quality features mapped from tenant clusters. In phase 4, candidate services in each service cluster are ranked according to their utility. Finally, in phase 5, the services with high utilities in each cluster are recommended as representatives to build the search space of service selection, and the solution of service composition is found with Mixed Integer Programming (MIP) technology. These five phases are detailed one by one in this section.

5.1.1 Phase 1: Tenant clustering

In this phase, with the help of clustering technique in data mining [34], the tenants are partitioned into different clusters based on the similarity and diversity in their quality requirements. Quality values are often measured on different scales. We need to normalise them to the same range. In our approach, we use widely used cost and response time as two example quality dimensions. The tenants' quality requirements with these two quality dimensions are first normalised using (5.1) with widely used Min-Max normalisation technique [13]:

$$rn_p(t_e)=\begin{cases}\frac{r_p^{max}(T)-r_p(t_e)}{r_p^{max}(T)-r_p^{min}(T)} & \text{if } r_p^{max}(T)\neq r_p^{min}(T)\\ 1 & \text{if } r_p^{max}(T)=r_p^{min}(T)\end{cases} \tag{5.1}$$

where $r_p^{max}(T)$ and $r_p^{min}(T)$ are the maximum and minimum values, respectively, for the p^{th} quality requirements of all tenants, and $r_p(t_e)$ is the value of the p^{th} quality requirement of the e^{th} tenant, while $rn_p(t_e)$ is the normalised value.

Then, we use K-Means, a popular algorithm for clustering, to build a list of tenant clusters $TC = \{tc_0, tc_1, ..., tc_k\}$ according to the tenants' normalised quality requirements. K-Means is a lightweight algorithm that can efficiently cluster tenants according to the characteristics of their quality requirements [84][85]. The selection of k value in K-Means algorithm is domain-specific and can be determined by the system engineers empirically on a case-by-case basis. In addition, various approaches have been proposed for optimal k value selection [86], however, we do not discuss this in detail in this thesis. Because tenants usually have specific optimisation goals for the system quality [20][22], as an example throughout this chapter, we consider two optimisation goals of tenants based on the two example quality dimensions: minimising cost or minimising response time. Accordingly, the tenants can be categorised into two clusters based on their quality requirements. Thus we feed K-Means algorithm with $k = 2$, which indicates the number of tenant clusters. After clustering, the tenants in a cluster are of high similarity compared to one another in terms of quality requirements but are very dissimilar to tenants in any other cluster.

5.1.2 Phase 2: Similarity mapping

Based on the numerical characteristic of quality requirements and quality values, both tenants and services can be represented as the points in a multi-dimensional vector space. Although given the fact that tenants' quality requirements for the

SBS are global whereas the services' quality values are local, we can identify a connection between these two levels of quality. For example, tenant t_e requires fast response time and consequently can accept a high cost, and candidate service $s_{i,j}$ may have the similar quality feature, i.e., it can respond the service request quickly at a high price. Therefore, $s_{i,j}$ has the high potential to participate in the execution plan that meets t_e's quality requirements. When clustering is completed, there is a centroid in each tenant cluster that represents the feature of quality requirements in that cluster. Since the tenants' quality requirements have been normalised in phase 1 and all quality values have been converted into the range of [0, 1], thus the centroids of tenant clusters can be mapped to the points in the multi-dimensional vector spaces that represent the service classes, where the mapped points will be used as centroids for service clustering in phase 3.

5.1.3 Phase 3: Service clustering

In this phase, candidate services are partitioned into different clusters based on their multi-dimensional quality. Each service cluster has the similar quality feature with its corresponding tenant cluster. For the same purpose of eliminating the incomparability between the measurement scales for different quality dimensions, the quality values of candidate services are first normalised using (5.2) with Min-Max normalisation technique [13]:

$$qn_p(s_{i,j})=\begin{cases}\dfrac{q_p^{max}(SC_i)-q_p(s_{i,j})}{q_p^{max}(SC_i)-q_p^{min}(SC_i)} & \text{if } q_p^{max}(SC_i)\neq q_p^{min}(SC_i)\\ 1 & \text{if } q_p^{max}(SC_i)=q_p^{min}(SC_i)\end{cases} \tag{5.2}$$

where $q_p^{max}(SC_i)$ is the maximum value and $q_p^{min}(SC_i)$ is the minimum value for the p^{th} quality dimension in the i^{th} service class, $q_p(s_{i,j})$ is the quality value of the p^{th} dimension of the j^{th} service in the i^{th} service class, and $qn_p(s_{i,j})$ denotes the p^{th} normalised quality value of $s_{i,j}$.

After that, for each candidate service, the Euclidean distances between the services and the existing centroids mapped from tenant clusters are calculated with (5.3). The service is categorised into the service cluster represented by the centroid with shortest distance to the service.

$$d_{EUS} = \sqrt{\sum_{p=1}^{t} |qn_p(s_{i,j}) - q_p(csc_{i,l})|^2} \tag{5.3}$$

where $q_p(csc_{i,l})$ is the quality value of the p^{th} dimension of the mapped centroid of the l^{th} service cluster in the i^{th} service class.

For example, the Euclidean distance between service $s_{0,0}$ and the two mapped centroids $csc_{0,0}$ and $csc_{0,1}$ are 0.42 and 0.14 respectively. Thus, $s_{0,0}$ is categorised into cluster 1 of service class 0. When service clustering completes, we can get a list $SCLU_i = \{sclu_{i,0}, sclu_{i,1}, ..., sclu_{i,k}\}$ for service class SC_i, where each entry $sclu_{i,l}(0 \leq l \leq k)$ denotes a service cluster.

5.1.4 Phase 4: Service ranking

Each service class $SC_i = \{s_{i,0}, ..., s_{i,m}\}, SC_i \in \mathbb{S}$, contains m functionally equivalent candidate services with differentiated multi-dimensional quality. It is a complex multi-attribute decision making problem to select from these services to create a service composition based on their quality values. For the aim of selecting representatives in each service cluster, we rank the services in the same service cluster according to their utility value. We use the method discussed in Section 4.4 for service utility evaluation based on tenants' multi-dimensional quality preferences on the entire SBS and service's multi-dimensional quality.

The tenants' quality preferences are usually differentiated. For example, a tenant may have strong preference for response time, while another may prefer high system throughput. In this thesis, we use weights to represent the tenants' preference trade-off between different quality dimensions. Given the fact that tenants' quality preferences are often varied, we use the average weights across all the tenants that share a given service class SC_i to calculate the utility of the service.

The average weight on a quality dimension of service class SC_i is calculated by (5.4):

$$w_{i,p} = \frac{1}{t_i} \times \sum_{e=1}^{t_i} w_{e,p}, p = 1, ..., d \tag{5.4}$$

where t_i is the number of tenants that share service class SC_i, d is the number of quality dimensions, $w_{e,p}$ indicates the e^{th} tenant's weight for its quality preference for the p^{th} quality dimension, $\sum_{p=1}^{d} w_{e,p} = 1$ and $w_{i,p}$ is the average weight for the quality preferences for the p^{th} quality dimension across all the tenants that share SC_i. The tenants' quality preferences can be obtained by the system engineer through the analysis of tenants' business requirements as well as the corresponding SLA [87].

The quality values of candidate services have been normalised in Phase 3 to allow a uniform measurement of the multiple quality dimensions regardless of their different scales. After that, we calculate the average utility of a given service $s_{i,j}$ using the normalised d-dimensional quality values and d-dimensional average quality weights. Formula (5.5) is the widely used Simple Additive Weighting (SAW) method [83]:

$$u_(s_{i,j}) = \sum_{p=1}^{d} \times qn_p(s_{i,j}) \tag{5.5}$$

where $qn_p(s_{i,j})$ is the normalised quality value of the p^{th} quality dimension of $s_{i,j}$.

Then, the services in each service cluster are ranked by their utility values. The services with higher utilities will be recommended for service composition in the next phase.

5.1.5 Phase 5: Service recommendation for composition

In this phase, service selection for multi-tenant SBS is modelled as a Constraint Optimisation Problem (COP). Given an SBS $\$$ that consists of $n(n \geq 1)$ tasks and serves $t(t \geq 1)$ tenants simultaneously, there are n service classes $SC_i, i = 1, ..., n$, and each SC_i contains $m(m \geq 1)$ candidate services $s_{i,j}, j = 1, ..., m$, that vary in

d-dimensional quality values $q_p, p = 1, ..., d$. The aim of service selection for $\mathbb{S}$ is to find a solution to create a service composition with t execution plans $epl_e, e = 1, ..., t$, to meet the corresponding tenants' overall quality requirements for the SBS simultaneously and individually, and in the meantime, to achieve the optimisation goal of service selection collectively, such as maximising the overall utility of the system.

We first use the tenants' overall quality requirements as a set of constraints on the end-to-end quality of the SBS, and model the problem as a Constraint Satisfaction Problem (CSP), which consists of a set of 0-1 integer variables $X = \{x_1, ..., x_y\}$ and a set of constraints over X. In the process of finding a solution to the CSP model, each variable is assigned a value of 0 or 1 and thus all constraints of the CSP can be fulfilled. For an SBS $\mathbb{S}$ shared by t tenants, if there are n service classes and each of them consists of m candidate services, then $t \times n \times m$ variables $x_{e,i,j}(e = 1, ..., t, i = 1, ..., n, j = 1, ..., m)$ are created in the CSP. $x_{e,i,j}$ with value of 1 indicates that the j^{th} service in the i^{th} service class is selected to participate in the execution plan for the e^{th} tenant, 0 otherwise. In the model, the constraints are as follows:

$$\sum_{j=1}^{m} x_{e,i,j} = 1, \ \forall e \in [1, t], \ \forall i \in [1, n] \tag{5.6}$$

$$\sum_{e=1}^{t} \sum_{j=1}^{m} x_{e,i,j} = t, \ \forall i \in [1, n], \ \forall j \in [1, m] \tag{5.7}$$

$$q_p(epl_e) \leq c_{e,p}, \ \forall e \in [1, t], \ \forall p \in [1, d] \tag{5.8}$$

Constraint family (5.6) guarantees that for each tenant one and only one service in each service class can be selected for composition, and constraint family (5.7) ensures that each tenant has an execution plan. Constraint family (5.8) makes sure that tenants' overall multi-dimensional quality requirements for the SBS can be satisfied, where $c_{e,p}, e = 1, ..., t, p = 1, ..., d$ is the p^{th} quality requirement of the e^{th} tenant.

When solving the CSP problem above, multiple feasible solutions can be found, among which the optimal one should be identified in order to achieve the optimisation goal of service selection. Therefore we turn the CSP into a COP by adding to it an objective function that quantifies the optimisation goal. We adopt maximising the overall system utility as the optimisation goal, which can be expressed by (5.9):

$$Objective(\mathbb{S}) : Maximising(u(\mathbb{S})) \tag{5.9}$$

where $(u(\mathbb{S})$ is the overall system utility of S, which is calculated as introduced in Section 4.4.

In order to build the search space of candidate services for the COP, we use a greedy method by which the representative services with the best utility values are selected from each service cluster in a multi-round manner. The steps of this method are as follows:

1. From each service cluster in a service class, $T(tc_c) \times 2^R$ services are selected and added into the search space in the R^{th} round, where $T(tc_c)$ is the number of tenants in the c^{th} cluster and $\sum_{c=1}^{k} T(tc_c) = t$ (k is the cluster number used for the K-Means algorithm and t is the total number of tenants), and R is the round in which the solving process is performed.

2. In the R^{th} round, after inserting the representative services into the search space, the IP technique is used to solve the COP.

3. If a solution of the COP is found, go to step 4. If no solution can be found and more candidate services are available, R increases by 1 and go to Step 1 for a new round. If a solution is not found and all the candidate services are exhausted, the process terminates.

4. After a solution of the COP is found, execution plans can be created based on the solution to fulfil both the quality requirements of tenants and the optimisation goal of service selection.

Based on our service recommendation approach, the service selection method introduced above is a greedy method intrinsically, which is able to find the near-optimal solutions to the COP.

5.2 Experimental evaluation

We have conducted a range of experiments to evaluate our service recommendation approach in terms of effectiveness and efficiency. Section 5.2.1 introduces the evaluation metrics. Section 5.2.2 describes the experimental setup of our evaluation. Section 5.2.3 analyses the experimental results of SSR4MTS.

5.2.1 Evaluation metrics

Extensive experiments are conducted to evaluate SSR4MTS in terms of effectiveness and efficiency. The effectiveness is measured by the following two metrics:

- Success rate, which is defined by the percentage of test instances where a satisfactory solution for service selection is found.
- System optimality, which is evaluated based on the objectives of the COPs.

The efficiency of all approaches is evaluated based on the average computation time needed for service selection for an SBS.

5.2.2 Experimental setup

We implemented SSR4MTS in Java with JDK 1.6.0. For the aim of comparison with other existing approaches, we also implemented four other optimisation approaches. IBM CPLEX v12.6, a popular linear programming tool, is used to solve the COPs.

The services used in our experiments are created based on QWS [88], a publicly available Web service dataset that consists of more than 2500 real-world Web services with nine-dimensional quality including response time. We randomly divided them into *n* (up to 100) service classes. Cost is randomly generated and added to each candidate service as an additional quality dimension, and the trade-off between cost and response time is considered, e.g., the cost of a service should be higher if it offers faster response time. The overall system utility is used to model the optimisation goal of maximising the tenants' satisfaction over the SBS.

We run experiments with different n, i.e., the number of service class, and the experimental results are consistent with those from the experiments on the *VOVS* introduced in Section 2.2.1. We use $n = 10$ as the default setup for the experiments introduced in next section unless otherwise specified.

For the purpose of simplicity without losing generality, tenants' quality overall requirements for the SBS are generated based on response time and cost, which are used as representative quality dimensions throughout this chapter. In this process, the tenants' optimisation goals based on the two example quality dimensions are considered. Thus, as discussed in Section 5.1.1, the tenants are partitioned into two clusters. We conducted experiments with different numbers of tenants (up to 500), and the experimental results are consistent with those from the experiments demonstrated in this section.

Tenants usually have diverse quality requirements for an SBS, which are modelled as the quality constraints in the COP for service selection. The difficulties in finding a solution to satisfy these requirements are different, sometimes easy sometimes hard. These different difficulties are quantified by the difficulty levels in the experiments. The work in [13] has demonstrated that the difficulty levels of tenants' quality requirements have a significant impact on the efficiency of service selection. Therefore, we use the method proposed in [13] to generate the quality constraints of the COP randomly with different difficulty levels: *easy*, *medium*, and *severe*. A constraint with difficulty level of easy can be satisfied easily, while a severe constraint is the most difficult to satisfy due to the stringent demand imposed on the corresponding quality dimension.

Most existing approaches for service selection are designed for single-tenant

SBSs [13][20][27]. In the experiments, we implemented an optimisation approach originated from [49] named *Skyline-Global* and three other optimisation approaches proposed in MSSOptimiser [22], which are originated from [27][49] and adjusted for the purpose of comparison in the context of multi-tenant SBSs:

- *Skyline-Global*: In each service class, skyline services are identified, and then clustered using K-Means. Representative service with the highest utility in each cluster is recommended for service selection. Service clustering, recommendation and selection are executed in a multi-round and greedy manner. Execution plans for all the tenants are created in one COP model.
- *Exact-Global*: Execution plans for all the tenants are created in one COP model. All the candidate services in each service class are inserted into the search space at one time.
- *Greedy-Global*: The search space of candidate services is built in multiple rounds with a greedy algorithm. The representative services with highest utility values are prioritised to enter the search space. Execution plans for all the tenants are also created in one COP model.
- *Exact-Local*: Execution plans for the tenants are created one by one in respective COP models. All the candidate services in each service class are added to the search space at one time.

We conducted the experiments on a machine running Windows 7x64 Enterprise with Intel(R) Core (TM) i5-4570 3.2 GHz CPU and 8 GB RAM. For each set of experiments, the results are collected, averaged and compared from 100 test instances.

5.2.3 Experimental results

In this section, we evaluate SSR4MTS in terms of its effectiveness and efficiency in service selection by comparing the success rate and the computation time with four other approaches introduced in the previous section.

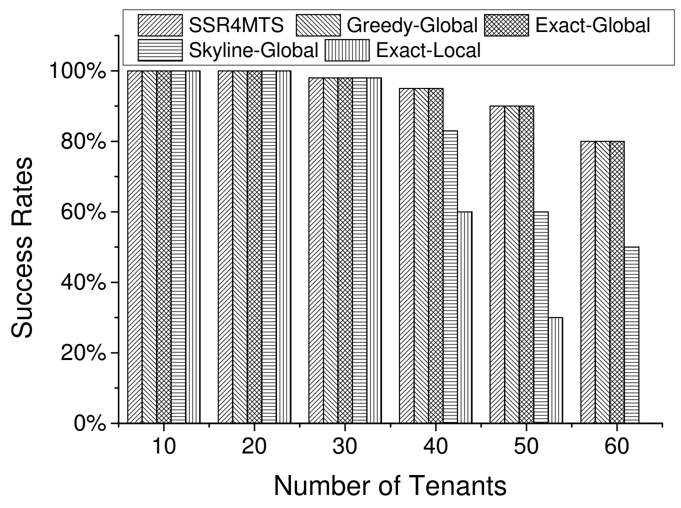


Figure 5.2: Selection success rates vs number of tenants.

We first compare the five optimisation approaches in terms of success rates, which are represented by the percentage of scenarios where a solution of the COP is found. We fixed the number of candidate services in each service class at 100 and varied the number of tenants from 10 to 60 in steps of 10. The results show that clustering technique and greedy algorithm used in our approach have no negative impact on the effectiveness. As shown in Fig. 5.2, the success rates demonstrated by Skyline-Global and Exact-Local decrease from 100% to 49% and 0% respectively as the number of tenants changes from 30 to 60, while the success rates obtained by other three approaches maintain at a relatively high level (about 90% or above) across all scenarios. The sharp decrease in success rate of Exact-Local is because it creates execution plan for each tenant in respective COP model. Once the candidate services are selected to create one execution plan for a tenant, they will be removed from the search space. This leads to the decrease in the number of available candidate services for the remaining tenants, which can cause failures in service selection. The success rate obtained by Skyline-Global also decreases because the available candidate services are reduced by using skyline technique. Other approaches consider all tenants in one COP model and can use all the candidate services without removing any one from the search space,

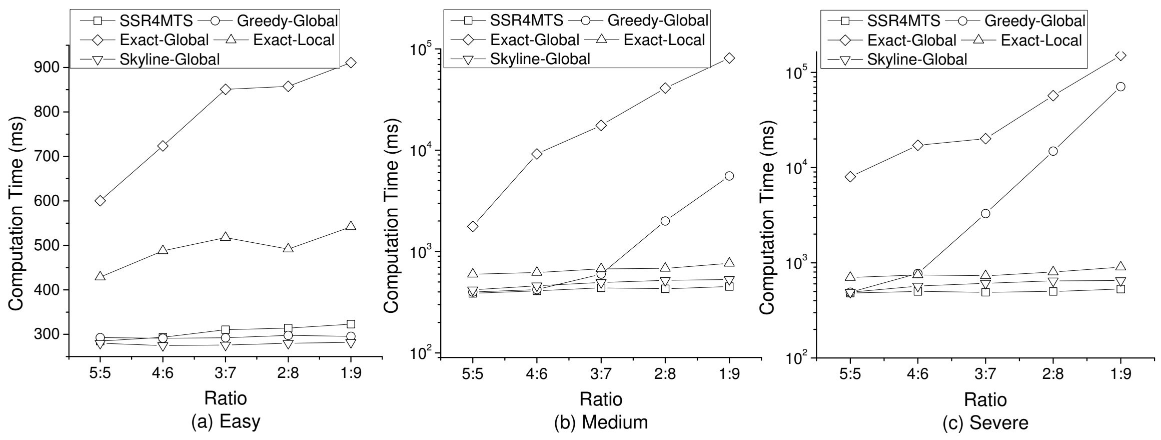


Figure 5.3: Selection computation time at different ratios, and difficulty levels of quality requirements.

and thus show relatively high success rates.

And then we evaluate the efficiency of SSR4MTS by comparing the computation time with other approaches. Computation time is used to represent the computational overhead introduced in building an SBS through service composition, which is a major concern of the SBS vendor. Due to the fact that tenants usually have different quality requirements, which can be regarded as the points in a multi-dimensional vector space, we use the ratio between the numbers of tenant in the two different clusters to present the distribution of tenants. In this set of experiments, we fixed the number of tenants at 100 and the number of candidate services in each service class at 300. Then we changed the difficulty levels of tenants' quality requirements from easy to severe, and the ratio between the numbers of tenants in the two clusters from 5:5 to 1:9, where 5:5 means the even distribution of tenants in the two clusters, while 1:9 indicates that there are much more tenants in the second cluster than in the first cluster.

The experimental results are shown in Fig. 5.3. When the difficulty level of quality requirements changes from easy to severe, or the ratio varies from 5:5 to 1:9, the computation times shown by all approaches increase. However, compared with four other approaches, SSR4MTS shows a better performance except when the difficulty level is easy, where all approaches can easily find solutions to the COPs and SSR4MTS is marginally outperformed by Greedy-Global and

Skyline-Global. The Exact-Global approach can obtain the global optimal solution but consumes the most computation time across all the test instances (over 100s on average in the worst case). Exact-Global and Exact-Local are outperformed by SSR4MTS in all cases. The computation time obtained by Greedy-Global increases dramatically when the difficulty level of quality requirements increases and the ratio becomes uneven, which reaches approximately 70s on average when difficulty level is severe and ratio is 1:9. Its slight advantage over SSR4MTS shown in Fig. 5.3 (a) is because SSR4MTS consumes some extra time in clustering tenants and candidate services, and in the scenarios where the solutions to the COP are easily found, the extra time consumed has obvious impact on the total computation time. Skyline-Global outperforms SSR4MTS marginally because it only clusters the skyline candidate services for selection. However, when quality requirements become hard to satisfy and the ratio becomes uneven, the extra time introduced by clustering constitutes only a very small portion of the total computation time, and thus can be neglected. In such circumstance the advantage of SSR4MTS approach becomes more apparent. As presented in Fig. 5.3 (c), when the difficulty level of quality requirements is severe, indicating that tenants' quality requirements are difficult to fulfil, the advantage of SSR4MTS in outperformance over Greedy-Global starts to increase significantly. The experimental results show that our service recommendation approach can capture the differences between tenants' quality requirements and distribution, based on which, SSR4MTS can achieve high efficiency in service selection.

In the experiments of this chapter, maximising the overall system utility is used as optimisation goal of the COP, as indicated by formula (4.6). Accordingly a higher overall system utility indicates higher system optimality. This can be used to evaluate the effectiveness of SSR4MTS in finding the optimal solution. We conducted a set of experiments to investigate the objective values of the solutions found by the five approaches after solving the COPs. The objective value indicates the overall system utility. We combine the utilities with the selection suc-

Table 5.1: Comparison in System Optimality (*Utility* $\times$ *Success Rate* / $Utility_{OPT}$)

Number of tenants: number of services	**SSR4MTS**	**Greedy-Global**	**Exact-Global**	**Exact-Local**	**Skyline-Global**
10 : 100	98.3%	90.1%	100%	100%	75.2%
20 : 100	98.1%	94.9%	100%	99.9%	74.1%
30 : 100	99.3%	98.7%	100%	59.1%	72.9%
40 : 100	99.6%	99.9%	100%	23.1%	61.4%
50 : 100	100%	100%	100%	2.3%	44.4%

cess rates and compare them with the optimal utilities obtained by Exact-Global, which considers all the candidate services at one time and thus can find the optimal solutions. Similar calculation and comparison of system optimality have also been employed in [13], which is extended in this thesis by taking success rate into consideration. We fix the number of candidate services per service class at 100, and vary the number of tenants from 10 to 50. As demonstrated in Table 5.1, the system optimality obtained by SSR4MTS is nearly the same as that of Exact-Global (more than 98% across all the cases). The Exact-Local and Skyline-Global show degraded optimalities when the number of tenants increases due to the success rate of finding a solution decreases dramatically. The experimental results show that in spite of the greedy method used in service recommendation and selection, SSR4MTS can still maintain high system optimality because of the utility-based service ranking process in the service recommendation.

In order to analyse the impact of difficulty level of tenants' quality requirements, we compare the computation time of the five approaches at different difficulty levels by fixing the ratio at 5:5. As presented in Fig. 5.4, when the difficulty of quality requirements varies from easy to severe, the computation time observed for all approaches increases. However, due to the greedy methods used to build service search space, SSR4MTS, Skyline-Global and Greedy-Global beat Exact-Local and Exact-Global. SSR4MTS outperforms other approaches across all test cases. Skyline-Global and Greedy-Global achieve computation time similar to SSR4MTS, approximately 200ms on average less than that achieved by

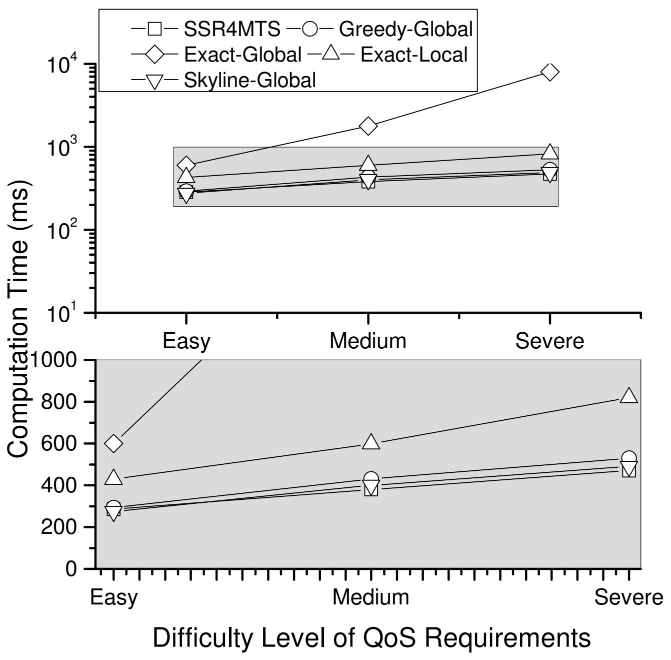


Figure 5.4: Selection computation time at different difficulty levels of quality requirements.

