ADAPTIVELY IMPROVING PERFORMANCE STABILITY OF CLOUD BASED APPLICATION USING THE MODERN PORTFOLIO THEORY

by

FAISAL ALREBIESH

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
February 2016

Abstract

The increasing number of Software-as-a-Service(SaaS) services available in the cloud market make them plausible and attractive for building cloud-based applications. However, performance instability is common in the cloud environment due to changes in supply and demand of shared computational infrastructure and resources. Candidate services are vulnerable to such instability. Current service selection and composition approaches do not explicitly address performance fluctuations when building cloud-based applications. This thesis proposes a novel approach to improve performance stability by leveraging on the principles of design diversity and portfolio-based thinking when selecting and composing cloud-based applications. The objective is to minimize the risks that could stem from selecting and composing cloud-based services that are vulnerable to performance instability.

More specifically, we present a self-adaptive approach which leverages the principle of Modern Portfolio Theory to construct a diversified set of candidate services that share the lowest possible correlation between their performances. The self-adaptive approach makes an explicit trade-off between the costs, benefits and likely risks when performing changes to the cloud-based applications.

In this thesis, we use two scenarios to illustrate the applicability and the effectiveness of the approach. As scalability is of paramount importance for efficient dynamic and adaptive selection and composition, the thesis adapt a systematic method to identify the various scalability dimensions that can affect the working of the approach and consequently evaluate the sensitivity of the approach to the identified dimensions. The thesis concludes with possible directions for future work.

Acknowledgements

In the name of Allah , the Most Gracious and the Most Merciful.

First and foremost, Alhamdulillah, all praises to Allah for the strengths and His blessing in completing this thesis. Then, I would like to express my sincere gratitude to my advisor Dr. Rami Bahsoon for the continuous support he rendered to me towards my Ph.D. study and related researches. Without his patience, motivation and guiding hand, this thesis would not be what it is today. I could not have imagined having a better advisor and mentor for my Ph.D. study.

Besides my advisor, I would like to thank the rest of my RSMG members: Dr. Rustam stolkin, and Prof. Xin Yao, for their insightful comments and encouragement, but also for the hard questions which incented me to widen my research from various perspectives. in addition, A very special thanks goes out to Dr. Letcia Duboc for the comments and suggestions she made in reference to Chapter 6 of this work.

Daily life in Computer Science has been a pleasure, because of all the friends that I made during the course of the PhD. The heated discussions, informative comments and passionate arguments that Ive had with Abdessalam Elhabbash, Momodou L Sanyang, Mohammad Alshammari ,Mohammed Alwanain, and many others have contributed to a rich, and fulfilling period of stay, in this institution. I should not forget to acknowledge brother Osamah and sisters Lama and Maha for their continuous love, support, understanding and good wishes whenever I needed.

Finally, I would like to thank my wife and best friend, Saja. Without her love, encouragement, support and editing assistance, I would not have finished this thesis. Her tolerance of my occasional mood swings is a testament in itself of her unyielding devotion and love. A debt of gratitude, more than I can ever repay, is owed to my parents Mohammed and Hessah. Each year that I grow older, I realize just how much I owe you, and how much of what I am, is due to you. You have been patient, indulgent and loving,

to a fault. Youve supported me through, what must have seemed like, crazy dreams and ambitions. I shall not attempt to thank you, because words will not be enough. I love you, and to you both, I dedicate this thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Enterprises are always searching for efficient and effective approaches for engineering software systems that maximize their profits while reducing their operational cost. They are looking for architectures that allow them to scale their operations without costing them a fortune on capital expenditure.

Cloud computing has emerged as a promising computing model for providing an affordable on-demand access to shared amount of computing power, storage and bandwidth. An important selling point of cloud computing is adopting the *pay-as-you-go* model based on the access and use of shared resources. As a result, cloud computing has introduced a new way to deliver IT services and architect software systems, bringing the convenient of traditional public utilities, such as electricity and water for computer users.

Another selling point relates to the potential scalability that the cloud can support when building cloud-based architectures. This is attributed to the elasticity primitives and on-demand access to the pool of shared resources benefiting from the economies of scale. In particular, cloud-based architectures can scale up, to accommodate growing load, by simply providing more cloud-based resources. On the other hand, when demand decreases, the cloud-based application can scale down by releasing the unutilised cloud-based resources. The distinctive advantages of cloud computing such as on-demand access, potential scalability and cost efficiency have encouraged service providers and software system architects to adopt the cloud computing model and to benefit from the types of services offered by the cloud. Among the services provided by the cloud, software system architect can gain, from Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) when composing cloud-based applications [1].