Exact-Local. The computation time obtained by Exact-Global increases exponentially when the difficulty level of quality requirements varies from easy to severe, which is much higher than those of other approaches.

After comparing with other four approaches in terms of success rate and computation time, we can see that our service recommendation-based approach SSR4MTS can ensure high effectiveness of service selection in finding a solution, and it is the best option in most cases where efficiency is the major concern. When the tenants' quality requirements are hard to satisfy and distributed unevenly with distinctive features, the advantages of SSR4MTS are obvious, because it is more efficient in service selection by finding appropriate services to fulfil the tenants' requirements based on the quality similarity between them. Exact-Local is able to obtain an excellent performance with satisfactory optimality where a large number of candidate services are available and thus the success rate can be guaranteed. Skyline-Global is similar to Exact-Local with higher efficiency but lower success rates and optimality. Exact-Global might be more preferred in scenarios with a strong preference in optimality in spite of the potential high computational overhead, especially in large-scale scenarios.

5.3 Discussion

SSR4MTS is designed to support effective and efficient service selection at build-time, however, it also can be used at runtime to improve the efficiency of service adaptation when re-optimisation of SBS is unavoidable. In fact, our runtime LSH-based service adaptation approach is based on the service clusters discussed in this chapter.

We use the widely known clustering algorithm K-Means in SSR4MTS. Due to the inherent limitations of the K-Means algorithm, the number of clusters must be pre-specified, and different initial centres may result in different final clusters. However, these can be overcome by utilising alternative clustering methods. In fact, SSR4MTS does not rely on the K-Means algorithm. Other clustering algorithms can also be integrated into SSR4MTS in different domains as long as they serve the domain-specific needs.

5.4 Summary

In this chapter, we present SSR4MTS, a novel approach for service selection using service recommendation based on clustering techniques. It focuses on the quality management for multi-tenant SBSs at the lifetime stage of service selection. The evaluation results show that SSR4MTS can guarantee high effectiveness whilst significantly improve the efficiency of service selection.

Chapter 6

Service selection based on correlated quality requirements

When different dimensions of tenants' quality requirements for an multi-tenant SBS become correlated, the quality requirements are no longer deterministic but dynamically varied, which raises new challenge to the service selection for multi-tenant SBSs. In this chapter, we introduce Service Selection based on Correlated quality requirements for Multi-Tenant SBSs (SSC4MTS), which supports the formalisation of quality correlations and service selection for multi-tenant SBSs based on correlated quality requirements. This chapter is based on my paper [89].

This chapter is organised as follows. Section 6.1 introduces the SSC4MTS approach. Then Section 6.2 presents the evaluation of SSC4MTS in terms of effectiveness and efficiency. Section 6.3 discusses some premises of SSC4MTS. Finally, Section 6.4 summarises this chapter.

6.1 SSC4MTS approach

SSC4MTS consists of two phases. First, the correlated quality requirements are formalised with quality correlation functions. Then the service selection for multi-tenant SBSs based on such quality requirements is modelled as a Constraint Optimisation Problem (COP) and Integer Programming (IP) techniques are employed to find the optimal solution.

6.1.1 Formalisation of correlated quality requirements

The correlated quality requirements for an SBS can be initially expressed with qualitative and linguistic terms, which then can be formalised empirically by the system engineers during the Service Level Agreement (SLA) negotiation process. From the perspective of the system engineer, three examples of correlated quality requirements on cost and throughput of the online video streaming system *VOVS* introduced in Section 2.2.1 are as follows:

- Requirement of tenant 1 ($r(t_1)$): *"It is acceptable to pay 1 dollar for each increase of 1, 000 req/s (requests per second) in throughput, up to 20 dollars in total, and the throughput cannot be less than 2, 000 req/s"*.
- Requirement of tenant 2 ($r(t_2)$): *"We would like to pay 2 dollars for each increase of 1, 000 req/s in throughput, but the throughput must be more than 4, 000 req/s and cost must be less than 32 dollars"*.
- Requirement of tenant 3 ($r(t_3)$): *"We only can pay 0.5 dollars for each increase of 1, 000 req/s in throughput after paying 6 dollars of basic service charge, and the throughput of more than 20, 000 req/s is unnecessary."*

The correlations, i.e., trade-off, between cost and throughput, as well as the constraint on each dimension, can be identified from the above quality requirements, which can be represented explicitly with three functions, named *quality* *correlation function* in this thesis. For a quality correlation with d ($d \in N^{+}, d \geq 2$) dimensions, the function is defined by (6.1):

$$\begin{aligned} q_p(t) =& f(q_1(t), ..., q_{p-1}(t), q_{p+1}(t), ..., q_d(t)) \mid \\ & for\ \forall i \in [1, d],\ q_i(t) \in [q_i^{min}, q_i^{max}] \end{aligned} \tag{6.1}$$

where $q_i(t)$ is the variable that represents the i^{th} dimension of quality requirement from tenant t. q_i^{min} and q_i^{max} are the minimum and maximum constraints on the i^{th} quality dimension respectively. Then the three abovementioned correlated quality requirements can be represented with (6.2) to (6.4):

$$r(t_1) : q_{ct}(t_1) = q_{tp}(t_1),\ q_{tp}(t_1) \geq 2, q_{ct}(t_1) \leq 20 \tag{6.2}$$

$$r(t_2) : q_{ct}(t_2) = 2(q_{tp}(t_2) - 4),\ q_{tp}(t_2) \geq 4, q_{ct}(t_2) \leq 32 \tag{6.3}$$

$$r(t_3) : q_{ct}(t_3) = \frac{1}{2} q_{tp}(t_3) + 6,\ 2 \leq q_{tp}(t_3) \leq 20 \tag{6.4}$$

where q_{ct} and q_{tp} are the variables that represent the quality values of cost and throughput respectively, and their units are dollars and $10^3 req/s$ respectively.

Similar to many other research works [13][20][40], in this thesis, we only fo-

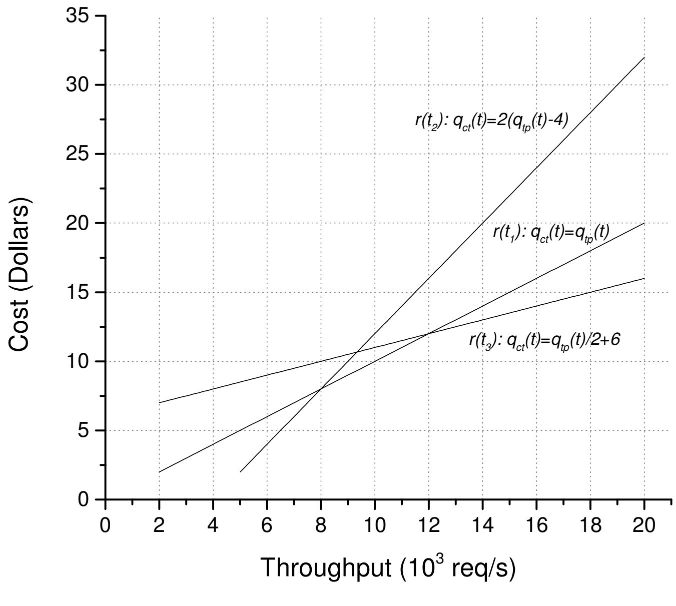


Figure 6.1: Example quality correlation functions.

cus on the quality dimensions with numerical values. Thus from the perspective of spatial geometry, a quality correlation function that involves d quality dimensions can be represented with a graph, e.g., a straight or curved line, in a d-dimensional Euclidean space. With cost and throughput, Fig. 6.1 shows the three example quality correlation functions including (6.2) to (6.4). Because the quality evaluation of throughput increases along with the increase of its quality value, a higher throughput incurs higher cost. Thus, as demonstrated by the correlation functions in Fig. 6.1, the cost monotonically increases with throughput.

In order to eliminate the impact of different measurement units of quality dimensions in quality correlation, we normalise the quality value of each dimension of a quality requirement into the same range [0, 1]. For a given $q_p(t)$, we can normalise it by (6.5) with widely used Min-Max normalisation technique [20].

$$Q_p(t)=\begin{cases}\dfrac{q_p(t)-q_p^{min}(\$)}{q_p^{max}(\$)-q_p^{min}(\$)} & \text{if } q_p^{max}(\$)\neq q_p^{min}(\$)\\ 1 & \text{if } q_p^{max}(\$)=q_p^{min}(\$)\end{cases} \tag{6.5}$$

where $q_p(t)$ is the variable that represents the quality value of the p^{th} dimension of quality requirements of tenant t, $q_p^{max}(\$)$ and $q_p^{min}(\$)$ are the p^{th} maximum and minimum quality values respectively that SBS \$ can offer based on the available candidate services, $q_p(t) \in [q_p^{min}(\$), q_p^{max}(\$)]$, and $Q_p(t)$ represents the normalised p^{th} quality value of the requirement. Then with (6.5), equation (6.6) holds:

$$q_p(t)=(q_p^{max}(S)-q_p^{min}(\$)) \times Q_p(t) + q_p^{min}(\$) \tag{6.6}$$

With (6.1) and (6.6), we can obtain the normalised quality correlation function (6.7):

$$\begin{aligned}Q_p(t) &= f(Q_1(t), ..., Q_{p-1}(t), Q_{p+1}(t), ..., Q_d(t)) \mid \\ &\quad for\ \forall i \in [1,\ d],\ Q_i(t) \in [Q_i^{min},\ Q_i^{max}]\end{aligned} \tag{6.7}$$

where Q_i^{min} and Q_i^{max} are the normalised minimum and maximum constraints on the i^{th} quality dimension respectively.

Taking the quality correlation functions (6.2) to (6.4) for example. Given the maximum and minimum throughput in Fig. 6.1 are 20, 000 and 2, 000 *req/s* respectively, and the maximum and minimum cost of the SBS are 32 and 2 dollars respectively, we can obtain the normalised quality correlation functions (6.8) to (6.10) according to (6.5) and (6.6):

$$R(t_1) : Q_{ct}(t_1) = \frac{3}{5} Q_{tp}(t_1),\ Q_{tp}(t_1) \geq 0, Q_{ct}(t_1) \leq 1 \tag{6.8}$$

$$R(t_2) : Q_{ct}(t_2) = \frac{6}{5} Q_{tp}(t_2) - \frac{1}{5},\ Q_{tp}(t_2) \geq \frac{1}{9}, Q_{ct}(t_2) \leq 1 \tag{6.9}$$

$$R(t_3) : Q_{ct}(t_3) = \frac{3}{10} Q_{tp}(t_3) + \frac{1}{6},\ 0 \leq Q_{tp}(t_3) \leq 1 \tag{6.10}$$

Their function graphs are shown in Fig. 6.2.

The maximum and minimum quality values that SBS $\mathbb{S}$ can offer can be acquired with greedy methods. For example, for each task in the SBS, we select the candidate service with the highest cost, and then we can obtain the maximum cost of the SBS by calculating its overall cost with the corresponding quality aggregation function introduced in Section 4.3.

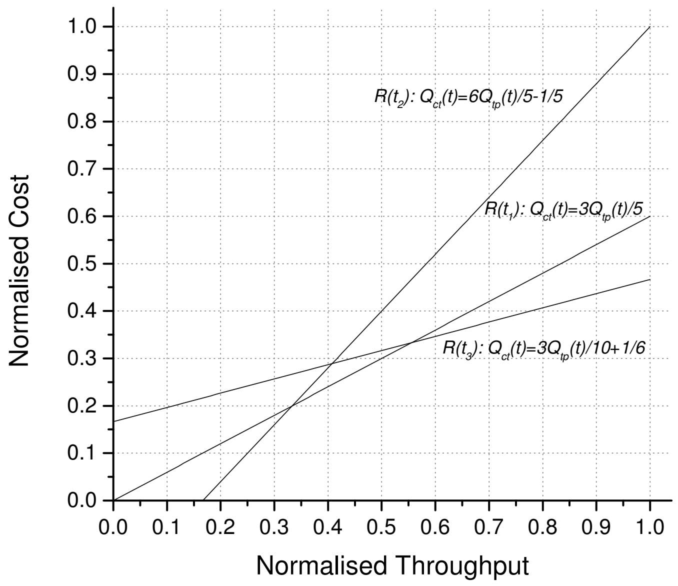


Figure 6.2: Example normalised quality correlation functions.

There may be multiple quality correlations in a tenant's requirement, which can be handled in the same manner. After the quality correlation functions are normalised, service selection can be performed based on the functions and the candidate services.

6.1.2 Service selection for multi-tenant SBSs

In the service selection of SBS $\mathbb{S}$, only the service compositions that satisfy the tenants' quality requirements in all dimensions can be regarded as satisfactory solutions. As shown in Fig. 6.3, the grey area is enclosed by the line segments of the example quality correlation functions (6.8) to (6.9) that represents the tenants' correlated quality requirements. Only the service compositions with normalised quality values that fall in this area are satisfactory to the tenants, such as s_1 to s_4.

There are often optimisation goals in service selection for the SBS, such as minimised cost, maximised throughput, etc. In such context, service selection is to select proper services to create service compositions to fulfill tenants' quality requirements, while achieving the optimisation goals. As shown in Fig. 6.3, service composition s_1 is the best solution if the optimisation goal is to minimise

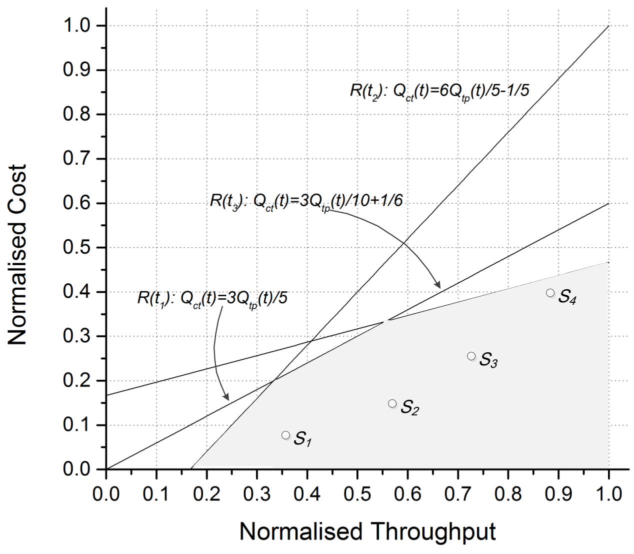


Figure 6.3: Service compositions with satisfactory quality.

the cost, while s_4 is definitely better than others when the optimisation goal is maximised throughput.

In SSC4MTS, we use *quality Satisfaction Degree* to represent the optimality of a satisfactory solution. Given a service composition s and the quality correlation function that represents the requirements of tenant t_i, f is the point with the shortest Euclidean distance to s in terms of quality values on the graph that represents the quality correlation function. The quality satisfaction degree (denoted by QSD) of s to a tenant t_i is defined by (6.11):

$$QSD(s,t_i) = w_1 \times \|\Delta Q_1(s,t_i)\| + ... + w_d \times \|\Delta Q_d(s,t_i)\|, i \in [1,T] \tag{6.11}$$

where

$$\Delta Q_p(s,t_i) = Q_p(s,t_i) - Q_p(f), p \in [1,d], i \in [1,T] \tag{6.12}$$

and

$$w_i = \frac{\frac{\partial Q_p(f)}{\partial Q_i(f)}}{\frac{\partial Q_p(f)}{\partial Q_1(f)} + ... + \frac{\partial Q_p(f)}{\partial Q_d(f)}} \mid \sum_{i=1}^{d} w_i = 1 \tag{6.13}$$

ΔQ_p is the difference in the p^{th} normalised quality value of tenant t_i's requirement between f and s. w_i represents the tenant's preference on the i^{th} quality dimension, which is calculated based on the partial derivatives of the quality correlation function with respect to each $Q_i(u)$ at f. All the points on the graph of quality correlation function have the same quality satisfaction degree with value of 0. Fig. 6.4 shows an example with cost and throughput, where f is the perpendicular foot of s to the line segment that represents the quality correlation function of tenant t_1.

Then the average QSD of service composition s is calculated by (6.14):

$$QSD(s) = \frac{1}{T}\sum_{i=1}^{T} QSD(s,t_i) \tag{6.14}$$

Then we can model the problem of service selection as a Constraint Optimisation Problem (COP). We use maximising the quality satisfaction degree as the objective of the COP. Considering an SBS $\mathbb{S}$ with n tasks, and each of them is associated with a service class containing m candidate services, we create a set of 0-1 integer variables $x_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,m}\}$ for the i^{th} service class. $x_{i,j}$ with value of 1 means that the j^{th} candidate service in the i^{th} service class is selected for the composition of SBS $\mathbb{S}$. The COP is formulated with (6.15) to (6.18):

$$Objective : Maximising\ QSD(S) \tag{6.15}$$

$$Q_p(S) \prec C_p, p \in [1, d] \tag{6.16}$$

$$Q_p(S, t_i) \prec f(Q_1(S, t_i), ..., Q_{p-1}(S, t_i), Q_{p+1}(S, t_i), ..., Q_d(S, t_i)), i \in [1, T] \tag{6.17}$$

$$\sum_{j=1}^{m} x_{i,j} = 1, i \in [1, n], j \in [1, m] \tag{6.18}$$

where symbol $\prec$ means "as good as or better than". Constraint family (6.16) ensures that tenants' constraints on every single quality dimension can be fulfilled, $Q_p(S)$ is the normalised p^{th} quality value of $\mathbb{S}$, and C_p is the tenants' normalised p^{th} quality constraint for $\mathbb{S}$, which can be obtained based on their quality correlation functions. Constraint family (6.17) guarantees that the quality correlations in the tenants' requirements can be satisfied. Constraint family (6.18) ensures that only one service in each service class can be selected to compose $\mathbb{S}$. We employ Integer Programming (IP) technique to find the optimal solution of the COP. The method for finding a point f with the shortest Euclidean distance to a given service composition in terms of quality values is domain-specific, which depends on the quality correlation function. Take the quality correlation functions in Fig. 6.4 for example, f can be found in many ways, such as IP techniques, Lagrange multiplier methods, etc.

6.2 Experimental evaluation

In this section, we present the experimental evaluation of SSC4MTS. Section 6.2.1 introduces the evaluation metrics. Section 6.2.2 introduces the setup of the experiments. Sections 6.2.3 and 6.2.4 present the evaluation of effectiveness and

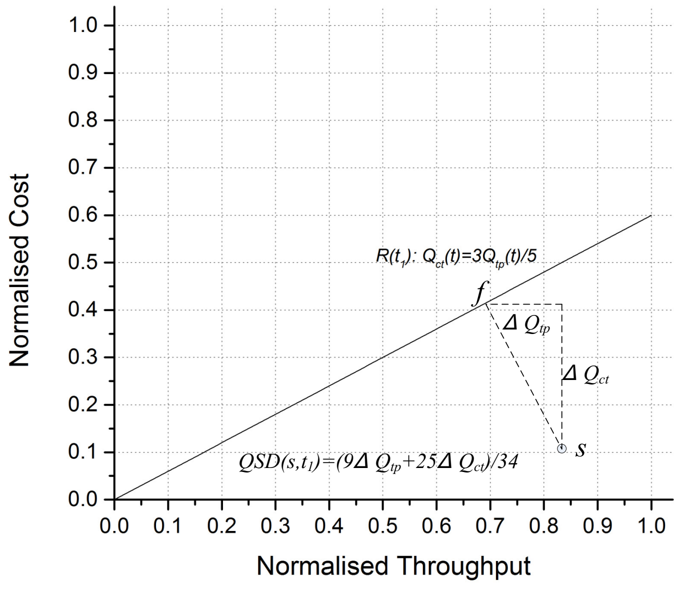


Figure 6.4: An example of quality satisfaction degree calculation.

efficiency of SSC4MTS respectively.

6.2.1 Evaluation metrics

We have conducted a range of experiments in a simulated environment to evaluate SSC4MTS in terms of effectiveness and efficiency by comparing it with other representative approaches. The effectiveness is again measured by two quantitative metrics: success rate and system optimality.

1. Success rate is defined by the percentage of test cases where a satisfactory solution for service selection is found.

2. System optimality is evaluated based on the objectives of the COPs, which are defined in (6.15). The objective values of the solutions found by different approaches are investigated and compared.

The efficiency of all approaches is evaluated based on the computational overhead, which is the average computation time needed for service selection for an SBS.

6.2.2 Experimental setup

SSC4MTS is implemented in Java with JDK 1.6.0. IBM CPLEX v12.6, a widely used linear programming tool, is again used to solve the COPs. Two representative approaches, similar to [22], are implemented for comparison as follows:

- **Utility-Optimal**: A COP is created for service selection where the objective is to maximise the system utility. Average user preference for each quality dimension is adopted.

- **Random**: A naive and non-optimal approach, which selects a candidate service randomly from each service class for service composition.

For the demonstration purpose, we use throughput and cost as two representative quality dimensions. The correlations between them in the quality requirements are linear. Other quality dimensions and correlations can be handled in the similar manner. The services used in the experiments are also generated based on QWS [88]. For each service, the cost is generated randomly by considering the trade-off between cost and throughput of the service.

The quality correlation and quality constraints determine the difficulty level of fulfilling a tenant's quality requirements. The works in [13][19] have demonstrated that the difficulty levels of tenants' quality requirements have significant impacts on the performance of service selection, especially the success rates. Thus, in the experiments, we generate tenants' quality constraints with different difficulty levels: easy, medium and severe, which are combined with different quality correlation functions to simulate the quality requirements in different difficulty levels. Fig. 6.5 shows an example based on cost and throughput, where the quality requirement represented by F_3 is more difficult to satisfy than that represented by F_1 due to a much small solution space.

In order to compare SSC4MTS with other approaches more comprehensively, we have mimicked SBSs with different numbers of tasks n (up to 100) and different numbers of candidate services in each service class m (up to 500). The

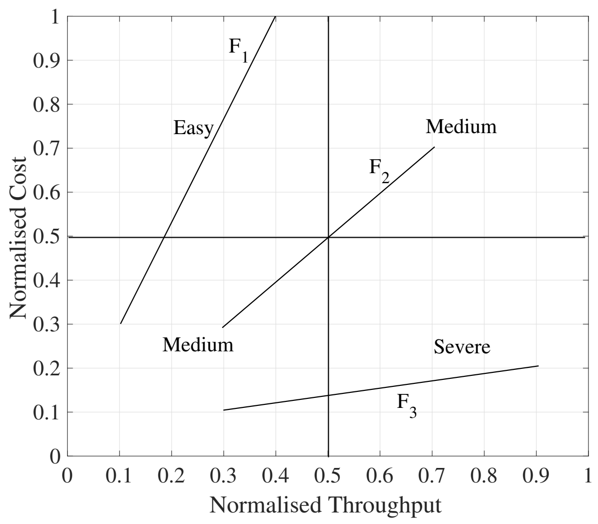


Figure 6.5: Difficulty levels of tenants' quality requirements.

findings are consistent with those from the experiments on the *VOVS* introduced in Section 2.2.1.

All the experiments were again conducted on a machine running Windows 7x64 Enterprise with Intel(R) Core (TM) i5-4570 3.2 GHz CPU and 8 GB RAM. The experimental results are collected, averaged and compared from 100 test instances.

6.2.3 Effectiveness evaluation

6.2.3.1 Comparison in success rates

With different n and m, similar comparison results in success rates are observed by changing difficulty levels of quality requirements. As an example, we fix n at 10, m at 10, vary the difficulty level of quality requirements from easy to severe, and investigate the success rates obtained by the three approaches in finding the solutions of service selection. As shown in Fig. 6.6, SSC4MTS and Utility-Optimal achieve the same success rates and outperform Random across all scenarios. When difficulty level changes from easy to severe, the success rates of SSC4MTS and Utility-Optimal decrease from 100% to 68%, while Random shows

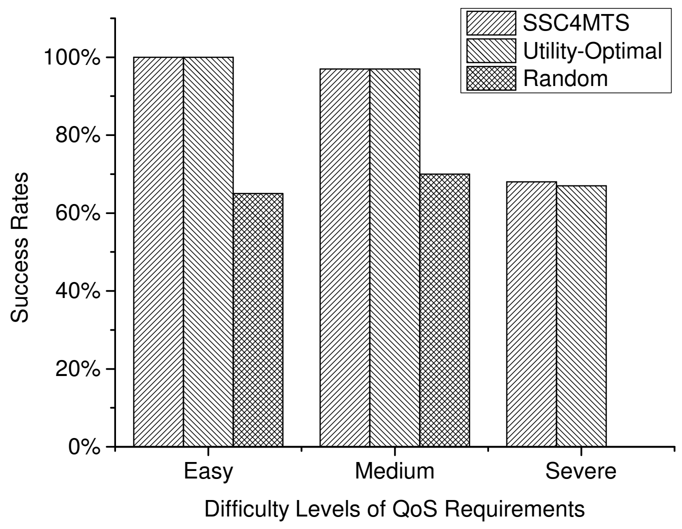


Figure 6.6: Success rates at different difficulty levels of tenants' quality requirements.

a dramatic decrease from 65% to 0%.

6.2.3.2 Comparison in optimality

Then we compare the three approaches in the system optimality with different difficulty levels of quality requirements. In this set of experiments, system optimality is measured by comparing SSC4MTS with Utility-Optimal and Random in terms of the quality satisfaction degree of the SBS. The comparison results are shown in Table 6.1. SSC4MTS shows the highest system optimality across all scenarios as expected. Utility-Optimal outperforms Random significantly with an average system optimality of 0.86. Random obtains the lowest system optimality, which is even not available due to the success rate of 0% when the difficulty level is severe.

Table 6.1: Comparison in System Optimality $(QSD(S))/QSD^{optimal})$

Difficult Level	SSC4MTS	Utility-Optimal	Random
Easy	1.0	0.78	0.25
Medium	1.0	0.89	0.51
Severe	1.0	0.92	N/A

6.2.4 Efficiency evaluation

6.2.4.1 Comparison in computation time

We conduct a range of experiments to evaluate the efficiency of SSC4MTS in this scenario. We change the number of tasks *n*, the number of candidate services per service class *m*, different quality correlation functions, etc., and compare the computation times consumed by all approaches. The results are consistent, which are demonstrated by an example in Fig. 6.7. We fix *n* at 10, difficulty level of quality requirements at medium, and vary *m* from 20 to 100 in steps of 20. Random shows the highest efficiency as expected with an average computation time of 0.01ms. SSC4MTS, which outperforms Utility-Optimal by 117ms on average, shows an increase in computation time from 37ms to 91ms when *m* increases to 100. The results show again that compared with other approaches, SSC4MTS can find the optimal solution to the COP with low computational overhead in most, if not all, scenarios.

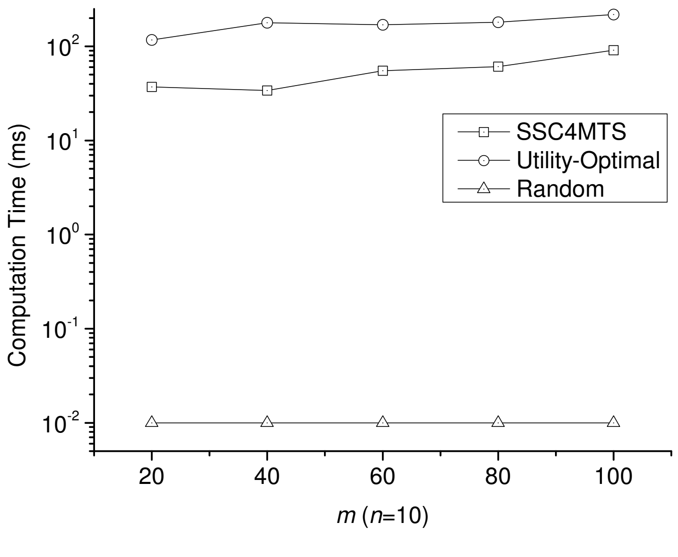


Figure 6.7: Computation time vs the number of candidate services per service class.

6.3 Discussion

In SSC4MTS, we focus on the quality requirements that can be represented by the linear quality correlation functions, however, the more complicated correlation functions, such as those with quadratic expressions, can be handled in a similar manner. The limitation is due to the capability of optimisation (linear or non-linear programming) tools. The quality correlation functions can be provided and formalised empirically by the system engineers of the SBS through the formulation of SLAs. However, the formalisation of quality correlation functions is domain-specific, which should be determined on a case-by-case basis.

6.4 Summary

In this chapter, we introduce a novel approach named SSC4MTS for service selection based on correlated quality requirements. The correlations between different quality dimensions of requirements are formalised with quality correlation functions and integrated into the COP that models the service selection problem. Quality satisfaction degree is proposed to use as the optimisation goal of service selection. The experimental results show that, compared with other representative approaches, SSC4MTS can find the optimal solutions for service selection with high success rates, high system optimality, and acceptable computational overhead.

Part II

Quality management for service monitoring

Chapter 7

Service monitoring based on criticality

As discussed in Section 1.2, it is of tremendous significance to monitor a multi-tenant SBS to provide quality guarantee to all tenants that share the SBS. Formulating cost-effective monitoring strategies is a challenging but crucial issue to be addressed. In this chapter, we introduce cost-effective Service Monitoring based on Criticality for Multi-Tenant SBSs (SMC4MTS), which aims at achieving cost-effectiveness in service monitoring for multi-tenant SBSs by identifying and prioritising the critical component services, i.e., the component services with high criticality, in the formulation of monitoring strategies. This chapter is based on my paper [82].

This chapter is organised as follows. Section 7.1 introduces the SMC4MTS approach. Then Section 7.2 presents the effectiveness and efficiency of SMC4MTS based on experimental evaluation. Section 7.3 discusses how the service criticality can be used in other scenarios such as service adaptation for multi-tenant SBSs. Some premises of SMC4MTS are also discussed. Finally, Section 7.4 summarises this chapter.

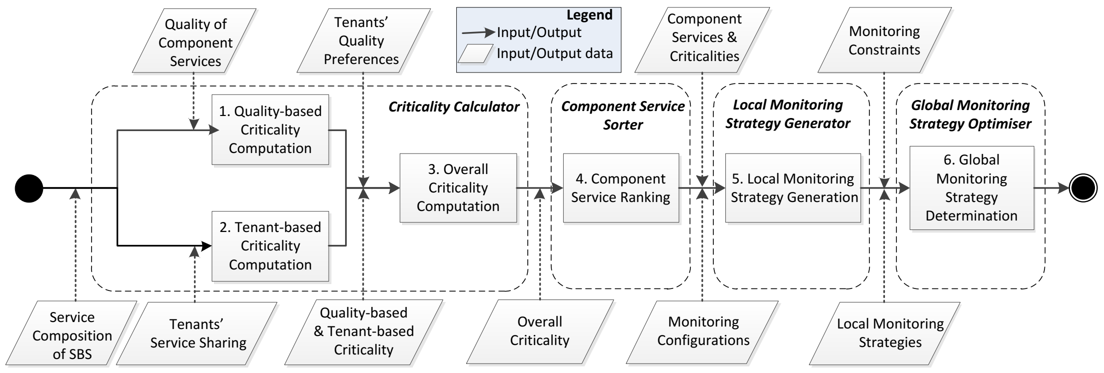

Quality of Component Services
Tenants' Quality Preferences
Legend
Input/Output
Input/Output data
Component Services & Criticalities
Monitoring Constraints
Criticality Calculator
1. Quality-based Criticality Computation
2. Tenant-based Criticality Computation
3. Overall Criticality Computation
Component Service Sorter
4. Component Service Ranking
Local Monitoring Strategy Generator
5. Local Monitoring Strategy Generation
Global Monitoring Strategy Optimiser
6. Global Monitoring Strategy Determination
Service Composition of SBS
Tenants' Service Sharing
Quality-based & Tenant-based Criticality
Overall Criticality
Monitoring Configurations
Local Monitoring Strategies

Figure 7.1: System architecture of SMC4MTS.

7.1 SMC4MTS approach

As discussed in Section 2.2.2, when identifying critical component services in the formulation of monitoring strategy for a multi-tenant SBSs, we must consider the adverse impacts on the system quality of an SBS caused by the component services upon runtime anomalies, as well as tenants' different business scenarios and their differentiated quality preferences. Thus, SMC4MTS calculates criticalities of the component services in a multi-tenant SBS based on the following three metrics: a) quality-based criticality, which is evaluated with sensitivity analysis techniques based on the impacts caused by service anomalies on the multi-dimensional quality of the SBS, b) tenant-based criticality, which is evaluated based on the impacts caused by service anomalies on multiple tenants in the SBS, and c) tenants' differentiated multi-dimensional quality preferences for the SBS. After the critical component services are identified, monitoring strategies can be formulated within the system vendor's monitoring budget by determining the component services to be monitored, and the corresponding monitoring parameters, such as number of monitors, monitoring frequency and granularity.

Fig. 7.1 shows the system architecture of SMC4MTS, which includes four functional components. Their roles are as follows:

1. ***Criticality Calculator***: This component calculates service criticality based

on quality of service, tenants' quality preferences and their service sharing.

2. ***Component Service Sorter***: This component ranks component services according to their criticalities.

3. ***Local Monitoring Strategy Generator***: This component generates local monitoring strategies for each component service in the SBS based on the monitoring parameters, and the corresponding benefits, resource cost and system overhead.

4. ***Global Monitoring Strategy Optimiser***: This component formulates an optimal global monitoring strategy for the entire SBS based on the local monitoring strategies.

SMC4MTS consists of six phases, as shown in Fig. 7.1. In Phase 1, we calculate the service criticality for each dimension of quality with the Sensitivity Analysis (SA) technique. In Phase 2, the criticality based on multi-tenancy is evaluated by analysing tenants' multi-dimensional quality preferences and service sharing. Based on the outcomes of Phase 1 and Phase 2, in Phase 3, the overall criticality of each service is calculated, by which the services in the composition are ranked in Phase 4. Local monitoring strategies are then generated in Phase 5. Finally, in Phase 6, formulation of cost-effective monitoring strategy based on criticality is modelled as an optimisation problem, and the Integer Programming (IP) technique is used to find the solution. These six phases are detailed one by one in this section.

7.1.1 Phase 1: quality-based criticality computation

In a dynamic cloud environment, various changes can occur to the component services of an SBS. The changes can be positive or negative, causing quality upgradation or degradation to the services, which often impacts the quality of the SBS. In SMC4MTS, we consider only the negative changes, i.e., quality degradation

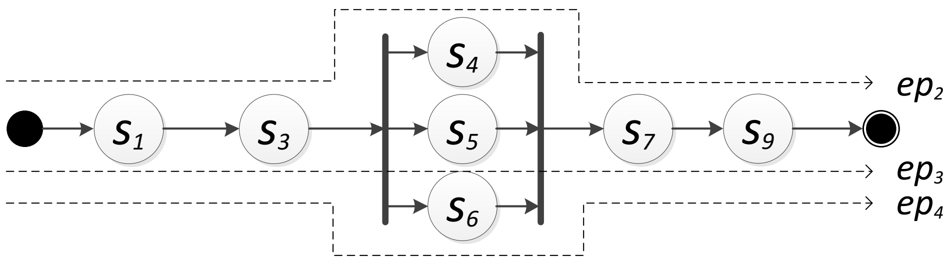


Figure 7.2: Execution plan 2 (epl_2) of $VOVS$.

incurred by runtime anomalies occurring to the services, as they can result in end-to-end system quality violation.

The sensitivities of the quality degradation of the SBS with respect to the quality degradation of different services may vary. Take Fig. 7.2 for example, as introduced in Section 4.2, there is a parallel structure composed by s_4, s_5 and s_6 in the execution plan epl_2 of $VOVS$. The execution time of the parallel structure depends on the maximum execution time of s_4, s_5 and s_6. Assume that the execution time of s_4, s_5 and s_6 are 5ms, 10ms and 15ms respectively, when a delay of 10ms occurs to each service respectively, the corresponding delays caused to epl_2 by s_4, s_5 and s_6 are 0ms, 5ms and 10ms, which thereby may have different impacts on the end-to-end response time of the SBS. We adopt the Sensitivity Analysis method, which is widely used in many areas [90][91][92], to quantify and analyse the impacts of quality degradation of component services on the end-to-end system quality.

We measure the criticality of a given service by the sensitivity of the quality degradation of an SBS $\mathbb{S}$ in comparison to the quality degradation of that service. In order to quantitatively evaluate the quality degradation of a component service, we first define *quality degradation coefficient* as follows:

Definition 7.1. ***Quality Degradation Coefficient***. Denoted by q_p^{dc}, the quality degradation coefficient indicates the maximum quality degradation level of the p^{th} quality parameter of a component service, represented by the ratio of the variation in the p^{th} quality value upon anomaly to the original quality value.

Take s_1 in Fig. 7.2 for example, if the quality degradation coefficients for response time and throughput are 200% and 80% respectively, the corresponding quality value of s_1 will increase by 200% to 300% of its original value and decrease by 80% to 20% of its original value respectively upon anomalies.

As shown in formula (7.1), we then degrade the p^{th} quality value of component service s_i by q_p^{dc} in steps of Δq_p percentage for r iterations, e.g., increasing response time or decreasing throughput of s_i by 20% each time. In the $k^{th}(0 \leq k \leq r)$ iteration, we investigate the variation percentage of p^{th} quality value of $\mathbb{S}$ relative to its original value, denoted by $\Delta q_{k,p}^{vp}(\mathbb{S}, s_i)$, and divide it by the variation percentage of the quality of s_i which is represented by $\Delta q_{k,p}^{vp}(s_i)$. The results are summed and averaged based on r iterations as the criticality of the p^{th} quality dimension of s_i, denoted by $cr_p^Q(s_i)$.

$$cr_p^Q(s_i) = \frac{\sum_{k=1}^{r}(\frac{\Delta q_{k,p}^{vp}(\mathbb{S}, s_i)}{\Delta q_{k,p}^{vp}(s_i)})}{r}, r = \frac{q_p^{dc}}{\Delta q_p} \tag{7.1}$$

When employing (7.1) in an SBS containing n component services, only the quality value of one service is varied at a time. In reality, however, the other $n-1$ component services may fail at the same time. According to the quality aggregation functions in Table 4.1 and the composition patterns in Figure 4.1, the quality degradations caused by multiple anomalous services can cumulate (as shown in Fig. 7.3 (a)) or mask (as shown in Fig. 7.3 (b)) one another. However, the effects of different anomalies on the quality of the SBS are independent, which therefore does not affect the criticalities of the component services.

In real-world scenarios, the quality degradation coefficient varies from service to service and thus must be determined empirically on a case-by-case basis. This is because there are many uncertain factors that may impact the actual quality degradation caused to a service by a runtime anomaly, e.g., the time taken to

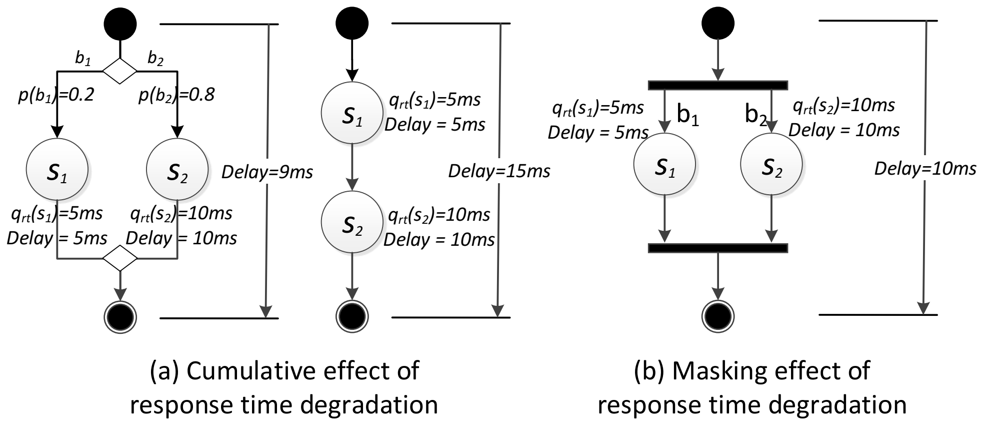


Figure 7.3: Effects of multiple service anomalies on SBS-level degradation in response time.

diagnose the anomaly, the difficulty of fixing the anomaly, etc. For the sake of simplicity, in this thesis we use a unified q_p^{dc} across all the services without affecting the effectiveness of the proposed approach. Alternatively, the maximum quality degradation level of a specific service can be evaluated by inspecting the service' past executions, clients' feedbacks, system vendors' profile, the SLA, etc. In addition, formula (7.1) does not handle the varied failure probabilities of different component services and the scenario where one quality degradation level is likely to occur more than another. For example, s_1 in *VOVS* may fail more frequently than s_2 when processing a same amount of service requests, and a 20% response time degradation more likely to occur in s_1 than a 40% degradation. However, the probability distributions of service failure and quality degradation are domain specific. The proposed approach can be adjusted to deal with specific requirements, e.g., using a weighting process based on probabilities to differentiate the contribution of each service failures to the system quality degradation, which does not influence the effectiveness of our approach either.

In SMC4MTS, we adopt response time and throughput as example quality dimensions to calculate the quality-based criticality of a component service. In terms of reliability, all the component services are equally important in contributing to the overall system quality according to the quality aggregation functions presented in Table 4.1. The cost of a service does not change upon anomalies. However, reliability and cost can play important roles in criticality calculation in scenarios where more complicated reliability model and pricing model are expected, such as the reliability based on the structure of an SBS, the consideration of the cost for fixing a service anomaly, etc. Please note that some criticality measures other than the one based on sensitivity analysis in this thesis, such as reliability-based criticality discussed in [73][74][75], can be used for service criticality calculation in a similar manner.

7.1.2 Phase 2: tenant-based criticality computation

As introduced in Section 2.2.2, a multi-tenant SBS such as *VOVS* often contains multiple execution plans that serve different tenants with different functional requirements. All the tenants can access the SBS through service requests simultaneously, and the SBS is executed upon the receipt of each service request. However, some component services are not shared by all the tenants, such as the Video on Demand service and Live Event service in *VOVS*. From the system vendor's perspective, these services are not equally critical and resources should be allocated to the services that serve more tenants and respond to more service requests. Given the fact that the quality degradation of a component service may only affect the tenants sharing the service, the tenant-based criticality of a service is calculated by (7.2):

$$cr^T(s_i) = \frac{\tau(s_i)}{\tau(\mathbb{S})} \times \frac{\xi(s_i)}{\xi(\mathbb{S})} \tag{7.2}$$

where $\tau(s_i)$ is the number of tenants sharing service s_i, $\tau(\mathbb{S})$ is the number of all tenants served by the SBS, $\xi(s_i)$ is the average number of service requests that s_i processes per unit of time, e.g., per second, and $\xi(\mathbb{S})$ is the average number of service requests that SBS $\mathbb{S}$ processes within the same unit of time. For example, we assume that $\tau(VOVS)$ is 100, $\tau(s_2)$ is 35, $\xi(VOVS)$ is 100 req/s, and $\xi(s_2)$ is 9 req/s, then we can calculate $cr^T(s_2)$ to be 0.032. The numbers of tenants and service requests in (7.2) can be obtained by the system engineer through the analysis of tenants' business requirements as well as the corresponding SLA.

7.1.3 Phase 3: overall criticality computation

A component service may not be of the same criticality according to different criteria. SMC4MTS aims at providing an integrated solution to the formulation of monitoring strategy across multiple tenants of the SBS. Therefore, we calculate the overall criticality of a component service in this phase by combining its multi-dimensional quality-based criticality and tenant-based criticality, considering tenants' quality preferences.

First, due to the different methods that can be adopted when calculating quality-based criticality, we normalise the criticality value with (7.3) to eliminate the impact of different measurement units, where cr_p^{QX} and cr_p^{QN} are the maximum and minimum criticality values, respectively, of the component services for the p^{th} quality dimension in SBS $\mathbb{S}$, and $cr_p^{Q}(s_i)$ is the p^{th} quality-based criticality value of the i^{th} component service in the composition of $\mathbb{S}$.

$$crn_p^Q(s_i)=\begin{cases}\dfrac{cr_p^Q(s_i)-cr_p^{QN}(\mathbb{S})}{cr_p^{QX}(\mathbb{S})-cr_p^{QN}(\mathbb{S})} & \text{if } cr_p^{QX}(\mathbb{S})\neq cr_p^{QN}(\mathbb{S})\\ 1 & \text{if } cr_p^{QX}(\mathbb{S})=cr_p^{QN}(\mathbb{S})\end{cases} \tag{7.3}$$

Having been normalised, the multi-dimensional quality preferences of the tenants that share s_i, with the tenant-based criticality of s_i, are used to calculate the overall criticality of s_i with formula (7.4):

$$cr^O(s_i) = (\sum_{p=1}^{d} crn_p^Q(s_i) \times w_p^{ave}) \times cr^T(s_i) \tag{7.4}$$

where w_p^{ave} is calculated with (4.4) and denotes the average preferences for each quality dimension of s_i across all the tenants sharing s_i. Now the criticalities of the component services of an SBS have been calculated, which are used to rank those component services in the next phase.

7.1.4 Phase 4: component service ranking

Table 7.1: Criticality Computation and Service Ranking in *VOVS*

Component Services	Response Time (ms)	Throughput (req/sec)	cr_{rt}^{Q}	crn_{rt}^{Q}	cr_{tp}^{Q}	crn_{tp}^{Q}	$\tau(s_i)/\tau(\$)$	$\xi(s_i)/\xi(\$)$	cr^{T}	cr^{O}	Rank
s_1	90.7	40	0.33	1	0.84	0.92	1	1	1	0.959	1
s_2	69.8	22	0.02	0.06	0.05	0	0.35	0.09	0.032	0.001	9
s_3	56.6	92	0.15	0.45	0.09	0.04	0.75	0.73	0.548	0.134	5
s_4	40.0	46	0.02	0.05	0.34	0.33	0.75	0.73	0.548	0.104	6
s_5	35.9	59	0.01	0.02	0.21	0.19	0.75	0.73	0.548	0.056	7
s_6	83.5	33	0.23	0.67	0.91	1	0.75	0.73	0.548	0.457	2
s_7	73.8	63	0.20	0.59	0.19	0.16	0.75	0.73	0.548	0.206	4
s_8	7.4	57	0.01	0	0.28	0.27	0.60	0.18	0.108	0.014	8
s_9	16.3	68	0.06	0.17	0.30	0.29	1	1	1	0.228	3

Based on service criticality, the component services of an SBS can be ranked and the top k ($1 \le k \le n$) most critical services can be identified. Table 7.1 shows the criticality computation and service ranking for *VOVS*. For the demonstration purpose, we use q_{rt}^{dc}=200% and Δq_{rt}=50% for response time, and use q_{tp}^{dc}=80% and Δq_{tp}=20% for throughput. Take s_1 for example, its response time-based and throughput-based criticalities are 0.334 and 0.838 respectively which are calculated with (7.1) based on the quality values in Table 7.1. Then the quality-based criticalities of s_1 are normalised to 1 and 0.918 respectively with (7.3). The tenant-based criticality of s_1 is 1 which is calculated with (7.2) based on the service sharing and service requests distribution in *VOVS*, represented by $\tau(s_i)/\tau(\$)$ and $\xi(s_i)/\xi(\$)$ in Table 7.1. The overall criticality of s_1 is 0.959 which is calculated using (7.4) with equal preferences for response time and throughput across all tenants.

For all the component services in *VOVS*, the phase-by-phase outcomes of criticality calculation and service ranking from Phases 1 to 3 as described in Sections 7.1.1 to 7.1.3 are presented in Table 7.1. The ranking results show that in this scenario s_1 has the highest criticality and should be given the highest priority in the allocation of service redundancy.

7.1.5 Phase 5: local monitoring strategy generation

The monitoring strategy of a component service is called local monitoring strategy in our approach, which is implemented according to the configuration of corresponding monitoring parameters. Various domain-specific monitoring parameters exist. In SMC4MTS, we consider the following three common parameters as examples [3]:

1. Number of monitors: A monitor is an entity that executes the monitoring actions. Different monitors can be used to monitor the same component service for different quality dimensions, e.g., one monitor for each quality dimension.

2. Monitoring frequency: This parameter indicates how often the status of a component service is checked. Usually, a higher monitoring frequency allows more timely detection of runtime anomalies.

3. Monitoring granularity: This parameter indicates the extent to which a monitoring target is inspected. For example, a message can be parsed to different protocol layer for monitoring purpose. In general, a finer monitoring granularity means a higher monitoring accuracy.

The monitoring benefit, resource cost and system overhead caused by a monitoring strategy are described as follows:

1. Monitoring benefits (B): The monitoring benefits are domain-specific and can be evaluated in different metrics, e.g., the reduced time of business interruption, the avoided penalty caused by SLA breach, etc. In this thesis we consider the benefits in terms of quality, which can be quantified as an overall score that represents the avoided quality degradation in monitored

quality dimensions, such as response time and throughput. In general, a high monitoring benefit is achieved via timely and accurate detection of runtime anomalies.

2. Resource cost (R): Monitoring a service requires software and hardware resources, and sometimes human labour, which incurs resource cost. Usually, frequent and fine-grain monitoring results in high resource cost.

3. System overhead (O): Monitoring also incurs system overhead, such as occupying CPU and taking up memory space, which can impose negative impacts on the quality of an SBS. Frequent and fine-grain monitoring usually causes high system overhead.