- *Software-as-a-Service (SaaS):* In this model, SaaS providers provision services on cloud infrastructure. The architect can compose a number of these services into an application by using a set of interfaces and well-defined APIs. The architect does not have any control over the underlying cloud infrastructure, such as processing power, virtual machines or storage. Payment for this model is often calculated on the basis of usage and subscription fee. Typical examples of SaaS providers are the Salesforce AppCloud and Amazon web services.

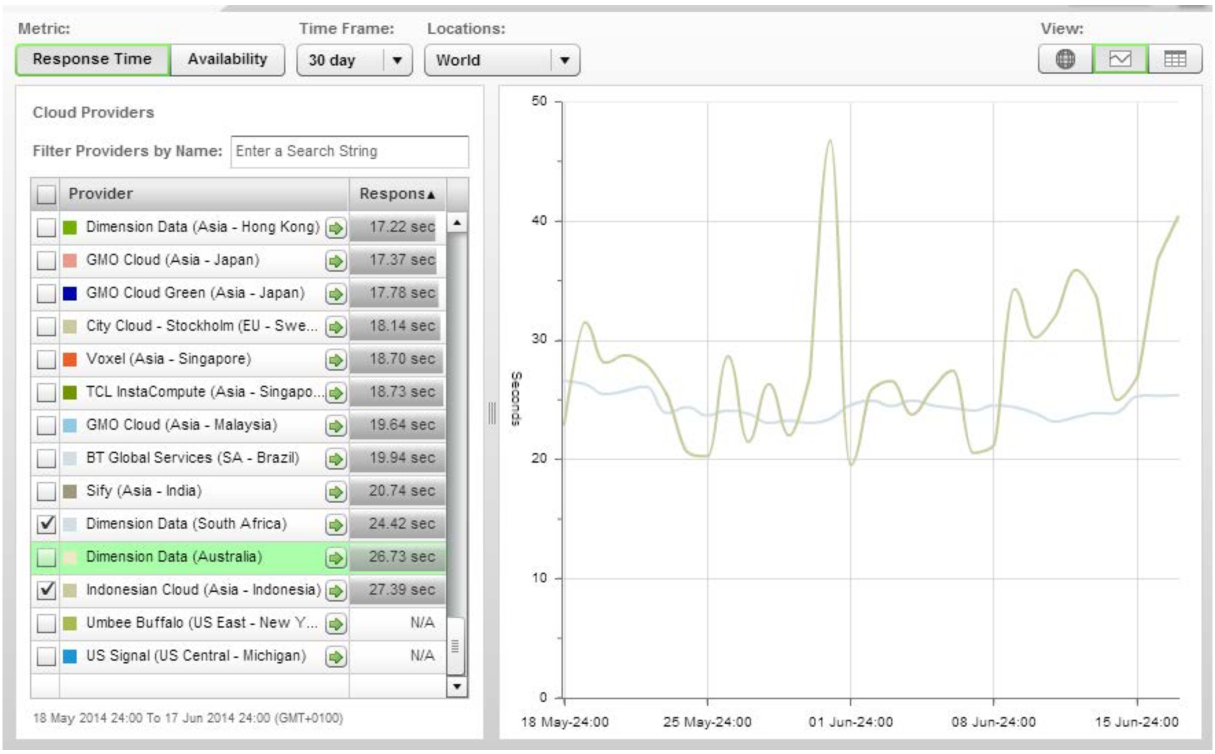
- *Platform-as-a-Service (PaaS)*: PaaS providers offer the architect a combination of middleware and deployment environment that facilitate the building and deployment of a cloud-based application. As a result, the complexity of managing the underlying middleware and hardware is transparent to the application developer. A representative example of PaaS is the Microsoft's Azure, which provides a .NET environment to application developers.

- *Infrastructure-as-a-Service (IaaS):* IaaS is recognized as the most basic form of service provided in the cloud model. IaaS providers offer computing resources such as virtual machines, virtual storage and networking. Unlike previous cloud services models, the architect in the IaaS model can benefit from various types of systems related services, such as middleware services. While IaaS provider is responsible for maintaining the hardware infrastructure, the atchitect can configure and maintain the software environment. Typical examples of IaaS providers include Google Compute Engine and Amazon Elastic Compute Cloud (EC2).