Different combinations of monitoring parameters can generate different monitoring benefit, resource cost and system overhead, leading to different trade-offs between them. Moreover, the monitoring benefits may not always increase with the increase of allocated monitoring resources, i.e., it is not always the case that more resources can generate more monitoring benefits. Because the increase in monitoring benefits obtained by increasing monitoring resource slows down at some point, where the allocated monitoring resources continue to increase but generate only insignificant or even no benefits at all.

In order to evaluate and compare the benefits of monitoring a component service s_i from the perspective of the entire SBS, we weight the monitoring benefits with the overall criticalities of the corresponding component services by (7.5):

$$B^{cri} = cr^{O}(s_i) \times B \tag{7.5}$$

where $cr^{O}(s_i)$ is the overall criticality of s_i calculated with (7.4). Then, a local monitoring strategy is represented by (7.6):

$$ms_{i,j} = (N_{i,j}, F_{i,j}, G_{i,j}, B^{cri}_{i,j}, R_{i,j}, O_{i,j}) \tag{7.6}$$

where $ms_{i,j}$ is the j^{th} monitoring strategy for the i^{th} component service, notations $N_{i,j}$ to $O_{i,j}$ represent the number of monitors, monitoring frequency, monitoring granularity, monitoring benefit, resource cost, and system overhead respectively.

Based on the tenants' requirements for the system quality, the system vendor has its own preferences for the monitoring benefit, resource cost and system overhead in the monitoring strategies, which can be represented by individual weights with (7.7):

$$w_B + w_R + w_O = 1 \tag{7.7}$$

In order to handle the trade-off between the monitoring benefit, resource cost and system overhead, we propose *monitoring utility* in this thesis, which is defined as follows:

Definition 7.2. ***Monitoring Utility***. Denoted by *mu*, monitoring utility is used to represent the generalised preferences for the monitoring strategies.

The monitoring utility of a local monitoring strategy is calculated by (7.8):

$$mu(ms_{i,j}) = w_B \times B_{i,j}^{norm} + w_R \times R_{i,j}^{norm} + w_O \times O_{i,j}^{norm} \tag{7.8}$$

where $B_{i,j}^{norm}$, $R_{i,j}^{norm}$ and $O_{i,j}^{norm}$ are the normalised monitoring benefit, resource cost and system overhead, respectively, of monitoring strategy $ms_{i,j}$ (the normalisation method used can be found in [20]). The monitoring utility of an SBS that contains n component services is calculated by (7.9):

$$mu(\$) = \sum_{i=1}^{n} mu(ms_{i,j}), i \in [1, n] \tag{7.9}$$

where $ms_{i,j}$ is the monitoring strategy implemented (if exists) for the i^{th} component service in the SBS.

Similar to the service utility introduced by many works in service selection [13][20], monitoring utility can be used as a metric to evaluate the optimality of a monitoring strategy.

7.1.6 Phase 6: global monitoring strategy determination

A global monitoring strategy can be formulated by determining the local monitoring strategies for certain critical component services within the system vendor's monitoring budget. The constraints for monitoring parameters such as monitoring granularity must be fulfilled, and the optimisation goal of monitoring, e.g., maximised monitoring utility (see Definition 7.2), must be achieved in the meantime. This can be modelled as a Constrained Optimisation Problem (COP). Thus we formulate a COP to find an optimal solution to global monitoring strategy determination.

Considering an SBS $\mathbb{S}$ that consists of n component services, and each of them is associated with m local monitoring strategies. A set of 0-1 integer variables $x_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,m}\}$ is created for component service s_i. $x_{i,j}$ with value of 1 means that the j^{th} monitoring strategy for the i^{th} component service is selected for the monitoring strategy of the SBS. The process for solving the COP model is to assign a value of 0 or 1 to each $x_{i,j}$ and ensure that all constraints are fulfilled and the objective of the COP is achieved. This is essentially a knapsack problem [21], and thus is NP-complete.

The COP model is formulated with (7.10) to (7.13):

$$Objective : Maximising(mu(\mathbb{S})) \tag{7.10}$$

$$\sum_{j=1}^{m} x_{i,j} \leq 1, \quad 1 \leq i \leq n, x_{i,j} \in \{0, 1\} \tag{7.11}$$

$$R(\mathbb{S}) \leq C_b \tag{7.12}$$

$$N_{i,j} \prec C_N, R_{i,j} \prec C_R, G_{i,j} \prec C_G \tag{7.13}$$

where constraint family (7.11) ensures that at most one local monitoring strategy can be selected for a component service, constraint family (7.12) guarantees that the resource cost of monitoring strategy does not exceed the system vendor's monitoring budget, constraint family (7.13) ensures that the constraints for the monitoring parameters of any selected local monitoring strategy are fulfilled, and again symbol $<$ means "as good as or better than".

The optimal global monitoring strategy can be found by solving the above COP with the Integer Programming (IP) technique [20]. With this monitoring strategy, the quality degradation of the SBS upon runtime anomalies can be alleviated significantly.

7.2 Experimental evaluation

We conduct extensive experiments to evaluate SMC4MTS by comparing it with other representative approaches. Section 7.2.1 introduce the metrics for evaluation. Section 7.2.2 introduces the setup of the experiments. The effectiveness and efficiency of SMC4MTS are evaluated in Sections 7.2.3 and 7.2.4 respectively.

7.2.1 Evaluation metrics

The effectiveness is measured in four metrics as follows:

- Success rate: If a component service is monitored, the runtime anomalies of this service can be detected and the monitoring is regarded as successful. The success rate is represented by the ratio between the number of anomalies detected successfully and the total number of anomalies in the SBS.
- Monitoring utility: Based on the COP model mentioned in Section 7.1.6, monitoring utility represents the optimality of a monitoring strategy. This is evaluated from the perspectives of entire SBS.
- Quality degradation: The aim of monitoring is to guarantee the overall quality of an SBS. In the experiments the percentage of quality degradation is inspected and compared when runtime anomalies occur and a monitoring strategy is in place, which is represented by the ratio between the quality degradation upon anomalies and the original quality of the SBS.

- Affected tenant percentage: When anomalies occur to a monitored component service, the negative impacts, e.g., quality degradation, to the tenants that share the service can be avoided. The affected tenant percentage is the proportion of tenants affected by anomalies to all tenants that share the SBS.

The efficiency is measured in two metrics as follows:

- System overhead: The system overheads incurred by monitoring are simulated with numerical values in the experiments, which are "the lower, the better". The system overhead of a global monitoring strategy is the sum of the system overheads incurred by the selected local monitoring strategies.
- Computation time: The average computation time for formulating a monitoring strategy for an SBS.

7.2.2 Experimental setup

A prototype of SMC4MTS is implemented in Java with JDK 1.6.0. The widely used IBM CPLEX (v12.6) is again employed to find the solutions to the COPs. Four representative approaches, similar to [3], are implemented to be compared with SMC4MTS:

1. *Non-Monitoring*: Denoted by *NonMon*, no monitoring strategy is formulated for the SBS in this approach.
2. *Random-Monitoring*: Denoted by *RandMon*, the component services and the corresponding local monitoring strategies are selected randomly.
3. *Mu-Greedy-Monitoring*: Denoted by *MuMon*, the component services are randomly selected and the corresponding local monitoring strategies are ranked based on their monitoring utilities. The monitoring strategies with the highest monitoring utilities are selected.
4. *Cri-Mu-Greedy-Monitoring*: Denoted by *CriMuMon*, the component services are first ranked based on their criticalities, and then the monitoring strategy

with the highest monitoring utility is selected for the corresponding component service with the highest criticality.

In order to evaluate SMC4MTS more comprehensively, we extend the SBS *VOVS* presented in Section 2.2.1 by randomly generating service compositions with n tasks. The component services in an SBS instance are randomly generated with two quality dimensions: response time and throughput. The tenants and their service sharing in an SBS are also generated randomly. The monitoring parameters are generated randomly to simulate m different local monitoring strategies for each component service. The monitoring benefit, resource cost and system overhead are generated randomly according to the rules in their introductions in Section 7.1.5. We use n=100 and m=20 as default setup unless explicitly stated otherwise. For the sake of simplicity without losing generality, average preferences for the monitoring benefit, resource cost and system overhead are used. The monitoring budget of the system vendor is a numeric value generated randomly based on the average resource cost of local monitoring strategies. It is used as an important constraint in the experiments to evaluate the cost effectiveness of all approaches.

In order to simulate a volatile environment, we inject runtime anomalies randomly to an SBS based on a fixed failure rate in each set of experiments. For example, given an SBS that consists of 100 component services and a failure rate of 0.1, we randomly select ten component services to fail. This is a similar approach as described in [26]. For the purpose of simplicity and consistency without losing generality, we assume that when anomalies occur to a component service that is not monitored, the quality of the service degrades according to a unified quality degradation coefficient (see Definition 7.1), which is calculated by $q_p^{dc} = 1-(2)^{-h}$ and $q_p^{dc} = (2)^h, h \geq 1$ for positive and negative quality dimensions respectively. For example, when h is 1, the quality degradation coefficients for throughput and response time are 50% and 200% respectively. In order to simulate the varied delays before adaptation caused by different monitoring strategies, when runtime

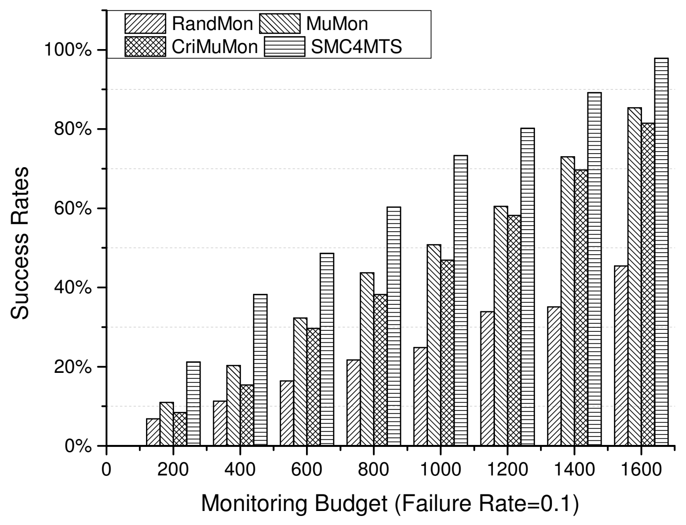


Figure 7.4: Comparison in success rates.

anomalies occur to a monitored component service, we randomly generate multi-dimensional quality degradations to the service, which are more serious with a lower monitoring frequency or coarser monitoring granularity.

All the experiments are again conducted on a machine running Windows 7x64 Enterprise with Intel(R) Core (TM) i5-4570 3.2 GHz CPU and 8 GB RAM. The experimental results are collected, averaged and compared from 100 test instances.

7.2.3 Effectiveness evaluation

7.2.3.1 Comparison in success rates

We vary the monitoring budget from 200 to 1600 in steps of 200. Similar results are observed when the failure rate varies from 0.1 to 0.4. Taking failure rate of 0.1 as example, as shown in Fig. 7.4, the success rates obtained by all approaches increase (NonMon is not applicable) because more component services are covered when the monitoring budget increases. However, SMC4MTS achieves the highest success rates across all cases, which increase from 21.2% to 99% as the monitoring budget increases from 200 to 1600. On average SMC4MTS outperforms MuMon and CriMuMon by 16.5% and 20.1% respectively. RandMon, as expected, obtains the lowest success rates, which increases from 6.8% to 45.4%.

7.2.3.2 Comparison in system monitoring utility

We compare the system monitoring utility to evaluate the optimality of monitoring strategies formulated by different approaches except NonMon which is not applicable. We change the monitoring budget from 200 to 1600 in steps of 200. Again we demonstrate the results of the experiments with a failure rate of 0.1 as example in Fig. 7.5. The utilities obtained by all approaches increase with the increase in the monitoring budget. SMC4MTS outperforms other approaches in all cases. Its system monitoring utility increases from 13.8 to 61.6 and beats MuMon by approximately 10% on average. CriMuMon is slightly outperformed by MuMon across all cases. RandMon, again, achieves the lowest monitoring utility among all approaches, as low as 18.0 when budget is 1600.

7.2.3.3 Comparison in quality degradation

In this series of experiments, with all the five approaches, the average degradation percentage of response time and throughput of the SBSs are compared with failure rate increasing from 0.1 to 0.4. Similar phenomena can be seen in experiments with different failure rates. However, the response time degrada-

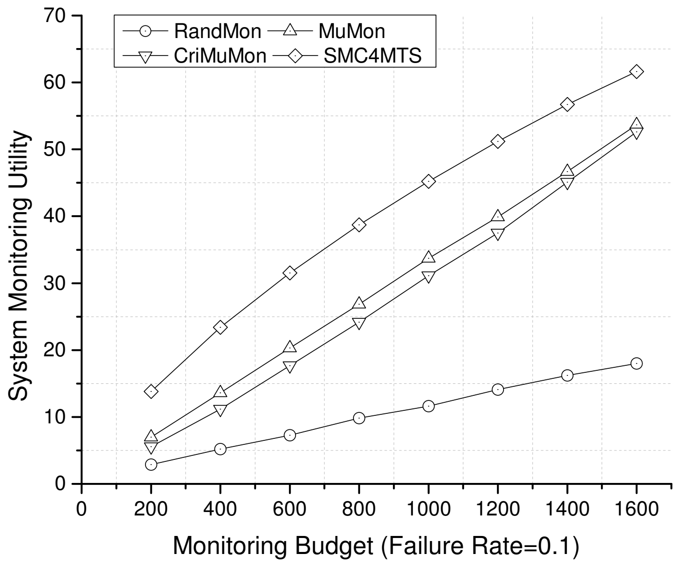


Figure 7.5: Comparison in system monitoring utility.

tions obtained by all approaches increase when the failure rate becomes higher. Again we use failure rate of 0.1 as an example. The comparison in response time degradation is shown in Fig. 7.6. NonMon shows much more serious degradation in response times (330% with slight fluctuations) than other approaches because of the absence of monitoring strategies. The response time degradations obtained by other four comparing approaches decrease gradually when the monitoring budget increases. SMC4MTS outperforms the comparing approaches in all cases and beats CriMuMon, MuMon and RandMon by approximately 47.5%, 66.4% and 72% on average respectively.

Similar comparison results can be seen for throughput degradation in Fig. 7.7. The throughput degradation demonstrated by all approaches except NonMon decrease when the monitoring budget varies from 200 to 1600. SMC4MTS achieves the best performance across all cases and beats CriMuMon by approximately 12% on average. MuMon shows a significant decrease in throughput degradation (from 89% to 44%) when the monitoring budget increases from 800 to 1600. NonMon and RandMon obtain the similar throughput degradation, which remains at approximately 93% when monitoring budget varies.

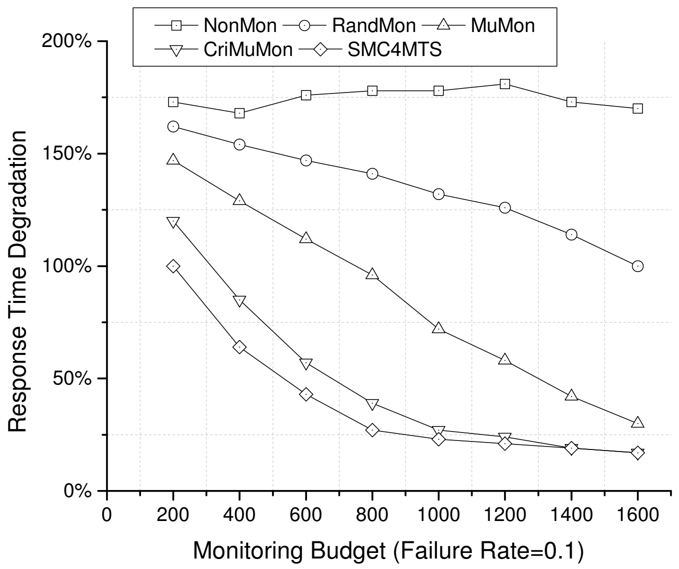


Figure 7.6: Comparison in response time degradations.

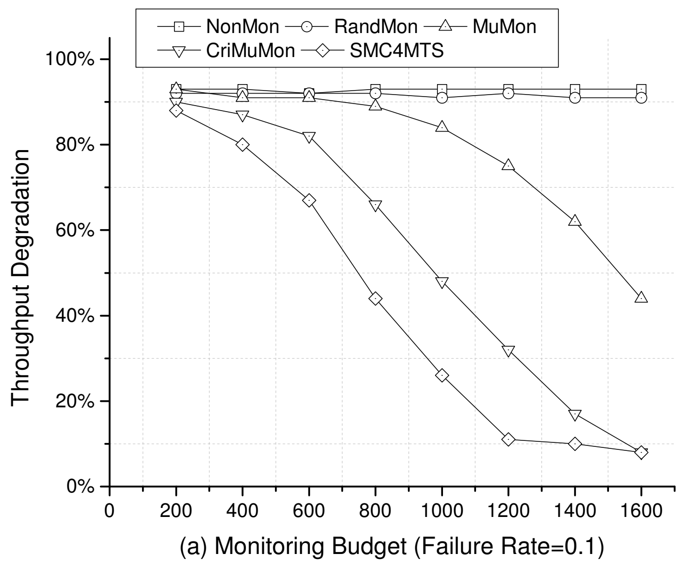


Figure 7.7: Comparison in throughput degradations.

7.2.3.4 Comparison in affected tenant percentages

We evaluate the effectiveness of SMC4MTS in supporting multi-tenancy in this series of experiments. The results with a failure rate of 0.1 are presented in Fig. 7.8. The affected tenant percentages obtained by all approaches increase when the environment becomes more volatile. By considering tenants' service sharing in criticality calculation, SMC4MTS reduces the affected tenant percentage significantly from 57% to 2% when the budget increase to 1600. CriMuMon outperforms NonMon, RandMon and MuMon by approximately 39%, 21%, and 6% on average respectively.

7.2.4 Efficiency evaluation

7.2.4.1 Comparison in system overhead

In this series of experiments we evaluate the system overheads incurred by the monitoring strategies. As shown in Fig. 7.9, all the four approaches show increases in system overheads when monitoring budget increases. RandMon formulates the monitoring strategies with the highest system overheads. SMC4MTS incurs medium overheads, which are slightly higher than those of MuMon and

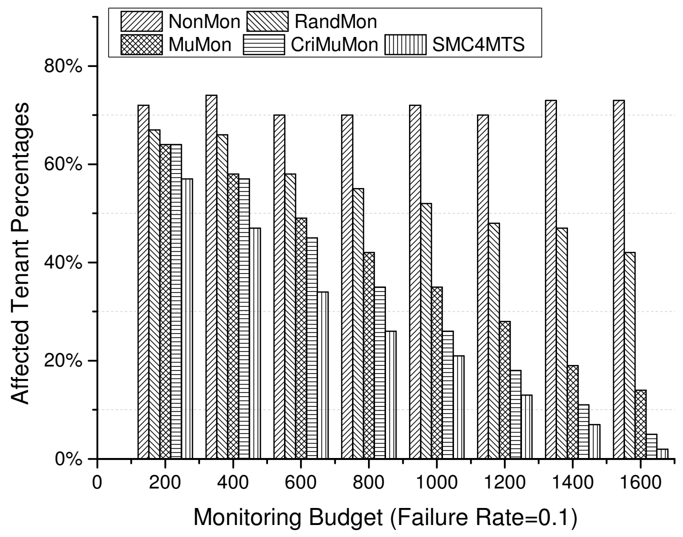


Figure 7.8: Comparison in affected tenant percentages.

CriMuMon.

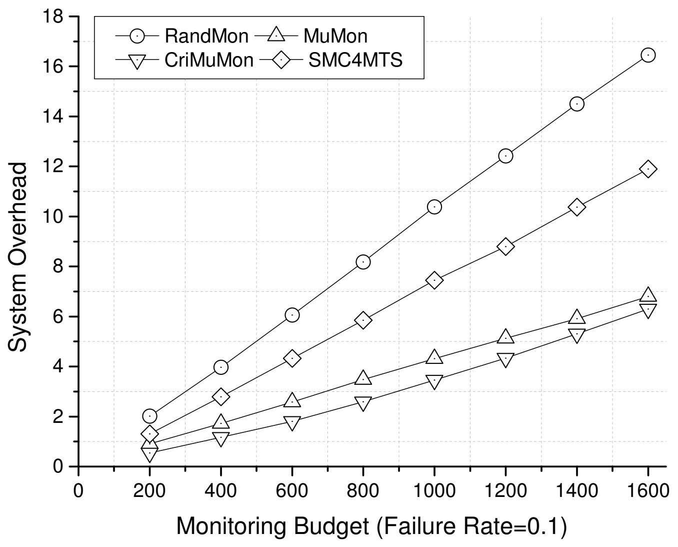


Figure 7.9: Comparison in system overheads.

7.2.4.2 Comparison in computation time

First, we collect and average the computation time of formulating monitoring strategies by varying the number of tasks in an SBS from 20 to 100 in steps of 20. The time for calculating service criticality is also evaluated in the meantime. The results are shown in Fig. 7.10. SMC4MTS consumes the most computation time (up to 23.3ms) due to the fact that finding an optimal solution is an NP-complete

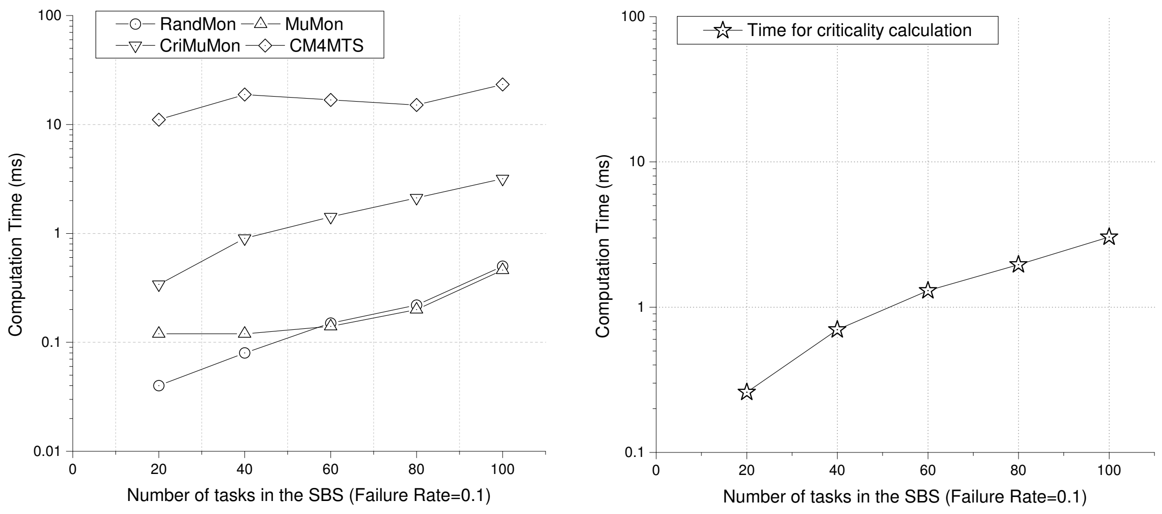


Figure 7.10: Computation time vs number of tasks.

problem. Approximately 8% of the consumed time on average of SMC4MTS is used for criticality calculation. The computation time consumed by CriMuMon is slightly higher than those of MuMon and RandMon, which are approximately 0.2ms. Next, we evaluate the computation time of the four approaches by changing the number of local monitoring strategies from 200 to 1000 in steps of 200. The COP scales up when more monitoring strategies are involved. As shown in Fig. 7.11, the time used by SMC4MTS increases from 309ms to 1625ms, consuming more time than the other approaches across all cases. CriMuMon, MuMon and RandMon consume 13.4ms, 10.3ms and 71.3ms on average. Overall, the experimental results demonstrate that the efficiency of SMC4MTS is acceptable in, if not all, most scenarios.

7.3 Discussion

Identifying the critical component services in an SBS is a challenging but important issue in service monitoring. In this chapter, the multi-dimensional quality used in service criticality calculation is linear and deterministic. The non-linear quality constraints and parameters are not considered in the COP model. For example, a quadratic expression can be used as quality constraint or optimisa-

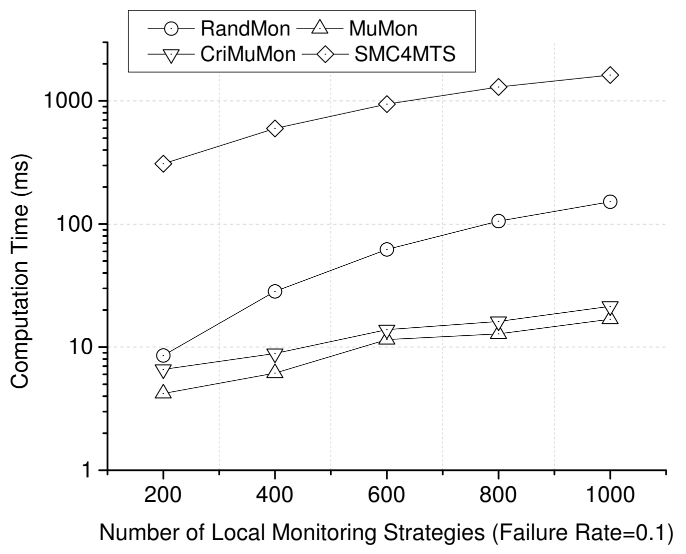


Figure 7.11: Computation time vs number of local monitoring strategies.

tion objective of the COP, and the values of a quality dimension can be nominal, such as the reputation of a service. However, this is not a significant limitation to our approach. On one hand, linear quality constraints and dimensions have been the same basis of many other research efforts, such as the work presented in [13][14][20][21][60]. Considering only the linear quality constraints and dimensions does not influence the comparison in the experiments between SMC4MTS and existing approaches. On the other hand, non-linear quality constraints and dimensions can be handled with various techniques [93], e.g. Sequential Quadratic Programming [94], or transformed into the linear expressions [95].

The service criticality proposed in this chapter can also be used in other lifetime stages of quality management such as service adaptation. In Chapter 9, we will explore the usage of service criticality in formulating cost-effective fault tolerance strategies for multi-tenant SBSs.

7.4 Summary

In this chapter, we introduce SMC4MTS, a novel approach for formulating cost-effective monitoring strategies for multi-tenant SBSs. Service criticality is employed to prioritise the component services in monitoring resource allocation. Monitoring utility is proposed to handle the trade-off between monitoring benefit, resource cost and system overhead. The optimal monitoring strategy of an SBS can be formulated based on a Constraint Optimisation Problem (COP) model. Extensive experiments show that SMC4MTS can significantly reduce the quality degradation of an SBS upon anomalies.

Part III

Quality management for service adaptation

Chapter 8

Service adaptation based on LSH

As discussed in Section 2.2.2, in order to achieve efficient service adaptation upon runtime anomalies, a quick service re-selection that replaces only the anomalous ones with alternative services with equivalent functionalities is much more practical than an overhaul of the entire SBS. Therefore, it is critical but challenging to select the appropriate services for rapid runtime system adaptation for multi-tenant SBSs. In this chapter, we introduce Service Adaptation based on LSH (Locality-Sensitive Hashing) for Multi-Tenant SBSs (SAL4MTS). In our approach, given a faulty service, LSH is firstly used to find its approximate nearest neighbour from the candidate services, which has the most similar quality performance with it, and thus the tenants' quality requirements can still be satisfied with a high probability after system adaptation. This chapter is based on my paper [19].

This chapter is organised as follows. Section 8.1 introduces the SAL4MTS approach including the preliminary knowledge of LSH. Then Section 8.2 presents the evaluation of SAL4MTS in terms of effectiveness and efficiency. Section 8.3 discusses how LSH can be used for other scenarios of quality management for SBSs. Some premises of SMC4MTS are also discussed. Finally, Section 8.4 summarises this chapter.

8.1 SAL4MTS approach

8.1.1 LSH preliminary

LSH is a well-known randomised algorithm that allows one to quickly find similar entries in a high-dimensional large dataset [96]. It can reduce the dimensionality of high-dimensional data. In this research, we seek to achieve high efficiency in runtime service adaptation. LSH can sufficiently capture our need to find a service similar to the faulty service in terms of multi-dimensional quality values for quick service replacement. The basic idea of LSH is to hash similar data points into the same "buckets" with high probability [97]. It has much in common with data clustering and nearest neighbour search and has been widely used in a variety of areas, such as near-duplicate detection, hierarchical clustering, image similarity identification, etc.

Let $\mathbb{R}^d$ be a d-dimensional space, and $\mathcal{H}$ be a family of hash functions mapping $\mathbb{R}^d$ to some universe U. $d_1 < d_2$ are two distance values and $p_1 > p_2$ are two probability values. Given any two points $x, y \in (\mathbb{R})$, we choose a hash function h from $\mathcal{H}$ randomly and observe the probability of $h(x) = h(y)$. The family $\mathcal{H}$ is called (d_1, d_2, p_1, p_2)-sensitive if it satisfies the following conditions.

- If $||x - y|| \leq d_1$ then $Pr_{\mathcal{H}}[h(x) = h(y)] \geq p_1$
- If $||x - y|| \geq d_2$ then $Pr_{\mathcal{H}}[h(x) = h(y)] \leq p_2$

where $h(x) = h(y)$ means x and y are similar objects and hashed into the same bucket.

In recent years, a variety of LSH families have emerged with different distance metrics, e.g., Hamming distance [98] and Jaccard distance [99]. Euclidean distance, as a metric widely used in the area of service computing to find the similar items, is used in this thesis, and l_2 LSH family based on 2-stable distribution, i.e., Gaussian distribution, is adopted. The hash function is given by

$$h^{a,b}(\vec{v}) = \lfloor \frac{\vec{a} \times \vec{v} + b}{w} \rfloor \tag{8.1}$$

where $\vec{v}$ is a query point in a d-dimensional space ($\mathbb{R}$), and $\vec{a}$ is a vector with components that are randomly selected from a Gaussian distribution. b is a random variable that uniformly distributes in $[0, w]$, and w is the width of each bucket.

8.1.2 Service adaptation based on LSH

Our approach of service adaptation based on LSH consists of two phases: build-time preprocessing and runtime system adaptation.

8.1.2.1 Phase 1: build-time preprocessing

In SAL4MTS, a service is regarded as a point in the d-dimensional Euclidean space $\mathbb{R}^d$ and is described by a vector that uses multi-dimensional quality of service as the components. For service S with d-dimensional quality, the vector generated is described by (8.2):

$$v_s =< q_1, ..., q_k, ..., q_d > \tag{8.2}$$

where q_k is the $k^(th)$ quality value of service S. Usually, services with similar quality values should stay close in R^d. An example is shown in Fig. 8.1, where two kinds of quality are considered: cost and response time. In such a 2-dimensional Euclidean space, the distance between S_1 and S_2 is close and these two services will fall into the same bucket with high probability if hashed accordingly, due to their similar quality performance. If S_1 is selected for service composition and failed at runtime, as the nearest neighbour of S_1, S_2 is potentially the best candidate for service replacement.

In order to facilitate rapid system adaptation at runtime, we build the hash

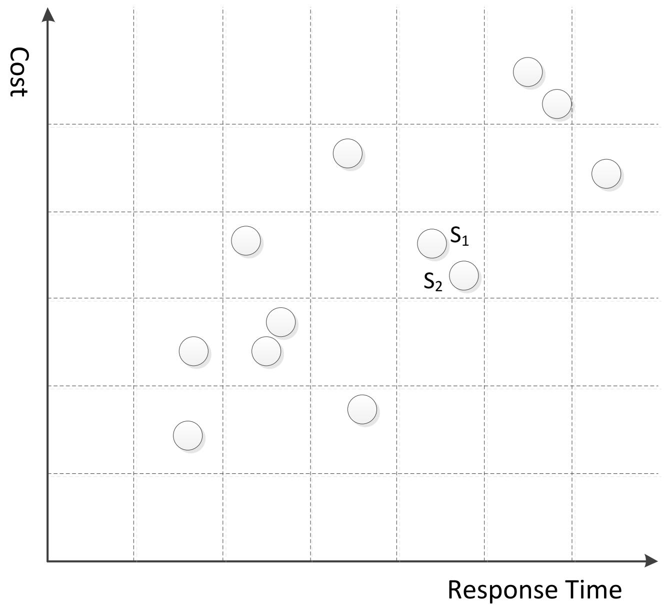


Figure 8.1: Services in 2-dimensional Euclidean space.

tables at build-time. The hash tables are built in two steps:

1. Randomly choose K functions $h_k, k = 1, \ldots, K$, from the LSH family $\mathcal{H}$ to generate the combined hash key of a service, by concatenating the K single hash values together, i.e., $g = (h_1, h_2, \ldots, h_K)$. Combined hash keys are used to build a hash table with respect to the chosen hash functions, which can significantly reduce the error of putting distant candidates into the same buckets.

2. Repeat Step 1 multiple times to build L hash tables. Multiple hash tables are employed to mitigate the problem that combined hash keys are possibly too strict to allow some buckets in a single hash table to have enough similar services. When querying approximate nearest neighbours of a service, we can seek simultaneously in the L hash tables and identify candidates from all query results.

As shown in Fig. 8.2, when the hash tables are constructed, all the candidate services are hashed into one or more hash tables, which can be used at runtime to quickly find the candidate services with similar quality performance as the faulty

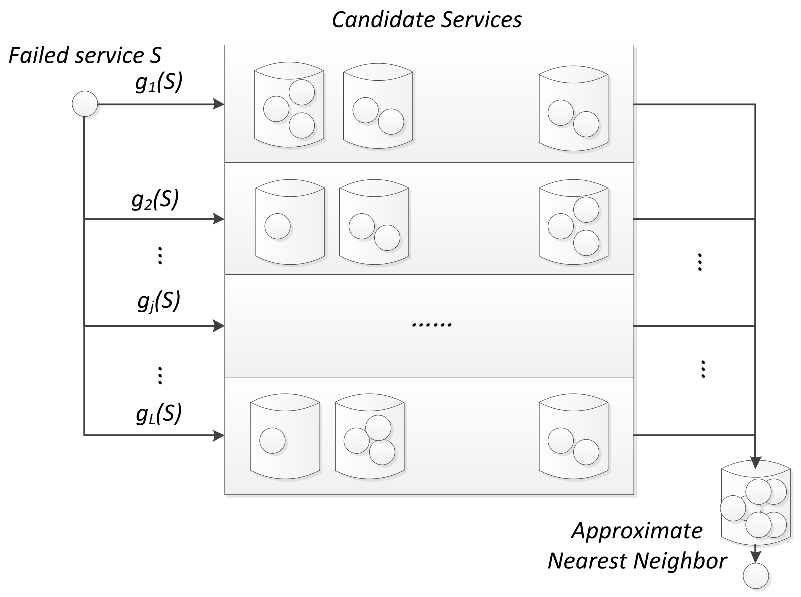


Figure 8.2: Query process of candidate services.

service. When a service joins or leaves a service class, or its quality changes, we only need to process that service without rebuilding all the hash tables. The time complexity for preprocessing is $O(nLKt)$, where n is the number of points, L is the number of hash tables, K is the number of hash functions, and t is the time to evaluate a hash function on an input point. The space complexity of LSH algorithm used in this thesis is $O(nL)$. The query time complexity is $O(L(Kt + dnp_2^K))$, where d is the number of dimensions of the vector used for representing a point, p_2 is the maximum probability that two points are similar objects when their distance is more than a given threshold.

8.1.2.2 Phase 2: runtime system adaptation

At runtime, once the anomalies are detected in the service monitoring process, adaptation should be triggered immediately and automatically in order to guarantee the quality of SBS.

The query process for a given faulty service S is as follows:

1. Retrieve the services in the bucket $g_j(S)$ in the j^{th} hash table.

2. For each service in the bucket, compute the distance from S to it.
3. Rank the candidate services according to the distance calculated in Step 2 and report the nearest neighbour of S.

As shown in Fig. 8.2, for each faulty service, the approximate nearest neighbour is selected for adaptation, and then is deployed to replace the faulty service. The overall quality requirements of all tenants sharing the faulty service are evaluated by replacing the faulty service with selected candidate service in quality aggregations of corresponding execution plans. If the end-to-end quality cannot be guaranteed after service replacement, the approach introduced in Chapter 5 is used, i.e., a system re-optimisation process is triggered to expand the adaptation scope to other services that are running normally by creating a new service composition for the SBS.

8.2 Experimental evaluation

We have conducted extensive experiments to evaluate SAL4MTS in terms of effectiveness and efficiency. Section 8.2.1 introduces the metrics used in the evaluation. Then, Section 8.2.2 presents the experimental setup. The experimental results in effectiveness and efficiency are introduced in Section 8.2.3 and Section 8.2.4 respectively.

8.2.1 Evaluation metrics

The effectiveness is measured based on the success rates of service adaptation. If after service adaptation all tenant's quality requirements can still be satisfied, the service adaptation is regarded as successful. The efficiency is evaluated based on the computation time used for service adaptation at runtime and preprocessing at build-time.

8.2.2 Experimental setup

We implemented SAL4MTS in Java with JDK 1.6.0. We use the E2LSH (Exact Euclidean LSH) method, which is based on the Locality-Sensitive Hashing scheme described in [96].

The services used in our experiments are again created based on QWS [88]. Cost and throughput are randomly generated and added to each candidate service as two additional quality dimensions.

Similar to Section 5.2.2, we also use the method proposed in [13] to generate the quality constraints for the SBS randomly with different difficulty levels: easy, medium, and severe, which represent the difficulties in finding a solution to satisfy tenants' quality requirements.

For the purpose of comparison with other approaches for runtime system adaptation in terms of effectiveness and efficiency, we also implemented existing service replacement and system re-optimisation approaches as follows:

- *Random*: One service is selected randomly as the candidate to replace the faulty service. The system adaptation is regarded as unsuccessful if the end-to-end quality of the SBS cannot fulfil the requirements of tenants after service replacement.

- *Optimal-Global*: This approach originates from [22] and has been adjusted to realise system re-optimisation upon runtime anomalies. This approach recreates a new service composition from the scratch, and every service in the original composition may be replaced to fulfil tenants' quality requirements and achieve the optimisation goal, which is to maximise the system utility.

Same as before, the experiments are conducted on a machine running Windows 7x64 Enterprise with Intel(R) Core (TM) i5-4570 3.2 GHz CPU and 8 GB RAM. For each set of experiments, the results are collected, averaged and compared from 100 test instances.

8.2.3 Effectiveness evaluation

In order to evaluate the effectiveness and efficiency of SAL4MTS for runtime system adaptation, we simulate a volatile cloud environment for an SBS by injecting one anomaly to a randomly selected service each time, and then trigger the system adaptation. We change the difficulty level of tenants' quality constraints from easy to severe and compare SAL4MTS with the other two approaches described in Section 8.2.2 in terms of adaptation success rates. If the tenants' quality constraints can still be satisfied after system adaptation, we think that it is a successful system adaptation. As shown in Fig. 8.3, when the difficulty level of quality constraints changes from easy to severe, the success rates obtained by all the approaches decrease. SAL4MTS maintains the similar high success rate (about 87% or above) as the Optimal-Global approach when the difficulty level of quality constraints changes. With the lowest performance among the three approaches, the random approach also gains a better performance when the quality constraints are easy to satisfy. The results show that due to the advantage of LSH-based service adaptation in finding replacement services with the most similar quality values to the faulty services, the tenants' quality requirements can be satisfied with a high probability after system adaptation.

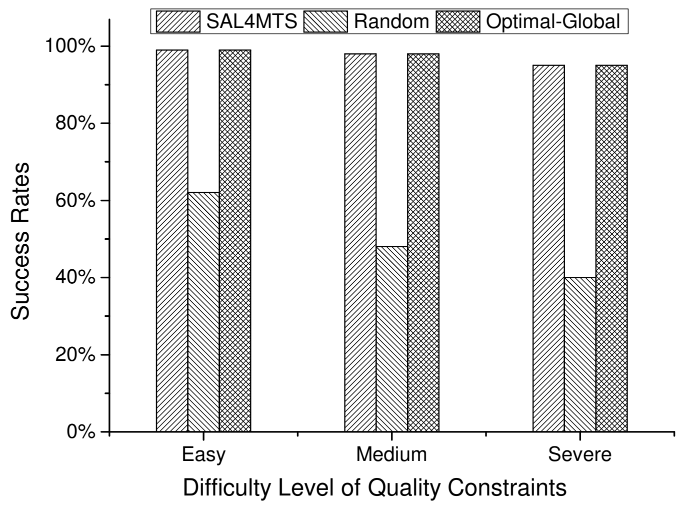


Figure 8.3: Adaptation success rates vs difficulty level of quality constraints.

Table 8.1: Comparison in System Optimality ($Utility \times Success\ Rate/Utility_{OPT}$)

Difficulty level of quality constraints	SAL4MTS	Random	Optimal-Global
Easy	99.8%	57.5%	100%
Medium	99.9%	45.8%	100%
Severe	99.9%	38.4%	100%

Then, we compare the system optimality obtained by these approaches. In this set of experiments, we use maximising the overall system utility as optimisation goal of the COP. Thus, a higher overall system utility represents higher system optimality. The utilities obtained by different approaches are combined with the corresponding adaptation success rates and compared with the optimal utilities obtained by Optimal-Global, which re-optimise the entire SBS with the highest utility. We fix the numbers of tenants and candidate services per service class at 100 and 1000 respectively, and vary the difficulty level of quality constraints from easy to severe. As shown in Table 8.1, the system optimality obtained by SAL4MTS is nearly the same as that of Optimal-Global (more than 99% across all the cases). Due to the low success rates, Random obtains the lowest system optimality, which is approximately 47.2% on average.

8.2.4 Efficiency evaluation

We run a series of experiments with the aim of comparing the efficiency of these approaches by their computation time. We fix the number of tenants at 100 and change the number of candidate services from 200 to 1000. The experiments with different difficulty levels of quality constraints show similar results that the computational overhead of SAL4MTS is slightly higher than the random approach but much lower than Optimal-Global. As presented in Fig. 8.4 where the difficulty level of quality constraints is medium, when the number of candidate services increases, the computation times of Optimal-Global and SAL4MTS also increase gradually, while SAL4MTS increases more slightly than Optimal-

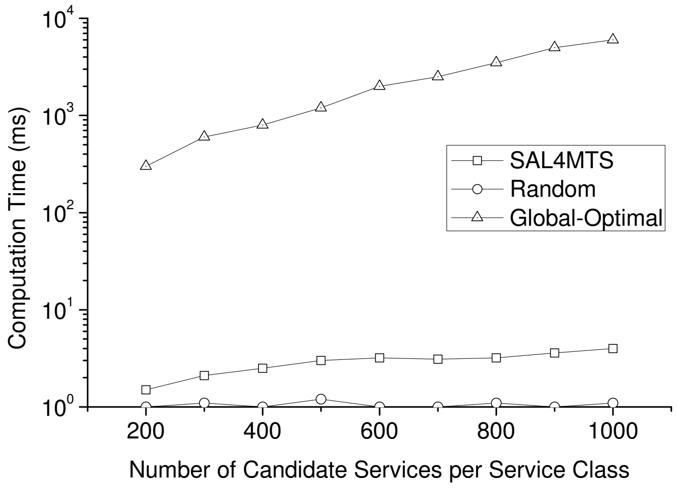


Figure 8.4: Adaptation success rates vs difficulty level of quality constraints.

Global. Random always remains the best performance when the number of services changes. However, its low success rate has been observed in Fig. 8.3. The results show that compared with the optimal approach, SAL4MTS is much quicker because of the high efficiency of the LSH algorithm used.

As discussed in Section 8.1.2.1, SAL4MTS builds index structure with hash tables for all the candidate services at build-time. In order to evaluate the efficiency of SAL4MTS in this preprocessing step, we conducted a series of experiments to investigate the computational overhead it introduces, which is measured by means of, again, the computation time. We focus on four parameters that affect the computation performance significantly in building index structure, including the numbers of candidate services per service class, quality dimensions, used hash tables and hash functions. The default setup used for these four parameters are (1000, 2, 2, 4). When one parameter varies, other parameters use the default values. As shown in Fig. 8.5 from (a) to (d), the increase of computation time can be seen across all the experiments. However, the time consumed for building index structure of candidate services is relatively very short (less than 12ms when there are 1000 candidate services with two-dimensional quality values, two hash tables and 32 hash functions). Compared with the advantage in computation time gained by our LSH-based approach over the approach of system re-optimisation,

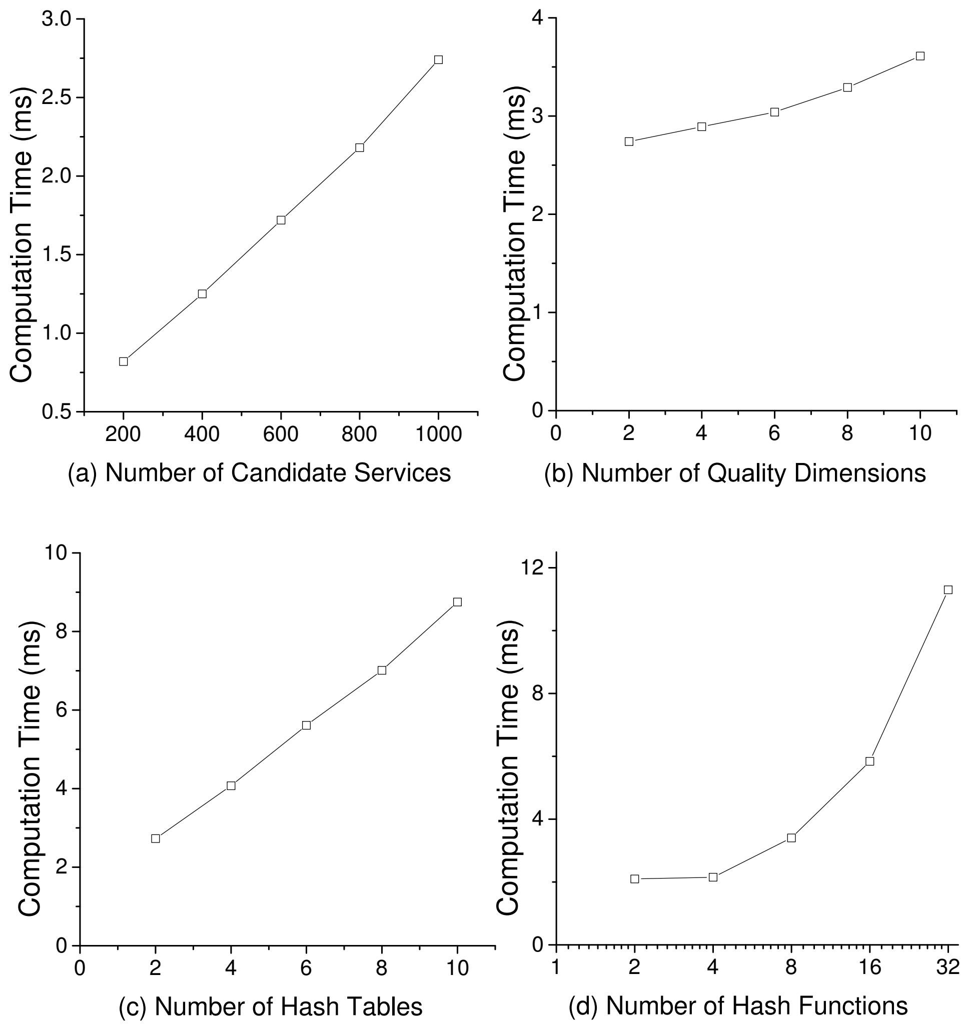


Figure 8.5: Adaptation success rates vs difficulty level of quality constraints.

the computation time consumed for building hash tables in preprocessing step is negligible.

The experiments on system adaptation show that due to the high computational time, optimal approaches are often impractical in the dynamic and volatile cloud environment. SAL4MTS is able to find appropriate replacement services efficiently with a high success rate.

8.3 Discussion

Locality-sensitive hashing (LSH) can efficiently reduce the dimensionality of high-dimensional data and is widely used in the nearest neighbour search. In this chapter, we innovatively use LSH in service adaptation at runtime by effectively and efficiently finding the similar replacement services in terms of quality values to the anomalous services. However, LSH-based service adaptation is local adaptation strategies, and once the tenants' quality requirements cannot be satisfied after adaptation, a system re-optimisation is unavoidable. Thus, we employ the SSR4MTS approach introduced in Chapter 5 to achieve this goal. Moreover, The efficiency of SAL4MTS can be further improved with the clustering techniques introduced in Chapter 5, which can categorise candidate services into different clusters and then the search space of finding replacement services can be further reduced.

8.4 Summary

In this chapter, we introduce SAL4MTS, a novel approach for service adaptation based on LSH for multi-tenant SBSs. Alternative services are found efficiently by LSH upon runtime anomalies to replace the faulty services in an SBS to continue fulfilling the quality requirements of tenants. Experimental results demonstrate that our LSH-based service adaptation approach can facilitate rapid runtime system adaptation at a high success rate with much less computational cost compared with the system re-optimisation approach.

Chapter 9

Fault tolerance based on criticality

As discussed in Section 1.2, fault tolerance is another promising way to provide quality guarantee for multi-tenant SBSs. By running component services and redundant services in certain redundancy mode, such as sequence or parallel, the impacts of runtime anomalies on the system quality of an SBS can be alleviated or mitigated. In order to support cost-effective fault tolerance within the system vendor's budget, the critical component services in an SBS must be identified and prioritised in the formulation of fault tolerance strategies. In this chapter, we introduce Fault Tolerance based on Criticality for Multi-Tenant SBSs (FTC4MTS), an approach for formulating cost-effective fault tolerance strategies for multi-tenant SBSs based on the service criticality introduced in Section 7.1. This chapter is based on my paper [80].

This chapter is organised as follows. Section 9.1 introduces the FTC4MTS approach. Then, Section 9.2 demonstrates the evaluation of FTC4MTS in terms of effectiveness and efficiency. Section 9.3 discusses some premises of FTC4MTS. Finally, Section 9.4 summarises this chapter.