The cloud model has provided the architect with the flexibility to build cloud-based applications benefiting from SaaS offered by multiple providers. Although SaaS provision is governed and its use is mandated by Service Level Agreement (SLA), yet providers can not always guarantee the delivered Quality of Service (QoS) of the SaaS. This is attributed to dynamic, shared and on-demand nature of the cloud, where the demand on services and its underlying resources tend to fluctuate. By QoS, we refer to non-functional requirements such as availability, performance, security and so forth. These QoS are fundamental to users satisfaction and the working of an architecture [2].

We posit that performance is one of the critical dimensions for the satisfaction of QoS. It criticality stems from the fact that offering SaaS services through a cloud-based market comes with underlying risks that relates to performance fluctuation. This is due to the undependable service provision of the cloud service provider, hardware malfunctions, unpredicted fluctuations in demand for the traded services and shared resource, etc. All these factors may increase the uncertainty and risks associated with performance fluctuations benefiting from the cloud-based market.

By envisaging Figure 1.1 as a motivating example, one can see that performance fluctuation can vary from services offered by one provider to another. Although some providers tend to provide a relatively stable performance(e.g. the case of Dimension Data South Africa), yet others are vulnerable to highly fluctuating performance (e.g. the case of Indonesian Cloud). Existing service selection solutions focus on ensuring QoS properties of services, such as cost, privacy and security. By contrast, the risk of performance fluctuation, arising from the uncertainty in the cloud environment, is largely ignored. It is necessary to consider how to reduce the performance fluctuation when building application using SaaS from the cloud environment and to make fluctuation explicit concern when architecting cloud-based solutions.



☐		Provider	Respons▲
☐	🟩	Dimension Data (Asia - Hong Kong)	17.22 sec
☐	🟥	GMO Cloud (Asia - Japan)	17.37 sec
☐	🟦	GMO Cloud Green (Asia - Japan)	17.78 sec
☐	■	City Cloud - Stockholm (EU - Swe...	18.14 sec
☐	🟧	Voxel (Asia - Singapore)	18.70 sec
☐	🟩	TCL InstaCompute (Asia - Singapo...	18.73 sec
☐	🟦	GMO Cloud (Asia - Malaysia)	19.64 sec
☐	■	BT Global Services (SA - Brazil)	19.94 sec
☐	■	Sify (Asia - India)	20.74 sec
✓	■	Dimension Data (South Africa)	24.42 sec
☐	■	Dimension Data (Australia)	26.73 sec
✓	■	Indonesian Cloud (Asia - Indonesia)	27.39 sec
☐	🟩	Umbee Buffalo (US East - New Y...	N/A
☐	🟦	US Signal (US Central - Michigan)	N/A

18 May 2014 24:00 To 17 Jun 2014 24:00 (GMT+0100)

Figure 1.1: The Historical Record of Performance for Two Cloud Providers Dimension Data-South Africa and Indonesian Cloud From the 18/5/2014 to 16 /6/2014 [3].

1.1 Problem Definition

As we discuss earlier, cloud computing promises the delivery of scalable, affordable and on-demand resources [4], which encourages web service vendors to supply their services through a cloud market [5]. The popularity of the cloud and its distinctive advantages make cloud-based web services a plausible and attractive option for architecting cloud-based applications. Conversely, performance fluctuation is common in the cloud environment due to changes in supply and demand of shared computational infrastructure and resources [6].

Several advances in the web services field have facilitated the automated discovery, browsing and integration of web services. Among such advances are the introduction of languages such as Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP), the Universal Description, Discovery, and Integration (UDDI) [7] and ones that focus on the integration and binding of service compositions, called Business

Process Execution Language for Web Service (BPEL4WS). Despite all these advances in the field of web services, the selection of an optimal set of services to build cloud-based applications is still a highly complex task for the following four reasons.

Firstly, SaaS services hosted on the cloud are vulnerable to performance fluctuations because of either limited shared resources or changes in user demands. In a performance analysis study that covered Amazon EC2 cloud, Dejon et al. [8] observed severe performance drops, which caused the response time to vary between 200 ms and 900 ms, with a mean of roughly 500 ms and a standard deviation of about 200 ms. Candidate cloud-based applications are expected to be vulnerable to such instability. For that reason, we need to consider how to evaluate and improve the performance stability by reducing the performance fluctuation of the cloud-based application through the selection of an optimal set of candidate services.

Secondly, there are an increasing number of available candidate services offering similar functionalities with different QoS. It is hard to select an optimal service selection from a large pool of candidate services that satisfies the user's needs.

Thirdly, as the cloud is a dynamic environment, where services are added and moved from the market at run time, self-adaptivity becomes a key requirement when architecting a QoS-aware solution that benefits from the cloud market. Among the concerns, self-adaptivity shall be concerned with selection and consequently composition of these services in response to changes in the dynamic cloud environment.