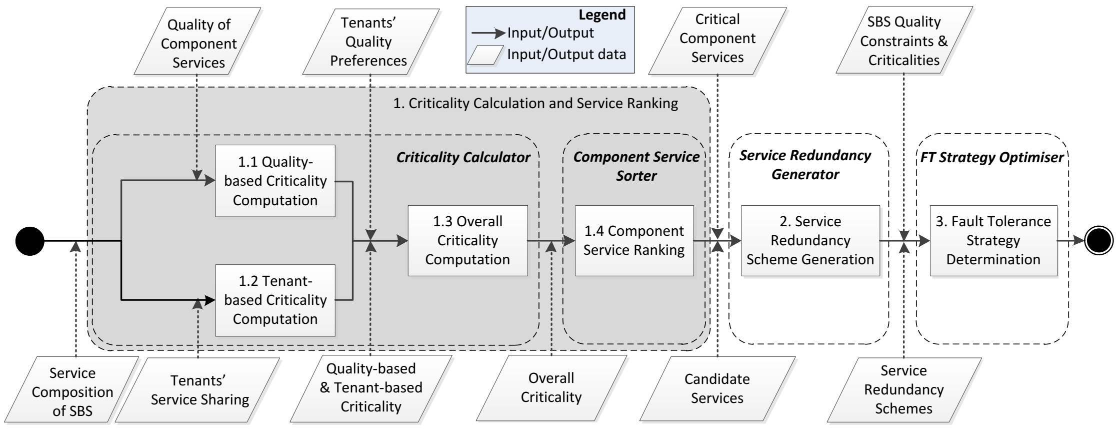

Quality of Component Services
Tenants' Quality Preferences
Legend
Input/Output
Input/Output data
Critical Component Services
SBS Quality Constraints & Criticalities
1. Criticality Calculation and Service Ranking
1.1 Quality-based Criticality Computation
Criticality Calculator
Component Service Sorter
Service Redundancy Generator
FT Strategy Optimiser
1.3 Overall Criticality Computation
1.4 Component Service Ranking
2. Service Redundancy Scheme Generation
3. Fault Tolerance Strategy Determination
1.2 Tenant-based Criticality Computation
Service Composition of SBS
Tenants' Service Sharing
Quality-based & Tenant-based Criticality
Overall Criticality
Candidate Services
Service Redundancy Schemes

Figure 9.1: System architecture of FTC4MTS.

9.1 FTC4MTS approach

As shown in Fig. 9.1, the system architecture of FTC4MTS includes four functional components. The method of calculating service criticality is the same as that in Chapter 7, and the roles of *Criticality Calculator* and *Component Service Sorter* have been introduced in Section 7.1. The functionalities of other two components are as follows:

- ***Service Redundancy Generator***: This component generates service redundancy schemes for component services, which consists of redundant services and redundancy mode.
- ***FT Strategy Optimiser***: This component determines the optimal fault tolerance strategy of an SBS, including the component services to allocate fault tolerance and the corresponding fault tolerance schemes.

FTC4MTS consists of three major phases, as shown in Fig. 9.1. In Phase 1, we calculate the criticalities for the component services, based on which, the component services are ranked. Redundancy schemes are then generated in Phase 2. Finally, in Phase 3, formulation of cost-effective fault tolerance strategy with service redundancy based on criticality is modelled as an optimisation problem, and the Integer Programming (IP) technique is used to find the solution. These three phases are detailed one by one in this section.

9.1.1 Criticality calculation and service ranking

In this phase, criticalities of the component services in the SBS are evaluated by analysing the quality of component services, multiple tenants' preferences for multi-dimensional system quality, and the service sharing across the tenants in the SBS. Overall service criticality of a component services is calculated based on quality-based criticality and tenant-based criticality. Then, the component services of the SBS are ranked based on their overall criticalities. The detailed process of criticality calculation has been introduced when we present our approach for criticality-based service monitoring in Chapter 7. Readers can refer to Section 7.1 for more information.

9.1.2 Service redundancy scheme generation

In this phase, FTC4MTS generates the *service redundancy scheme*, which is defined as follows:

Definition 9.1. ***Service Redundancy Scheme***. Denoted by *SRS*, service redundancy scheme is the combination of its member services (denoted by *MS*, including the component service and redundant services) and the corresponding redundancy mode (denoted by *RM*), such as sequence or parallel.

Various redundancy modes have been proposed to improve the reliability of critical applications in conventional software system [100][101][102]. In this thesis we consider two modes widely adopted: *Sequence* and *Parallel*, as shown in Fig. 9.2. Other modes can be included in FTC4MTS in a similar manner. The two modes function as follows:

1. *Sequence*. The services are executed sequentially from the first service until one of them completes successfully.

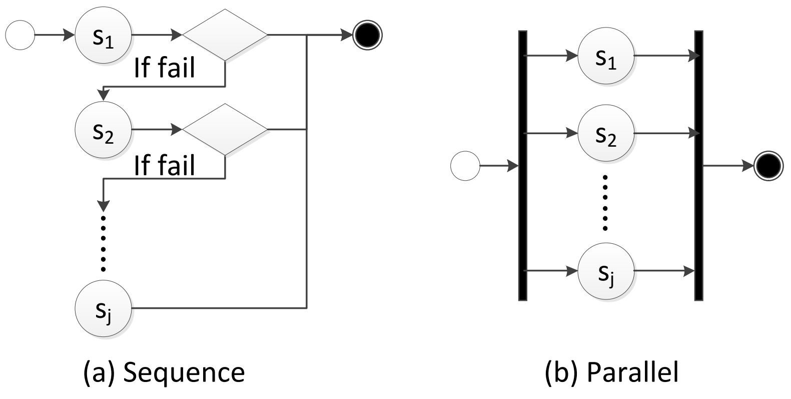


Figure 9.2: Service redundancy modes.

2. *Parallel*. The services are executed concurrently, and the first service that completes successfully determines the final result.

In both cases, a redundancy mode fails only if all the services fail. Similar to other work [13][20][21], in this thesis we assume that a task can be bound to any functionally equivalent candidate service in its corresponding service class, which means that these services are idempotent and the service requests are handled in an idempotent manner. In general, service redundancy schemes with *Sequence* mode have lower cost but may suffer from poor response time performance in volatile environments, while *Parallel* schemes can obtain faster response time at the cost of more resources consumed.

Let $SRS_{i,j}$ be the j^{th} service redundancy scheme of service class sc_i containing m_i candidate services. For $SRS_{i,j}^{sq}$ with $RM_{i,j} = Sequence$, and $MS_{i,j} = (s_{i,j_0}, s_{i,j_1}, ..., s_{i,j_m})$, $s_{i,j_0} = s_i$, $1 \leq m \leq m_i$, the quality aggregation functions are in (9.1):

$$\begin{aligned}
q_{ct}(SRS_{i,j}^{sq}) &= \sum_{f=0}^{m}(\sum_{r=0}^{f} q_{ct}(s_{i,j_r}) \times \prod_{r=0}^{f-1}(1 - q_{re}(s_{i,j_r}))) \\
q_{rt}(SRS_{i,j}^{sq}) &= \sum_{f=0}^{m} q_{rt}(s_{i,j_f}) \times \prod_{r=0}^{f-1}(1 - q_{re}(s_{i,j_r})) \\
q_{re}(SRS_{i,j}^{sq}) &= 1 - \prod_{r=0}^{m}(1 - q_{re}(s_{i,j_r})) \\
q_{tp}(SRS_{i,j}^{sq}) &= \sum_{f=0}^{m} q_{tp}(s_{i,j_f}) \times \prod_{r=0}^{f-1}(1 - q_{re}(s_{i,j_r}))
\end{aligned} \tag{9.1}$$

where $q_{re}(s_{i,j_r})$ is the reliability of the r^{th} service in $SRS_{i,j}^{sq}$, and service s_{i,j_r} will be executed only when all the preceding services in the same scheme have failed. Hence, the cost, response time and throughput of all the services in $SRS_{i,j}^{sq}$ should be summed, and each weighted by the probability that the service is executed.

The quality of service redundancy scheme $SRS_{i,j}^{pl}$ with the *Parallel* mode is calculated with (9.2):

$$\begin{aligned}
q_{ct}(SRS_{i,j}^{pl}) &= \sum_{r=0}^{m} q_{ct}(s_{i,j_r}) \\
q_{rt}(SRS_{i,j}^{pl}) &= \sum_{\lambda=1}^{2^m-1} \min_{s_{i,j_r}\in MS_\lambda} q_{rt}(s_{i,j_r}) \times \prod_{s_{i,j_r}\in MS_\lambda} q_{re}(s_{i,j_r}) \\
&\quad \times \prod_{s_{i,j_r}\notin MS_\lambda} (1-q_{re}(s_{i,j_r})) \\
q_{re}(SRS_{i,j}^{pl}) &= 1-\prod_{r=0}^{m}(1-q_{re}(s_{i,j_r})) \\
q_{tp}(SRS_{i,j}^{pl}) &= \sum_{\lambda=1}^{2^m-1} q_{tp}(\underset{s_{i,j_r}\in MS_\lambda}{\mathrm{SMRT}}(s_{i,j_r})) \times \prod_{s_{i,j_r}\in MS_\lambda} q_{re}(s_{i,j_r}) \\
&\quad \times \prod_{s_{i,j_r}\notin MS_\lambda} (1-q_{re}(s_{i,j_r}))
\end{aligned} \tag{9.2}$$

where MS_λ is the λ^{th} non-empty subset of $MS_{i,j}$. Since all the services in $SRS_{i,j}^{pl}$ are performed simultaneously, the costs of all services in the scheme are summed, while the response time is calculated by summing the minimum response time of all MS_λ, which is weighted by the probability that only the services in MS_λ are executed successfully. In *Parallel* mode the throughput is determined by the first service in MS_λ that completes successfully. The operator *SMRT* returns the service in MS_λ with minimum response time.

In both redundancy modes, the reliability of a service redundancy scheme is the probability that at least one service completes successfully.

In order to capture the requirements of formulating cost-effective fault tolerance strategy for an SBS, we introduce *Fault Tolerance Budget* and *Fault Tolerance Cost* in this thesis. Fault tolerance budget indicates how much the SBS vendor is willing to pay for the deployment of fault tolerance, while fault tolerance cost represents how much the SBS vendor has to pay for that. The cost of the j^{th} service redundancy scheme in the i^{th} service class can be calculated with (9.3):

$$q_{ft}(SRS_{i,j}) = q_{ct}(SRS_{i,j}) - q_{ct}(s_i) \tag{9.3}$$

where $q_{ct}(s_i)$ is the cost of component service s_i in the i^{th} service class.

The fault tolerance cost of an SBS $\mathbb{S}$ can be calculated by (9.4):

$$q_{ft}(\mathbb{S}) = \sum_{\substack{SRS_{i,j} \in epl_e \\ epl_e \in \mathbb{S}}} p(epl_e) \times q_{ft}(SRS_{i,j}) \tag{9.4}$$

where $p(epl_e)$ is the execution probability of the e^{th} execution plan of $\mathbb{S}$.

9.1.3 Fault tolerance strategy determination

Fault tolerance strategy determination is the process of allocating service redundancies for some or all of the component services in an SBS. To determine the fault tolerance strategy for the SBS, we formulate a Constraint Optimisation Problem (COP). Considering an SBS $\mathbb{S}$ with n component services $\{s_1, s_2, \ldots, s_n\}$, and each of them is associated with a service class sc_i containing m_i ($m_i \geq 1$) candidate services $\{s_{i,1}, s_{i,2}, \ldots, s_{i,m_i}\}$. The system engineer needs to determine the fault tolerance strategy based on the criticalities of component services under the fault tolerance budget. The objective of optimisation is to maximise the sum of criticalities of component services to be protected, which means the impacts on the system quality upon service anomalies can be minimised under a limited budget.

Fault tolerance strategy determination is a complex decision making process, which is NP-complete and often very time-consuming given a large number of service redundancy schemes. In order to find a solution rapidly, FTC4MTS first ranks the candidate services within each service class by their utilities, which are calculated with the approach introduced in Section 4.4, and then use a greedy method that increases the number of redundant services gradually in a multi-round manner. Given a service class containing m_i candidate services, in each round, the top k_r candidate services with the highest utilities in each service class are selected to generate service redundancy schemes, where k_r is incremental in each round. A solution to the optimisation model can be found more rapidly with a larger k_r. In FTC4MTS we use $k_r = 2v$ $(1 \leq v \leq \lceil m_i/2 \rceil)$ and v is the round in which the optimisation is performed. k_r can be bounded by the system engineer by specifying a maximum round number v^{max}, which can reduce the complexity of fault tolerance strategy determination.

In each round of fault tolerance strategy determination, we create a set of 0-1 integer variables $x_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,l}\}$ for service class sc_i, where l is the number of service redundancy schemes in the i^{th} service class. $x_{i,j}$ with value of 1 means that the j^{th} service redundancy scheme in the i^{th} service class is selected for the formulation of fault tolerance strategy for SBS $\mathbb{S}$. We also create a 0-1 integer variable y_i for component service s_i, which is 1 if service redundancy is to be allocated for the component service s_i, and 0 otherwise. The process of finding a solution to the COP model is to assign a value of 0 or 1 to each variable and ensure that all constraints can be satisfied, including the quality constraints. This is equivalent to a knapsack problem, thus is an NP-complete problem, which makes it computationally expensive, if not impractical, to find an optimal solution in large-scale scenarios.

The COP model is formulated by (9.5) to (9.9):

$$Objective(\mathbb{S}): \{Maximising(\sum_{i=1}^{n} cr^{O}(s_i) \times y_i)\} \tag{9.5}$$

$$\begin{aligned} q_p^{pos}(\mathbb{S}) &\geq C_p^{pos} \\ q_p^{neg}(\mathbb{S}) &\leq C_p^{neg} \end{aligned} \tag{9.6}$$

$$q_{ft}(\mathbb{S}) \leq C_b \tag{9.7}$$

$$y_{i+1} \leq y_i, \; i = 1, \ldots, n-1 \tag{9.8}$$

$$\sum_{j=1}^{l} x_{i,j} = y_i, \; x_{i,j}, y_i \in \{0, 1\} \tag{9.9}$$

where constraints family (9.6) ensures that the multi-dimensional quality constraints for the SBS can be satisfied, where q_p^{pos} and q_p^{neg} are the p^{th} end-to-end quality of $\mathbb{S}$, which may be positive or negative quality dimension. C_p^{pos} and C_p^{neg}

are the p^{th} quality constraints over $\mathbb{S}$. Constraints family (9.7) ensures that fault tolerance cost cannot exceed budget C_b. Constraints family (9.8) ensures that only most critical component services can have redundant services. Constraints family (9.9) guarantees that at most one service redundancy scheme can be selected if the i^{th} component service is to be protected. Formula (9.5) shows the optimisation objectives for the COP model: maximising the sum of overall criticalities of component services that will be allocated service redundancy.

After the COP model is created, the Integer Programming (IP) technique is employed to find the optimal solution. Once the fault tolerance strategy is determined, the redundant services in the selected redundancy schemes can be deployed in the SBS.

The computational overhead introduced by FTC4MTS includes two main parts: the computation time of generating service redundancy schemes from candidate services, and the computation time of determining the fault tolerance strategy based on the COP model. As discussed in Section 9.1.2, the increase in the number of redundant services can result in a rapid increase in the number of service redundancy schemes, which may make it more computationally expensive to find the solution based on the COP model. However, the fault tolerance strategy is formulated at design time or in an offline manner at runtime, and thus does not cause extra computational overhead at runtime. In very large scenarios, the redundancy schemes can be enumerated in parallel by different machines with distributed computing techniques, such as MapReduce. On the other hand, service recommendation or filtering approaches, e.g., clustering-based service recommendation[19], and skyline techniques[49], can be used on candidate services as well as service redundancy schemes to further improve the efficiency of FTC4MTS. In overall terms, the computational overhead caused by FTC4MTS is much less significant than that introduced by the conventional runtime adaptation approaches, especially in extremely volatile environments where frequent runtime adaptation is needed.

9.2 Experimental evaluation

We have conducted a range of experiments in a simulated volatile environment, aiming at evaluating FTC4MTS in terms of effectiveness and efficiency in fault tolerance by comparing it with other representative approaches.

Section 9.2.1 presents evaluation metrics. Section 9.2.2 describes the setup of the experiments. Sections 9.2.3 and 9.2.4 evaluate the effectiveness and efficiency of FTC4MTS respectively.

9.2.1 Evaluation metrics

The effectiveness of fault tolerance is measured by three quantitative metrics as follows:

1. Success rate of fault tolerance: The percentage of successful fault tolerance in all test instances given the same fault tolerance budget. If anomalies occur and the component service has service redundancy in position, the fault tolerance is regarded as successful unless the component service and all redundant services fail at the same time.

2. Quality degradation: Including that in response time and throughput, quality degradation is represented by the average variation percentage of the end-to-end quality value of an SBS upon runtime anomalies.

3. Affected tenant percentage: The proportion of tenants affected by anomalies to all tenants that share the SBS.

In addition, the impacts of the environment volatility is also analysed. Efficiency of all approaches is evaluated based on the computational overhead, which is the average computation time needed for formulating a fault tolerance strategy for an SBS.

9.2.2 Experimental setup

Same as before, FTC4MTS is implemented in Java with JDK 1.6.0 and Eclipse Java EE IDE. IBM CPLEX v12.6 is again employed to solve the COPs. We have also implemented three representative and state-of-the-art approaches for comparison in the context of this research. These approaches, namely Quality-Optimal, FTCloud and Random, are described as follows:

- **Quality-Optimal**: Originated from the work presented in [60], this approach formulates fault tolerance strategies without taking service criticality into account. All component services are considered equally important. COP models are created only based on the quality performance, and the optimisation goal is to maximise the system utility. For the sake of fair comparison, Quality-Optimal in our experiments uses the same quality dimensions and constraints as FTCloud, Random and FTC4MTS.

- **FTCloud**: This approach is originated from [26], which evaluates the criticalities of component services of an SBS based on the invocation structure and invocation frequencies of the component services. A component service invoked more frequently by other services is considered more important. Service redundancy schemes are generated accordingly. Based on the service redundancy schemes, a COP model is created to formulate the fault tolerance strategies aiming at reducing the system failure probability. Local rather than global quality constraints, i.e., the constraints on individual component service instead of the service composition of SBS, are adopted in the COP. However, for this approach in the experiments, FTCloud employs the global quality constraints, the same as other approaches, for a fair and objective comparison.

- **Random**: A service redundancy scheme is created for a given component service by choosing redundancy mode and redundant services randomly.

The component services are also selected in a random manner and each of them can be selected once only. Same quality dimensions as FTC4MTS are adopted and the quality constraints on the SBS must not be violated in this process. This approach is also used in [26] as an existing approach to compare with.

In order to simulate a volatile cloud environment, we adopt the similar approach as described in [26]. Each component service has a failure rate. The more service requests it receives, the more failures may occur. In our series of experiments, same as works in [3][26][76], we set the failure rate at a fixed value, such as 1%, and then inject service anomalies to the SBS at runtime. In order to create different levels of volatility in the experimental environment, we vary the number of anomalies injected at one time and the quality degradation coefficient (see Definition 7.1) of service, and their impacts are analysed. For the purpose of simplicity and consistency without losing generality, when anomalies occur to a component service without service redundancy, we assume that the quality degrades according to a unified quality degradation coefficient, which is calculated by $q_p^{dc} = (2)^h$ and $q_p^{dc} = 1-(2)^{-h}, h \geq 1$ for negative and positive quality dimensions respectively. For example, when h is 1, the response time and throughput degrade to 200% and 50% of their original values respectively. This is the same experimental setup as that in Section 7.2.2.

In all the experiments conducted, we use fault tolerance budgets as a critical constraint in the formulation of fault tolerance strategies. In this way, we evaluate the effectiveness of FTC4MTS in formulating cost-effective fault tolerance strategy by comparing with the Quality-Optimal, FTCloud and Random.

The services used in the experiments are also generated based on QWS [88]. For each service, the cost and throughput are generated randomly based on normal distribution.

Though we conducted experiments on the motivating example *VOVS* introduced in Section 2.2.1 as a case study, in order to compare FTC4MTS with existing

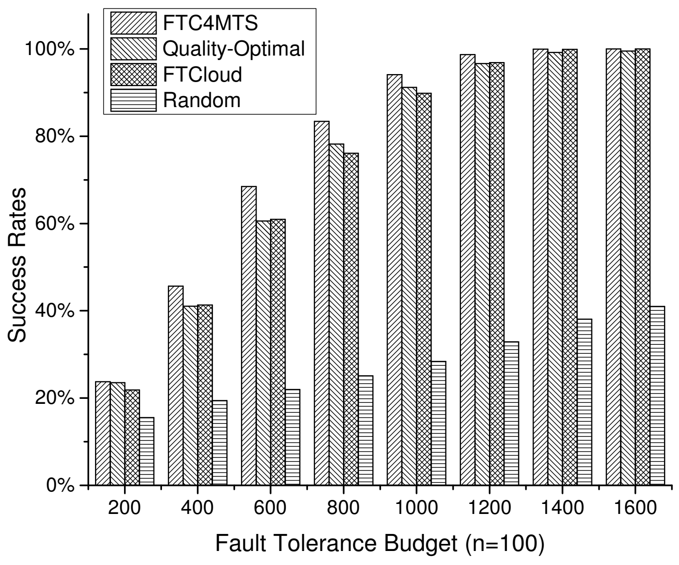


Figure 9.3: Comparison in success rates.

approaches more comprehensively, we have mimicked SBSs with different numbers of tasks n (up to 100). The findings are consistent with those from the experiments on $VOVS$. For each test instance, a business process consisting of n tasks is constructed first with randomly generated composition patterns introduced in Section 4.1. After that, 100 candidate services are created for each service class, based on which a service composition is formulated with IP techniques.

For each test instance with n service classes, $10 \times n$ tenants are created and distributed randomly when conditional branches exist in the service composition. Tenants' quality preferences are also generated randomly.

Same as before, all the experiments were conducted on a machine running Windows 7x64 Enterprise with Intel(R) Core (TM) i5-4570 3.2 GHz CPU and 8 GB RAM. The experimental results are collected, averaged and compared from 100 test instances.

9.2.3 Effectiveness evaluation

9.2.3.1 Comparison in success rates

In order to examine the effectiveness of FTC4MTS compared to Quality-Optimal, FTCloud and Random, we compare the success rates of fault tolerance obtained by these approaches. For a fixed number of service classes, we fix v^{max} at 2 and increase the fault tolerance budget in steps of 200. One anomaly is injected to the SBS in each test instance. The experimental results show that, when budget increases, the success rates obtained by all approaches increase, and FTC4MTS outperforms other approaches in all scenarios. We use test instances with $n = 100$ as samples to demonstrate the results. As shown in Fig. 9.3, FTCloud and Quality-Optimal achieve similar performance and are beaten by FTC4MTS by approximately 5% on average. Random obtains the lowest success rates among all the approaches as expected.

9.2.3.2 Comparison in quality degradation

In order to evaluate the effectiveness of FTC4MTS in alleviating system quality degradation upon runtime anomalies against existing approaches, we conduct a set of experiments, where different approaches in comparison are employed, to observe the quality degradations in response time and throughput caused by runtime anomalies. In this set of experiments, the quality of a faulty service degrades to the maximum degradation level, which is represented by the quality degradation coefficient (see Definition 7.1). For a fixed number of service class n, the fault tolerance budget is increased in steps of 200. One anomaly is injected to the SBS in each test instance. The experimental results show that quality degradation of both quality dimensions can be alleviated with all approaches when the budget increases, while FTC4MTS outperforms other three approaches significantly. Again, we use test instances with $n = 100$ as examples to demonstrate the results. As shown in Fig. 9.4 (a), in terms of response time degradation, Quality-Optimal

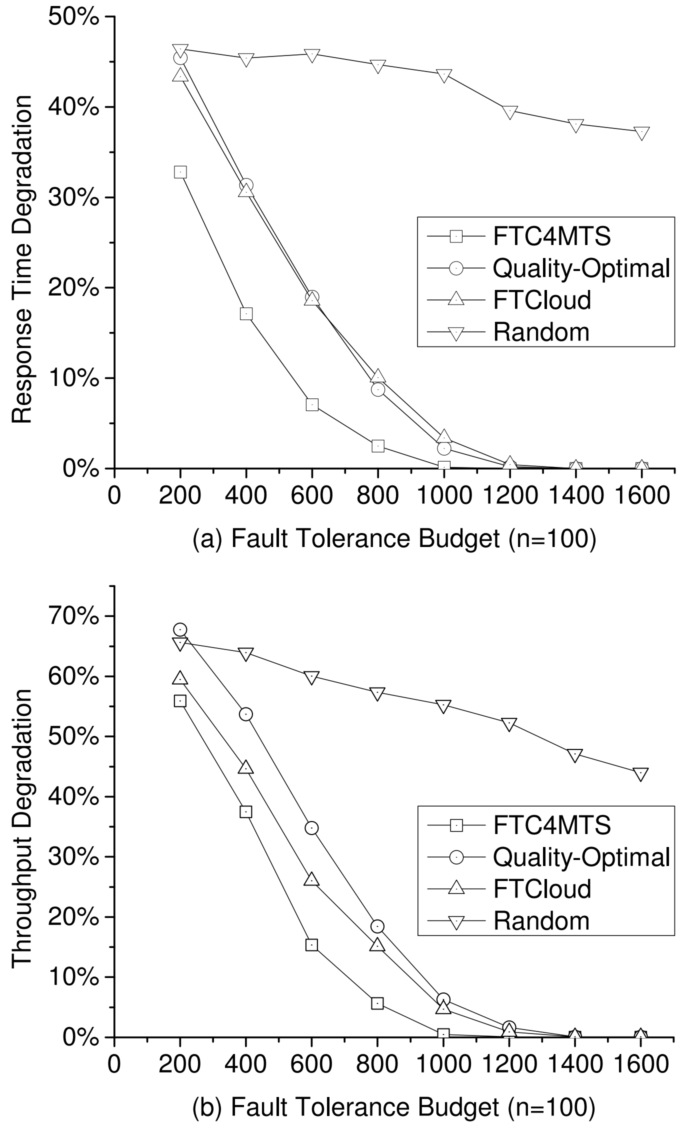


Figure 9.4: Comparison in SBS-level quality degradation. (a) Degradation of response time; (b) Degradation of throughput.

gains a similar performance with FTCloud and outperforms it slightly when the budget exceeds about 600. FTC4MTS outperforms Quality-Optimal and FTCloud by 10% on average. Random, as expected, is beaten by other three counterparts significantly. From the perspective of throughput degradation, a similar result can be seen in Fig. 9.4 (b), where FTC4MTS shows the best performance and outperforms FTCloud and Quality-Optimal by 4.5% and 8.3% on average respectively.

The reason for the noticeable advantage of FTC4MTS over other approaches under a constrained fault tolerance budget is that with FTC4MTS, the component service that may cause more severe quality degradation has a higher criticality, and thus is given higher priority in the allocation of service redundancy. As a result, the severity of system quality degradation caused by runtime anomalies can be alleviated. When the budget is high enough, all approaches except Random can achieve similar and satisfactory results.

9.2.3.3 Comparison in affected tenant percentage

When an anomaly occurs to a component service of a multi-tenant SBS without a proper fault tolerance strategy, all the tenants sharing that component service may experience unexpected quality degradation. When quality degradation is unavoidable, the common practice is to limit the degradation to as few tenants as possible. With the same setup of the experiments in Section 9.2.3.2, we conduct experiments, where different approaches in comparison are employed, to observe the affected tenant percentage. The results are presented in Fig. 9.5. As demonstrated, FTC4MTS prevents significantly more tenants from being affected by runtime anomalies than the other approaches. When fault tolerance budget varies from 200 to 1600, the affected tenant percentages obtained by all approaches decrease gradually and drop more quickly with FTC4MTS and FT-Cloud. Due to the consideration of service sharing across the tenants when calculating service criticality, FTC4MTS gains the most obvious advantage (over 20% on average) when budget is between 200 and 1000.

9.2.3.4 Impact of environment volatility

In this series of experiments, we evaluate the impacts of environment volatility on the quality of the SBS from two perspectives: the impacts of the number of anomalies occurred to the SBS concurrently and the quality degradation severity of a failed component service. The latter depends on the quality degradation coefficient (see Definition 7.1) in this thesis. The experimental results show that when the operating environment is extremely volatile (e.g., large-scale service unavailability in the event of natural disaster), severe quality degradation is inevitable. In such an operating environment, FTC4MTS outperforms other approaches by relatively larger margins when the operating environment becomes more volatile.

A sample is shown in Fig. 9.6 (a), where we vary the number of anomalies in each test instance from 10 to n ($n = 100$), fix fault tolerance budget at 500 and the quality degradation coefficient at 2, and observe the degradation (increase) in system response time. As depicted in Fig. 9.6 (a), when the number of concurrent anomalies rises, the response time degradation shown by other three approaches increases more rapidly than that presented by FTC4MTS, which shows a significant advantage over Quality-Optimal with a response time degradation margin of approximately 50% on average. With FTCloud and Random, which are much more vulnerable to the anomalies among the four approaches in this experiment, the response time degrades dramatically (roughly from 47% to 270% on average) along with the growth of the number of anomalies that occurred to the SBS concurrently.

We conduct another set of experiments with regards to the impact of quality degradation severity, which is determined by the quality degradation coefficient.

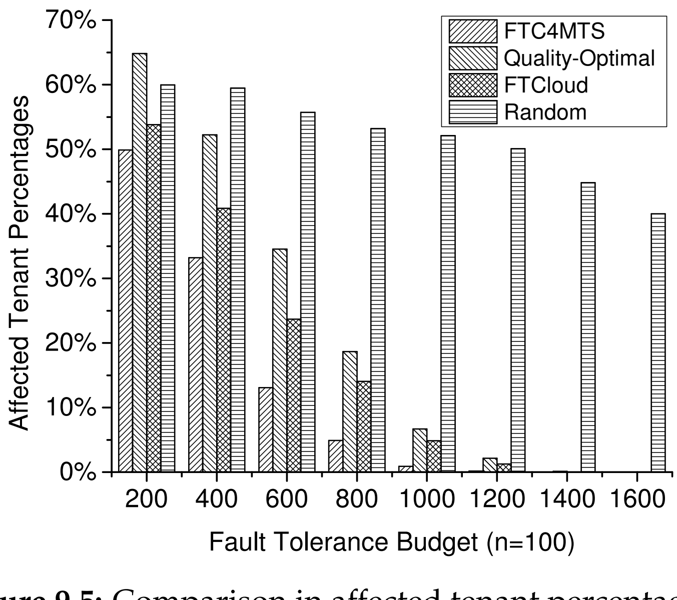


Figure 9.5: Comparison in affected tenant percentages.

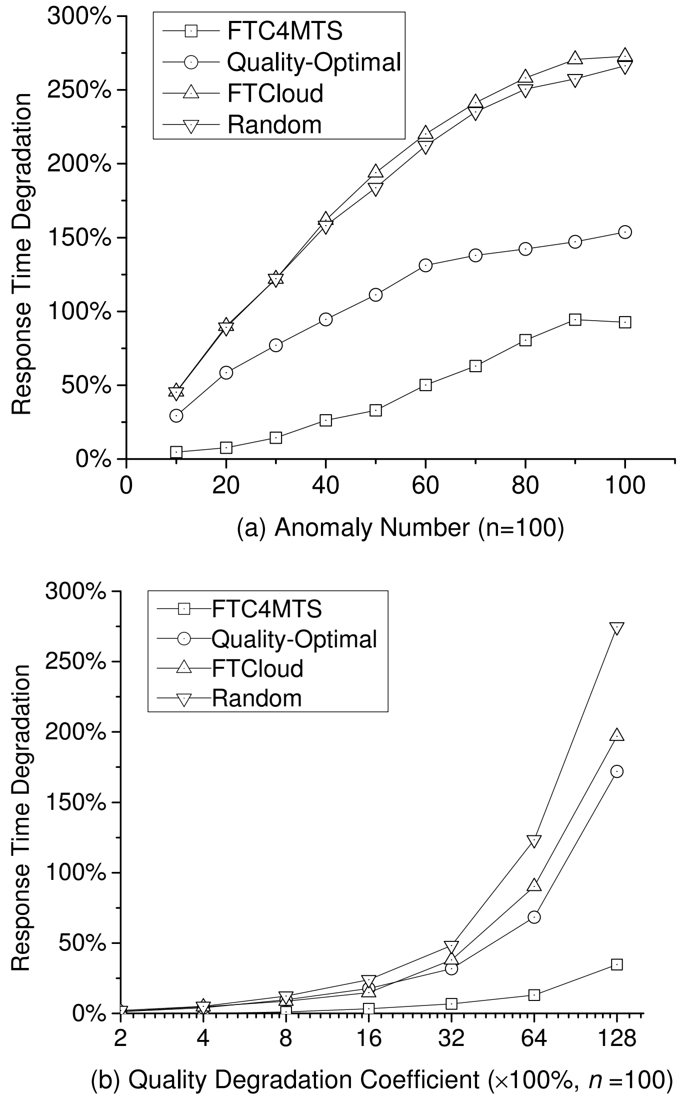


Figure 9.6: Impact of environment volatility. (a) Impact of the number of anomalies that occurred to the SBS concurrently; (b) Impact of quality degradation coefficient.

As described in Section 9.2.2, the quality degradation coefficients for response time and throughput are defined by $(2)^h$ and $1-(2)^{-h}$ respectively. As an example, we use 100 service classes, fix fault tolerance budget at 500, and vary quality degradation coefficient of response time by increasing h from 1 to 7 in steps of 1. One anomaly is injected to the SBS in each test instance. Fig. 9.6 (b) compares the increases in system response time when the four approaches are adopted. When the quality degradation coefficient varies from 200% to 12800%, the increases in response time obtained by the four approaches are approximately

34.6% (FTC4MTS), 172.1% (Quality-Optimal), 196.8% (FTCloud), and 274.7% (Random) on average. Fig. 9.6 (b) illustrates that quality degradations occurred and worsened regardless of the adopted approach. However, FTC4MTS has an obvious advantage over the other three.

9.2.3.5 Analysis of statistical significance

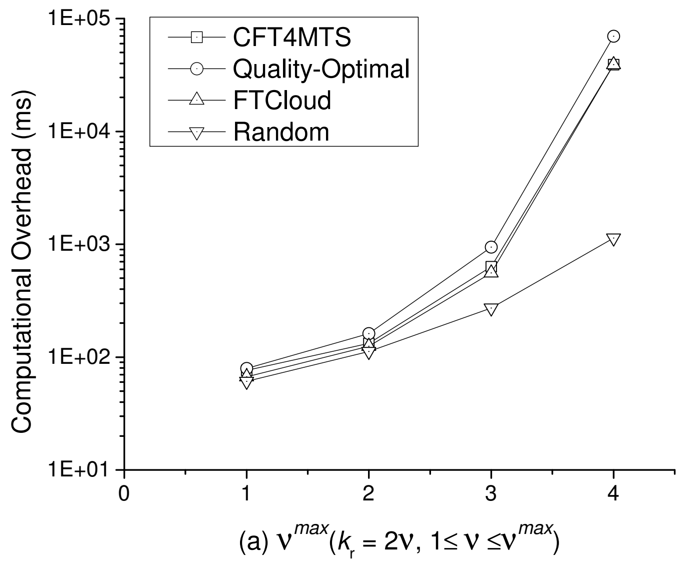
In order to assess the statistical significance of the advantages held by FTC4MTS over other approaches, we conduct *t*-test based on the results collected in the experiments in Sections 9.2.3.1 to 9.2.3.3 respectively. The *t*-test results are shown in Table 9.1, where all the null hypothesises that FTC4MTS has no difference to the second best approach in each set of experiments are rejected. Thus we can conclude that FTC4MTS produces significant advantages in effectiveness over other three approaches.

Table 9.1: *t*-test Results with Critical Value of 2.365 (95% Confidence Interval)

Sample Data	*t*-values	*p*	Reject/Accept Null Hypothesis
Success Rates	3.254	0.0140	Reject
Response Time Degradation	2.991	0.0202	Reject
Throughput Degradation	3.014	0.0195	Reject
Affected Tenant Percentage	3.085	0.0177	Reject

9.2.4 Efficiency evaluation

In order to evaluate the efficiency of FTC4MTS, we conduct a set of experiments to assess its computational overhead. The impacts of two parameters are evaluated: the maximum number of redundant services for a component service and the number of service class. These two parameters have important influences on the complexity of the fault tolerance strategy formulation, which, as described in Section 9.1.3, is an NP-complete problem. The results show that with the increase in complexity, the computational overhead introduced by all approaches



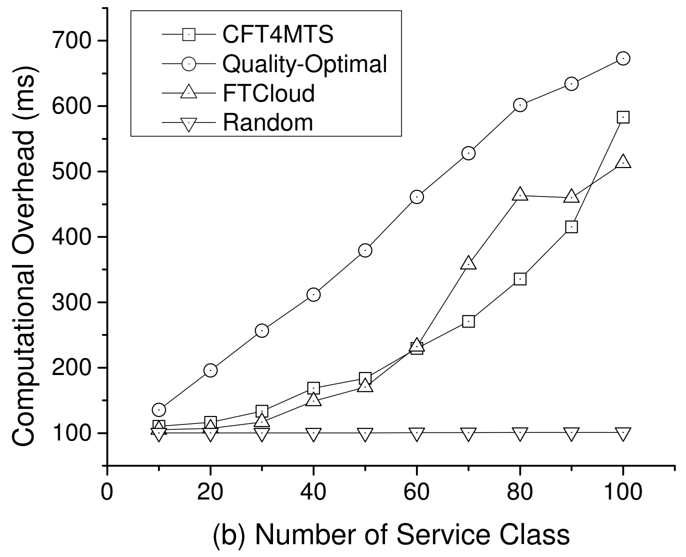


Figure 9.7: Comparison in computational overhead. (a) Impact of ν^{max}; (b) Impact of the number of service class.

increases. However, FTC4MTS shows a better or similar performance compared with other approaches except Random, which is a naive and non-optimal approach.

We first evaluate the impacts of maximum redundant service number on the computational overhead. In this set of experiments, we fix n at 10, fault tolerance budget at 500 and change ν^{max} from 1 to 4 to vary the value of k_r ($k_r = 2\nu, 1 \le \nu \le \nu^{max}$), and then collect the average computation time achieved by the four approaches for finding solutions successfully. In order to examine the performance of these approaches in the worst-case scenarios and compare the efficiency on an equal basis, we let $v = v^{max}$ in each round. Fig. 9.7 (a) shows that the computation times presented by all approaches grow rapidly along with the increase of v^{max}. This is because the scale of service redundancy scheme grows with the increase of v^{max} (here $v = v^{max}$), which consequently increases the complexity of the optimisation problem and makes it increasingly difficult to find an optimal solution. However, a large v^{max} is not necessary in practice considering the overhead it introduces and the improvement it brings to the system quality, which is analysed in Section 9.3.

Fig. 9.7 (b) shows the impacts of service class number n on the computational overhead. We fix v^{max} at 2 and fault tolerance budget at 500, and vary n from 10 to 100. The results show that the computation times consumed by all four approaches increase when n rises. The computational overheads introduced by FTC4MTS, Quality-Optimal and FTCloud increase much more significantly (from about 130ms to approximately 600ms on average) than that of Random, which stays stably at around 100ms.

9.3 Discussion

In this chapter, we use response time and throughput as two example quality dimensions. Other quality dimensions can be used in the similar manner. Similar to the service monitoring discussed in Section 7.3, the non-linear quality constraints and dimensions are not considered in the COP model either.

k_r is a factor that may impact the experimental comprehensiveness in this chapter. It represents the maximum number of redundant services determined by v^{max} in a service redundancy. We use a relatively small k_r in the experimental setup, because on one hand a large k_r would inevitably cause heavy computational overhead. On the other hand, deploying more services for fault tolerance introduces higher cost but may not yield significant improvement in system quality. For example, from the reliability point of view, the probability of that the component service and all redundant services fail at the same time is very small. We assume there are three redundant services deployed for a component service in a service redundancy and each service has a reliability of 90%, then theoretically the overall reliability of formulated service redundancy can reach 99.9999%, which is very high. Therefore, we believe that k_r is not a major limitation in practice.

9.4 Summary

Quality-aware cost-effective fault tolerance for multi-tenant SBS is a critical issue in the dynamic and volatile cloud environments. This chapter introduces a novel approach named FTC4MTS that formulates fault tolerance strategies for SBSs based on the service criticality. Component services are ranked by their criticalities and critical component services are given the priorities in the allocation of service redundancy. By doing so, the fault tolerance capabilities of SBSs can be improved significantly. To evaluate the effectiveness and efficiency of FTC4MTS, we have conducted a series of experiments, where we compared FTC4MTS with three representative approaches and analysed the impacts of various factors. The results show that FTC4MTS can formulate fault tolerance strategies for SBSs under a limited budget, which effectively and efficiently reduce the risk of system quality degradation for multi-tenant SBSs.

Chapter 10

Conclusions and future work

In this chapter, we summarise this thesis and present the contribution of this research. Some future research directions of lifetime quality management for multi-tenant SBSs are discussed.

10.1 Summary of this thesis

The aim of this research is to investigate a novel and systematic solution named LQM4MTS to address the issues in lifetime quality management for multi-tenant SBSs. LQM4MTS includes a set of approaches that focus on the quality management for different lifetime stages of multi-tenant SBSs, namely service selection, service monitoring and service adaptation, which are organised in Part I to Part III respectively in this thesis. The chapters in this thesis are summarised as follows:

- Chapter 1 introduces the concept of multi-tenant SBSs including their lifetime stages of service selection, service monitoring and service adaptation. The key issues to be addressed in this research and the structure of this thesis are also introduced.

- Chapter 2 first reviews the related work in each lifetime stage. Then, based on a motivating example, the research requirements are presented, which argue that a systematic, effective and efficient solution is needed for quality management across all the lifetime stages of multi-tenant SBSs.
- Chapter 3 presents the framework of LQM4MTS (Lifetime Quality Management for Multi-Tenant SBSs), which consists of a set of techniques to address the key issues in the entire lifetime of a multi-tenant SBS.
- Chapter 4 introduces the composition model used in this research, including service composition pattern, execution path and execution plan, quality characteristics, and utility evaluation. The composition model presented is the basis of the techniques proposed in this thesis.
- Chapter 5, as the first chapter of Part I, introduces SSR4MTS (Service Selection based on service Recommendation for Multi-Tenant SBSs), which supports efficient service selection using service recommendation based on clustering techniques. Tenants are firstly clustered according to the characteristics of their quality requirements, based on which, the candidate services are also clustered and the services with similar quality features are recommended for service selection. Experimental results show that SSR4MTS can significantly improve the efficiency of service selection for multi-tenant SBSs with high effectiveness.
- Chapter 6, as the second chapter of Part I, presents SSC4MTS (Service selection based on Correlated quality requirements for Multi-Tenant SBSs), which supports service selection that fulfils tenants' multi-dimensional quality requirements with correlation between different quality dimensions. The quality correlations are firstly formalised with correlation functions, and then the service selection are modelled as Constraint Optimisation Problems. Quality Satisfaction Degree is proposed to represent the goal of system optimisation. According to the experimental results, SSC4MTS can sup-

port the service selection based on correlated quality requirements in an effective and efficient way.

- Chapter 7, as Part II, introduces SMC4MTS (Service Monitoring based on Criticality for Multi-Tenant SBSs), which is designed for formulating cost-effective monitoring strategies for multi-tenant SBSs based on service criticality. Quality-based criticality and tenant-based criticality are calculated and integrated as the overall service criticality, based on which, the component service in an SBS are ranked and those with higher criticalities are prioritised in the formulation of monitoring strategies. The monitoring benefits, resource cost for monitoring and incurred system overhead are comprehensively considered in the proposed model. The evaluation results show that SMC4MTS can significantly reduce the quality degradation of an SBS upon runtime anomalies with high efficiency and effectiveness especially cost-effectiveness.

- Chapter 8, as the first chapter of Part III, presents SAL4MTS (Service Adaptation based on LSH for Multi-Tenant SBSs), which supports rapid runtime system adaptation for multi-tenant SBSs by selecting the alternative services with equivalent functionalities to replace the anomalous component services. Locality-Sensitive Hashing is used to find the candidate services with most similar quality performance with the faulty services. The experimental results demonstrate that SAL4MTS is able to find proper replacement services efficiently with high effectiveness.

- Chapter 9, as the second chapter of Part III, introduces FTC4MTS (Fault Tolerance based on Criticality for Multi-Tenant SBSs), which supports formulating cost-effective fault tolerance strategies for multi-tenant SBSs based on service criticality. In our research, this is the other scenario that service criticality is used to find the critical component services in a multi-tenant SBS. Service redundancy is allocated to the component services with high criticalities in the formulation of fault tolerance strategies. The experimental evaluation shows that FTC4MTS can alleviate the system quality degradation for a multi-tenant SBS within the limited budget in an effective and efficient way.

10.2 Contributions of this thesis

The significance of this research is that it addresses the problem of quality management for multi-tenant SBSs at different lifetime stages, including service selection, service monitoring and service adaptation. We propose a novel and systematic solution which consists of a set of approaches that supports the quality management activities across the entire lifetime of multi-tenant SBSs. The effectiveness and efficiency of the proposed approaches have been evaluated by extensive experiments. The major contributions of the work presented in this thesis are as follows:

- The challenges and issues in lifetime quality management for multi-tenant SBSs are identified and a systematic solution named LQM4MTS is proposed to address the issues. LQM4MTS consists of a set of innovative approaches support effective and efficient quality management at different lifetime stages of multi-tenant SBSs.

- Two innovative approaches are proposed for effective and efficient service selection: SSR4MTS and SSC4MTS. SSR4MTS is suitable for the scenario where tenants and services are clusterable with distinctive features in their quality requirements and quality values respectively. By innovatively exploring the similarity between tenants' quality requirements and candidate services' quality values, SSR4MTS uses service recommendation based on clustering techniques to achieve effective and efficient service selection for multi-tenant SBSs. SSC4MTS is used to handle tenants' correlated quality requirements for an SBS in service selection. It considers the correlations between different quality dimensions of tenants' quality requirements. The methods for formalising the quality correlations and calculating quality satisfaction degree are proposed to facilitate the optimal service selection. SSC4MTS effectively and efficiently addresses a challenging issue in service selection existed in a real-world scenario but is insufficiently considered.

- A novel approach named SMC4MTS is proposed to support formulation of cost-effective service monitoring strategies. Service criticality is used to identify the critical component services in a multi-tenant SBS to achieve cost-effective service monitoring within limited monitoring budget. Service criticality is calculated based on three metrics: tenants' multi-dimensional quality, tenants' service sharing in the SBS, and their quality preferences for the SBS. The typical and important monitoring parameters are considered in the formulation of monitoring strategies, including the number of monitors, monitoring frequency and monitoring granularity. We propose monitoring utility to model the trade-off between monitoring benefit, monitoring resource cost and system overhead caused by different monitoring parameters. The optimal solution to cost-effective service monitoring can be found by solving a constraint optimisation problem, which is proposed to model the formulation of monitoring strategies.

- Two new approaches for service adaptation are proposed to provide quality guarantee for multi-tenant SBS: SAL4MTS and FTC4MTS. By using Locality-Sensitive Hashing (LSH) to find the nearest neighbours of the faulty services in terms of quality performance, SAL4MTS supports effective and efficient service adaptation with service replacement upon runtime anomalies. When tenants' quality requirements cannot be satisfied after service replacements, SSR4MTS is adopted to re-optimise the system to ensure the system quality. As another effective way for service adaptation, fault tolerance strategies are formulated by FTC4MTS based on service redundancy.

Service criticality is used again to achieve cost-effectiveness and high efficiency in fault tolerance within limited budget. The component services with high criticalities are prioritised in fault tolerance and run with the redundant services in certain redundancy mode. In this way, the quality degradation caused by runtime anomalies can be alleviated effectively and efficiently.

10.3 Future work

In the future, further investigation into lifetime quality management for multi-tenant SBSs can be carried out in several directions:

- For service selection, we plan to exploit the alternative clustering algorithms to overcome the limitations of K-Means, which is used in SSR4MTS for clustering tenants and services, e.g., the issues of pre-specified number of clusters, and sensitivity to outliers. More runtime scenarios with respect to service recommendation and selection will be investigated, such as the change of tenants' quality requirements, joining and leaving of new tenant, etc. As extension of SSC4MTS, more types of quality correlations will be investigated, and more comparison with other approaches such as those based on fuzzy logic will also be carried out.
- For service monitoring, the model of SMC4MTS will be enhanced by considering more monitoring parameters and exploring more flexible monitoring patterns.
- For service adaptation, more complicate and comprehensive adaptation strategy will be explored, e.g., adaptation based on quality prediction, using diverse adaptation actions, etc. The runtime adaptation of the formulated fault tolerance strategy will be investigated.
- At present, we focus on the linear quality constraints and dimensions or

those that can be linearised. In the future, we plan to extend the proposed approaches by considering the handling of non-linear quality constraints and dimensions, with which the constraint optimisation problem modelled in most of the approaches becomes a non-linear programming problem.

- In this thesis, the service sharing among tenants are deterministic. As a future work, the tenants' correlation in service sharing will be investigated, based on which, the sharing of services among different tenants becomes probabilistic.
- The quality handled in our approaches are also deterministic, and the probabilistic quality will be investigated as one of the future works.

Bibliography

[1] L. Baresi and S. Guinea, "Self-Supervising BPEL Processes," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 247–263, 2011.

[2] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS Management and Optimization in Service-Based Systems," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 387–409, 2011.

[3] Q. He, J. Han, Y. Yang, H. Jin, J.-G. Schneider, and S. Versteeg, "Formulating Cost-Effective Monitoring Strategies for Service-Based Systems," *IEEE Transactions on Software Engineering*, vol. 40, no. 5, pp. 461–482, 2014.

[4] W3C. (2007) SOAP Version 1.2. [Online]. Available: http://www.w3.org/TR/soap12/

[5] ——. (2007) Web Services Description Language (WSDL) Version 2.0. [Online]. Available: http://www.w3.org/TR/wsdl20/

[6] Oasis. (2004) UDDI Version 3.0.2. [Online]. Available: http://www.uddi.org/pubs/uddi_v3.htm

[7] W3C. (2004) Web Services Glossary. [Online]. Available: http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice

[8] ProgrammableWeb. (2014) Growth in Web APIs from 2005 to 2013. [Online]. Available: http://www.programmableweb.com/api-research

[9] Q. He, J. Han, F. Chen, Y. Wang, R. Vasa, Y. Yang, and H. Jin, "QoS-Aware Service Selection for Customisable Multi-Tenant Service-Based Systems: Maturity and Approaches," in *Proceedings of the 8th IEEE International Conference on Cloud Computing (CLOUD2015)*, 2015, pp. 237–244.

[10] M. Sajid and Z. Raza, "Cloud computing: Issues & challenges," in *Proceedings of International Conference on Cloud, Big Data and Trust*, vol. 20, no. 13, 2013, pp. 13–15.

[11] Q. He, J. Yan, R. Kowalczyk, H. Jin, and Y. Yang, "Lifetime Service Level Agreement Management with Autonomous Agents for Services Provision," *Information Sciences*, vol. 179, no. 15, pp. 2591–2605, Jul. 2009.

[12] D. Ardagna, B. Panicucci, and M. Passacantando, "A Game Theoretic Formulation of the Service Provisioning Problem in Cloud Systems," in *Proceedings of the 20th International Conference on World Wide Web (WWW2011)*, 2011, pp. 177–186.

[13] Q. He, J. Yan, H. Jin, and Y. Yang, "Quality-aware Service Selection for Service-based Systems based on Iterative Multi-attribute Combinatorial Auction," *IEEE Transactions on Software Engineering*, vol. 40, no. 2, pp. 192–215, 2014.

[14] P. A. Bonatti and P. Festa, "On Optimal Service Selection," in *Proceedings of the 14th International Conference on World Wide Web (WWW2005)*, 2005, pp. 530–538.

[15] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[16] B. P. Rimal, E. Choi, and I. Lumb, "A Taxonomy and Survey of Cloud Computing Systems," *INC, IMS and IDC*, pp. 44–51, 2009.

[17] N. Poggi, D. Carrera, R. Gavalda, and E. Ayguadé, "Non-intrusive Estimation of QoS Degradation Impact on E-Commerce User Satisfaction," in *Proceedings of the 10th IEEE International Symposium on Network Computing and Applications (NCA2011)*, 2011, pp. 179–186.

[18] J. Barr, A. Narin, and J. Varia. (2011) Building Fault-Tolerant Applications on AWS. [Online]. Available: http://media.amazonwebservices.com/AWS_Building_Fault_Tolerant_Applications.pdf

[19] Y. Wang, Q. He, and Y. Yang, "QoS-Aware Service Recommendation for Multi-Tenant SaaS on the Cloud," in *Proceedings of the 12th IEEE International Conference on Service Computing (SCC2015)*, 2015, pp. 178–185.

[20] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.

[21] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints," *ACM Transactions on the Web (TWEB)*, vol. 1, no. 1, article 6, 2007.

[22] Q. He, J. Han, Y. Yang, J. Grundy, and H. Jin, "QoS-driven Service Selection for Multi-tenant SaaS," in *Proceedings of the 5th IEEE International Conference on Cloud computing (Cloud2012)*, 2012, pp. 566–573.

[23] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "QoS-Aware Web Service Recommendation by Collaborative Filtering," *IEEE Transactions on Services Computing*, vol. 4, no. 2, pp. 140–152, 2011.

[24] X. Chen, Z. Zheng, Q. Yu, and M. R. Lyu, "Web Service Recommendation via Exploiting Location and QoS Iformation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1913–1924, 2014.

[25] M. C. Jaeger, G. Muhl, and S. Golze, "QoS-aware Composition of Web Services: A Look at Selection Algorithms," in *Proceedings of the 12th IEEE International Conference on Web Services (ICWS2005)*, 2005, pp. 807–808.

[26] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King, "Component Ranking for Fault-tolerant Cloud Applications," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 540–550, 2012.

[27] D. Ardagna and B. Pernici, "Adaptive Service Composition in Flexible Processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 369–384, 2007.

[28] V. Nallur and R. Bahsoon, "A Decentralized Self-adaptation Mechanism for Service-based Applications in the Cloud," *IEEE Transactions on Software Engineering*, vol. 39, no. 5, pp. 591–612, 2013.

[29] K. Birman, R. van Renesse, and W. Vogels, "Adding High Availability and Autonomic Behavior to Web Services," in *Proceedings of the 26th International Conference on Software Engineering (ICSE2004)*, 2004, pp. 17–26.

[30] Z. Zheng and M. R. Lyu, "A Distributed Replication Strategy Evaluation and Selection Framework for Fault Tolerant Web Services," in *Proceedings of the 6th IEEE International Conference on Web Services (ICWS2008)*, 2008, pp. 145–152.

[31] W. Zhao, P. Melliar-Smith, and L. E. Moser, "Fault Tolerance Middleware for Cloud Computing," in *Proceedings of 3rd IEEE International Conference on Cloud Computing (CLOUD2010)*, 2010, pp. 67–74.

[32] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "QoS-Aware Replanning of Composite Web Services," in *Proceedings of the 3rd IEEE International Conference on Web Services (ICWS2005)*, 2005, pp. 121–129.

[33] Z. Zheng, X. Wu, Y. Zhang, M. R. Lyu, and J. Wang, "QoS Ranking Prediction for Cloud Services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1213–1222, 2013.

[34] J. Han, J. Pei, and M. Kamber, *Data Mining: Concepts and Techniques*. Elsevier, 2011.

[35] M. Alrifai and T. Risse, "Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition," in *Proceedings of the 18th International Conference on World Wide Web (WWW2009)*, 2009, pp. 881–890.

[36] F. Zhang, K. Hwang, S. U. Khan, and Q. M. Malluhi, "Skyline Discovery and Composition of Multi-Cloud Mashup Services," *IEEE Transactions on Services Computing*, vol. 9, no. 1, pp. 72–83, 2016.

[37] F. Wagner, F. Ishikawa, and S. Honiden, "Robust Service Compositions with Functional and Location Diversity," *IEEE Transactions on Services Computing*, vol. 9, no. 2, pp. 277–290, 2016.

[38] Z. Xiang, S. Deng, and H. Gao, "Service Selection Using Service Clusters," in *Proceedings of the 12th IEEE International Conference on Services Computing (SCC2015)*, 2015, pp. 769–772.

[39] S.-Y. Hwang, C.-C. Hsu, and C.-H. Lee, "Service Selection for Web Services with Probabilistic QoS," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 467–480, 2015.

[40] S. Deng, H. Wu, D. Hu, and J. L. Zhao, "Service Selection for Composition with QoS Correlations," *IEEE Transactions on Services Computing*, vol. 9, no. 2, pp. 291–303, 2016.

[41] Y. Zhang, G. Cui, S. Deng, and Q. He, "Alliance-Aware Service Composition Based on Quotient Space," in *Proceedings of the 23rd IEEE International Conference on Web Services (ICWS2016)*, 2016, pp. 340–347.

[42] Y. Du, H. Hu, W. Song, J. Ding, and J. Lü, "Efficient Computing Composite Service Skyline with QoS Correlations," in *Proceedings of the 12th IEEE International Conference on Services Computing (SCC2015)*, 2015, pp. 41–48.

[43] Y. Chen, J. Huang, C. Lin, and X. Shen, "Multi-Objective Service Composition with QoS Dependencies," *IEEE Transactions on Cloud Computing*, 2016, http://dx.doi.org/10.1109/TCC.2016.2607750.

[44] X. Liang, A. Qin, K. Tang, and K. C. Tan, "QoS-aware Web Service Composition with Internal Complementarity," *IEEE Transactions on Services Computing*, 2016, http://dx.doi.org/10.1109/TSC.2016.2598776.

[45] Z. Ye, S. Mistry, A. Bouguettaya, and H. Dong, "Long-term QoS-aware Cloud Service Composition using Multivariate Time Series Analysis," *IEEE Transactions on Services Computing*, vol. 9, no. 3, pp. 382–393, 2016.

[46] M. Lin, J. Xie, H. Guo, and H. Wang, "Solving QoS-Driven Web Service Dynamic Composition as Fuzzy Constraint Satisfaction," in *Proceedings of IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE2005)*, 2005, pp. 9–14.

[47] M. Almulla, K. Almatori, and H. Yahyaoui, "A QoS-based Fuzzy Model for Ranking Real World Web Services," in *Proceedings of the 9th IEEE International Conference on Web Services (ICWS2011)*, 2011, pp. 203–210.

[48] K. K. Fletcher, X. F. Liu, and M. Tang, "Elastic Personalized Nonfunctional Attribute Preference and Trade-off based Service Selection," *ACM Transactions on the Web (TWEB)*, vol. 9, no. 1, pp. 1–26, 2015.

[49] M. Alrifai, D. Skoutas, and T. Risse, "Selecting Skyline Services for QoS-based Web Service Composition," in *Proceedings of the 19th International Conference on World Wide Web (WWW2010)*, 2010, pp. 11–20.

[50] C. Platzer, F. Rosenberg, and S. Dustdar, "Web Service Clustering using Multidimensional Angles as Proximity Measures," *ACM Transactions on Internet Technology (TOIT)*, vol. 9, no. 3, pp. 1–26, 2009.

[51] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis, "Ranking and Clustering Web Services using Multicriteria Dominance Relationships," *IEEE Transactions on Services Computing*, vol. 3, no. 3, pp. 163–177, 2010.

[52] Z. Zheng, Y. Zhang, and M. R. Lyu, "CloudRank: A QoS-driven Component Ranking Framework for Cloud Computing," in *Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems*, 2010, pp. 184–193.

[53] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King, "FTCloud: A component ranking framework for fault-tolerant cloud applications," in *Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering (ISSRE2010)*, 2010, pp. 398–407.

[54] G. Kang, J. Liu, M. Tang, X. Liu, B. Cao, and Y. Xu, "Awsr: Active web service recommendation based on usage history," in *Proceedings of the 19th IEEE International Conference on Web Services (ICWS2012)*, 2012, pp. 186–193.

[55] L. Liu, F. Lecue, and N. Mehandjiev, "Semantic Content-based Recommendation of Software Services using Context," *ACM Transactions on the Web (TWEB)*, vol. 7, no. 3, pp. 17–20, 2013.

[56] G. Kang, M. Tang, J. Liu, X. F. Liu, and B. Cao, "Diversifying Web Service Recommendation Results via Exploring Service Usage History," *IEEE Transactions on Services Computing*, vol. 9, no. 4, pp. 566–579, 2016.

[57] Z. Jiang, A. Zhou, S. Wang, Q. Sun, R. Lin, and F. Yang, "Personalized Service Recommendation for Collaborative Tagging Systems with Social Relations and Temporal Influences," in *Proceedings of the 13th IEEE International Conference on Service Computing (SCC2016)*, 2016, pp. 786–789.

[58] J. Liu, M. Tang, Z. Zheng, X. F. Liu, and S. Lyu, "Location-Aware and Personalized Collaborative Filtering for Web Service Recommendation," *IEEE Transactions on Services Computing*, vol. 9, no. 5, pp. 686–699, 2016.

[59] Y. Zhong, Y. Fan, K. Huang, W. Tan, and J. Zhang, "Time-Aware Service Recommendation for Mashup Creation," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 356–368, 2015.

[60] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola, "Moses: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems," *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1138–1159, 2012.

[61] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive Self-Adaptation under Uncertainty: a Probabilistic Model Checking Approach," in *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE2015)*, 2015, pp. 1–12.

[62] W. N. Robinson, "Monitoring Web Service Requirements," in *Proceedings of the 11th IEEE International Requirements Engineering Conference (RE2003)*, 2003, pp. 65–74.

[63] K. Mahbub and G. Spanoudakis, "Run-time Monitoring of Requirements for Systems Composed of Web-Services: Initial implementation and Evaluation Experience," in *Proceedings of the 12th IEEE International Conference on Web Services (ICWS2005)*, 2005, pp. 257–265.

[64] L. Baresi and S. Guinea, "Towards Dynamic Monitoring of WS-BPEL Processes," in *Proceedings of the 3rd International Conference on Service-Oriented Computing (ICSOC2005)*, 2005, pp. 269–282.

[65] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini, "Validation of Web Service Compositions," *IET Software*, vol. 1, no. 6, pp. 219–232, 2007.

[66] L. Baresi and S. Guinea, "Dynamo: Dynamic Monitoring of WS-BPEL Processes," in *Proceedings of the 3rd International Conference on Service-Oriented Computing (ICSOC2005)*, 2005, pp. 478–483.

[67] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-time Monitoring of Instances and Classes of Web Service Compositions," in *Proceedings of the 13th IEEE International Conference on Web Services (ICWS2006)*, 2006, pp. 63–71.

[68] L. Baresi, S. Guinea, M. Pistore, and M. Trainotti, "Dynamo+ Astro: An Integrated Approach for BPEL Monitoring," in *Proceedings of the 7th IEEE International Conference on Web Services (ICWS2009)*, 2009, pp. 230–237.

[69] L. Baresi, S. Guinea, O. Nano, and G. Spanoudakis, "Comprehensive Monitoring of BPEL Processes," *IEEE Internet Computing*, vol. 14, no. 3, pp. 50–57, 2010.

[70] G. Spanoudakis and K. Mahbub, "Non-intrusive Monitoring of Service-Based Systems," *International Journal of Cooperative Information Systems*, vol. 15, no. 3, pp. 325–358, 2006.

[71] P. Cunningham and M. Cunningham, "Filling the Gap between SLA and Monitoring," *Exploiting the Knowledge Economy: Issues, Applications and Case Studies*, vol. 3, pp. 52–59, 2006.

[72] F. Raimondi, J. Skene, and W. Emmerich, "Efficient Online Monitoring of Web Service SLAs," in *Proceedings of the 16th ACM SIGSOFT International*

Symposium on Foundations of Software Engineering (FSE2008), 2008, pp. 170–180.

[73] F. C. Meng, "Comparing the Importance of System Components by Some Structural Characteristics," *IEEE Transactions on Reliability*, vol. 45, no. 1, pp. 59–65, 1996.

[74] J. Freixas and M. Pons, "Identifying Optimal Components in a Reliability System," *IEEE Transactions on Reliability*, vol. 57, no. 1, pp. 163–170, 2008.

[75] H. Peng, D. W. Coit, and Q. Feng, "Component Reliability Criticality or Importance Measures for Systems with Degrading Components," *IEEE Transactions on Reliability*, vol. 61, no. 1, pp. 4–12, 2012.

[76] Q. He, J. Han, Y. Yang, J.-G. Schneider, H. Jin, and S. Versteeg, "Probabilistic Critical Path Identification for Cost-effective Monitoring of Service-based Systems," in *Proceedings of the 9th IEEE International Conference on Service Computing (SCC2012)*, 2012, pp. 178–185.

[77] D. E. Eckhardt, A. K. Caglayan, J. C. Knight, L. D. Lee, D. F. McAllister, M. Vouk, and J. P. Kelly, "An Experimental Evaluation of Software Redundancy as a Strategy for Improving Reliability," *IEEE Transactions on Software Engineering*, vol. 17, no. 7, pp. 692–702, 1991.

[78] A. Gorbenko, V. Kharchenko, and A. Romanovsky, "Using Inherent Service Redundancy and Diversity to Ensure Web Services Dependability," in *Methods, Models and Tools for Fault Tolerance*, 2009, pp. 324–341.

[79] Z. Zheng and M. R. Lyu, "Selecting an Optimal Fault Tolerance Strategy for Reliable Service-Oriented Systems with Local and Global Constraints," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 219–232, 2015.

[80] Y. Wang, Q. He, D. Ye, and Y. Yang, "Formulating Criticality-Based Cost-Effective Fault Tolerance Strategies for Multi-Tenant Service-

Based Systems," *IEEE Transactions on Software Engineering*, 2017, http://dx.doi.org/10.1109/TSE.2017.2681667.

[81] Y. Wang, Q. He, X. Zhang, D. Ye, and Y. Yang, "Efficient QoS-Aware Service Recommendation for Multi-Tenant Service-Based Systems in Cloud," *IEEE Transactions on Services Computing*, 2017, http://dx.doi.org/10.1109/TSC.2017.2761346.

[82] Y. Wang, Q. He, D. Ye, and Y. Yang, "Formulating Criticality-Based Cost-Effective Monitoring Strategy for Multi-Tenant Service-Based Systems," in *Proceedings of the 24th IEEE International Conference on Web Services (ICWS2017)*, 2017, pp. 325–332.

[83] M. Zeleny and J. L. Cochrane, *Multiple Criteria Decision Making*. University of South Carolina Press, 1973.

[84] C. Wu, W. Qiu, Z. Zheng, X. Wang, and X. Yang, "QoS Prediction of Web Services based on Two-phase K-Means Clustering," in *Proceedings of the 22nd IEEE International Conference on Web Services (ICWS2015)*, 2015, pp. 161–168.

[85] C. Yu and L. Huang, "CluCF: A Clustering CF Algorithm to Address Data Sparsity Problem," *Service Oriented Computing and Applications*, vol. 11, no. 1, pp. 33–45, 2017.

[86] A. K. Jain, "Data Clustering: 50 Years Beyond K-Means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[87] Y. Liu, A. H. Ngu, and L. Z. Zeng, "QoS Computation and Policing in Dynamic Web Service Selection," in *Proceedings of the 13th International on World Wide Web Conference on Alternate Track Papers & Posters*, 2004, pp. 66–73.

[88] E. Al-Masri and Q. H. Mahmoud, "Discovering the Best Web Service," in *Proceedings of the 16th International Conference on World Wide Web (WWW2007)*, 2007, pp. 1257–1258.

[89] Y. Wang, Q. He, D. Ye, and Y. Yang, "Service Selection based on Correlated QoS Requirements," in *Proceedings of the 14th IEEE International Conference on Service Computing (SCC2017)*, 2017, pp. 241–248.

[90] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola, *Global Sensitivity Analysis: the Primer*. John Wiley & Sons, 2008.

[91] G. N. Rodrigues, D. S. Rosenblum, and S. Uchitel, "Sensitivity Analysis for a Scenario-based Reliability Prediction Model," in *Proceedings of the 2005 Workshop on Architecting Dependable Systems*, 2005, pp. 1–5.

[92] M. Harman, J. Krinke, I. Medina-Bulo, F. Palomo-Lozano, J. Ren, and S. Yoo, "Exact Scalable Sensitivity Analysis for the Next Release Problem," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 23, no. 2, article 19, 2014.

[93] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific Belmont, 1999.

[94] P. T. Boggs and J. W. Tolle, "Sequential Quadratic Programming," *Acta Numerica*, vol. 4, pp. 1–51, 1995.

[95] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[96] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*. Cambridge University Press, 2014.

[97] A. Andoni and P. Indyk, "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions," in *Proceedings of the 47th An-*

nual IEEE Symposium on Foundations of Computer Science (FOCS2006)., 2006, pp. 459–468.

[98] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise Independent Permutations," in *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 1998, pp. 327–336.

[99] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-Probe LSH: Efficient Indexing for High-Dimensional Similarity Search," in *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB2007)*, 2007, pp. 950–961.

[100] A. Avizienis, "The N-Version Approach to Fault-Tolerant Software," *IEEE Transactions on Software Engineering*, vol. 11, no. 12, p. 1491, 1985.

[101] T. J. Shimeall and N. G. Leveson, "An Empirical Comparison of Software Fault Tolerance and Fault Elimination," *IEEE Transactions on Software Engineering*, vol. 17, no. 2, pp. 173–182, 1991.

[102] B. Randell, "System Structure for Software Fault Tolerance," in *ACM SIGPLAN Notices*, vol. 10, no. 6, 1975, pp. 437–449.

Appendices

Appendix A

Notation summary

Table A.1: Notation Summary for This Thesis.

Symbol	Description
$B_{i,j}$	The monitoring benefits for the j^{th} local monitoring strategy for the i^{th} component service.
C_p^{neg}	The p^{th} constraint of a negative quality parameter.
C_p^{pos}	The p^{th} constraint of a positive quality parameter.
$cr^O(s_i)$	The overall criticality of s_i.
$cr^T(s_i)$	The tenant-based criticality of s_i.
cr_p^{QN}	The minimum criticality for the p^{th} quality parameter.
cr_p^{QX}	The maximum criticality for the p^{th} quality parameter.
$cr_p^Q(s_i)$	The criticality of the p^{th} quality parameter of the i^{th} component service.
$crn_p^Q(s_i)$	The normalised p^{th} quality-based criticality of the i^{th} component service.
ep_i	The i^{th} execution path.

Continuation of Table A.1	
Symbol	**Description**
epl_i	The i^{th} execution plan.
$F_{i,j}$	The monitoring frequency for the j^{th} local monitoring strategy for the i^{th} component service.
FTC4MTS	Fault-Tolerance based on Criticality for Multi-Tenant SBSs.
$G_{i,j}$	The monitoring granularity for the j^{th} local monitoring strategy for the i^{th} component service.
k_r	The number of redundant services in a fault tolerance scheme.
LQM4MTS	Lifetime Quality Management For Multi-Tenant Service-based Systems.
$ms_{i,j}$	The j^{th} local monitoring strategy for the i^{th} component service.
$mu(ms_{i,j})$	The monitoring utility of local monitoring strategy $ms_{i,j}$.
$mu(\mathbb{S})$	The monitoring utility of SBS $\mathbb{S}$.
$MS_{i,j}$	The member services of the j^{th} service redundancy scheme in the i^{th} service class.
MS_λ	The λ^{th} non-empty subset of $MS_{i,j}$.
$N_{i,j}$	The number of monitors for the j^{th} local monitoring strategy for the i^{th} component service.
O	System overhead caused by service monitoring.
$p(b_i)$	The execution probability of the i^{th} branch in a composition pattern.
$p(epl_e)$	The execution probability of the e^{th} execution plan.
$q_p(s_{i,j})$	The p^{th} quality value of service $s_{i,j}$.
$Q_p(s_{i,j})$	The normalised p^{th} quality value of service $s_{i,j}$.
$q_p^{max}(sc_i)$	The maximum quality value for the p^{th} quality parameter in the i^{th} service class.
q_p^{dc}	The quality degradation coefficient of the p^{th} quality parameter.

Continuation of Table A.1	
Symbol	**Description**
$q_p^{min}(sc_i)$	The minimum quality value for the p^{th} quality parameter in the i^{th} service class.
$qn_p(s_{i,j})$	The p^{th} normalised quality value of $s_{i,j}$.
$q_p(csc_{i,l})$	The quality value of the p^{th} dimension of the mapped centroid of the l^{th} service cluster in the i^{th} service class.
$QSD(s)$	The average quality satisfaction degree of service composition s to all tenants.
$QSD(s, t_i)$	The quality satisfaction degree of service composition s to a tenant t_i.
$r_p^{max}(T)$	The maximum values for the p^{th} quality requirements of all tenants that share SBS $\mathbb{S}$.
$r_p^{min}(T)$	The minimum values for the p^{th} quality requirements of all tenants that share SBS $\mathbb{S}$.
$r_p(t_e)$	The value of the p^{th} quality requirement of the e^{th} tenant for SBS $\mathbb{S}$.
$rn_p(t_e)$	The normalised value of the p^{th} quality requirement of the e^{th} tenant for SBS $\mathbb{S}$.
R	Resource cost for service monitoring.
$\mathbb{R}^d$	A d-dimensional space.
$\xi(\mathbb{S})$	The number of service requests that SBS $\mathbb{S}$ processes per unit of time.
$\xi(s_i)$	The number of service requests that the i^{th} component service processes per unit of time.
$\mathbb{S}$	The SBS.
s_{i,j_r}	The r^{th} redundant service in $SRS_{i,j}$.
$s_{i,j}$	The j^{th} candidate service in the i^{th} service class.

Continuation of Table A.1	
Symbol	**Description**
s_i	The i^{th} component service.
sc_i	The i^{th} service class.
SAL4MTS	Service Adaptation based on LSH for Multi-Tenant SBSs.
SMC4MTS	Service Monitoring based on Criticality for Multi-Tenant SBSs.
$SRS_{i,j}$	The j^{th} service redundancy scheme in the i^{th} service class.
SSC4MTS	Service Selection based on Correlated quality requirements for Multi-Tenant SBSs.
SSR4MTS	Service Selection based on service Recommendation for Multi-Tenant SBSs.
$T(tc_c)$	The number of tenants in the c^{th} cluster.
$\tau(\$)$	The number of tenants sharing $\$$.
$\tau(s_i)$	The number of tenants sharing s_i.
t_i	The number of tenants that share service class SC_i.
$u(s_{i,j})$	The utility of service $s_{i,j}$.
$w_{e,p}$	The e^{th} tenant's weight for the p^{th} quality dimension.
$w_{t,p}$	The t^{th} tenant's preference for the p^{th} quality parameter.
w_p^{ave}	The tenants' average preference for the p^{th} quality parameter.
Δq_p	The percentage of quality degradation for the p^{th} quality parameter in each iteration.
ΔQ_p	The difference in the p^{th} normalised quality value of tenant t_i's requirement between f and service composition s. f is the perpendicular foot of s to the line segment that represents the quality correlation function of tenant t_1.
$\Delta q_{k,p}^{vp}(s_i)$	The variation percentage of the p^{th} quality value of s_i in the k^{th} iteration.

Continuation of Table A.1	
Symbol	**Description**
$\Delta q_{k,p}^{vp}(\mathbb{S}, s_i)$	The variation percentage of the p^{th} quality value of $\mathbb{S}$ in the k^{th} iteration.
ν	The round in which the optimisation is performed.
ν^{max}	The maximum value of ν.