Fourthly, optimizing QoS in a dynamic services composition is a NP-hard problem [9]. A significant challenge is to understand how services selection composition approaches can scale with respect to a number of different scaling dimensions, such as workflow size, the number of QoS constraints, the number of application requests and so forth.

The literature offers a large number of approaches for web service selection and composition that operate on the cloud. Most of these approaches focus on finding an optimal set of services based on constraints on the QoS of the candidates [9], [10], [11], [12], [13] , [14] [15] or on the reputation based on user feedback, such as [16], [17], [18] and [19].

However, they have not explicitly considered the performance stability of the cloud-based application in their models.

We posit that the principles of design diversity can provide a sensible solution to deal with performance instability when selecting and consequently composing architectures that relies on cloud-based services. Diverting from the cloud, the software engineering literature provides plenty of examples of how the principles of design diversity have attempted to improve the software reliability and dependability. In particular, design diversity techniques have been motivated and used as defence against uncertainty by using different variants of a program that provide similar functionalities. When a problem occurs to one of the variants, there is a chance that it will not affect the others. Other variants can continue to deliver the required functionalities entailed by the system specification. Classical approaches implement design diversity techniques such as [20], [21] and geographical diversity, such as [22] and [23]. However, none of these techniques have explicitly linked the selection of service composition to performance fluctuation and performance correlations between different candidate services offered in the cloud market.

This thesis objective is to design a stability-aware services selection and composition approach to adaptively reduce the performance fluctuation of a cloud-based application using the concept of design diversity. The challenge is mainly in two-folds: (1) find a mechanism to self-adaptively select a set of services that help to reduce the performance fluctuation of a cloud-based application and (2) find an efficient diversified solution while considering the performance fluctuation and the correlation between candidate services of a composition. To address these challenges, we also investigate whether composing services with a low-performance correlation can result in better performance stability of the cloud based application. In the following sections, we will introduce two application scenarios, which will help exemplify the problem in more details.

1.1.1 Service Selection In Scaling Up Scenario

As an illustrative example, let us consider a budget flight booking cloud-based application, Flight.com, which provides an online booking web service. We assume that all the required Flight.com functionalities can be satisfied by selecting single stand-alone cloud service named FlightBooking. Different cloud providers offer variants of the FlightBooking service in the cloud market. The variants tend to provide the same core functionalities, but they differ in price and the way they deal with QoS. In high seasons, Flight.com has decided to scale up its services to support an anticipated load in the number of users through selecting and subsequently allocating additional 100 instances of the FlightBooking services through a cloud-based market.

By using traditional auction-based methods as in (e.g., [24], [25]), the 100 instances of web services could be allocated from the sellers with the lowest price without giving much consideration to the risk of performance fluctuation. In contrast, our aim is to improve its performance stability application by diversifying the selection of 100 instances of the FlightBooking web services from multiple cloud sellers. The objective is to self-adaptively reduce risks associated with the fluctuation performance (measured by the throughput of the selected services) in the cloud-based application while maintaining a number of QoS constraints.

1.1.2 Cloud Services Composition Scenario

In the previous scenario, we considered a sample application that requires a single service to satisfy all of its functional requirements. However, here we address more complex cloud-based application that requires the integration of different web services to create an added value composition of services that satisfies the functional and QoS requirements. We refer to the process of selecting and allocating a number of interconnected cloud-based services as *Cloud Services Composition*(CSC).

Let us consider the following example, Mytrip.com, which is a CSC application that provides an online service for booking travel packages. In order to provide the travel packages, Mytrip.com constructs CSC with three cloud services that include Flight Booking, Car Booking and Hotel Booking services. We assume that each of the abstract web services can have multiple candidate web service instances, which can be leased from various clouds. As in the previous scenario, we assumed that the variants tend to provide the same core functionalities, but they differ in the way they deal with QoS (response time, security, and price).

A representation of a CSC problem is shown in Fig. 1.2. The aim is to design cloud services composition algorithm that improves the stability of CSC performance (measured by the response time of the selected CSC) while maintaining a number of QoS constraints. The self-adaptive mechanism will help the CSC to react to changes in the market and sustain the optimality of the CSC in a runtime environment. As QoS dynamic services composition is known to be a NP-hard problem [9], scalability is a paramount importance for the efficiency and effectiveness of the composition solution. For this reason, we support the approach with systematic scalability analysis method that can better understand how the composition technique react to changes in a variety of scalability dimensions for the CSC.

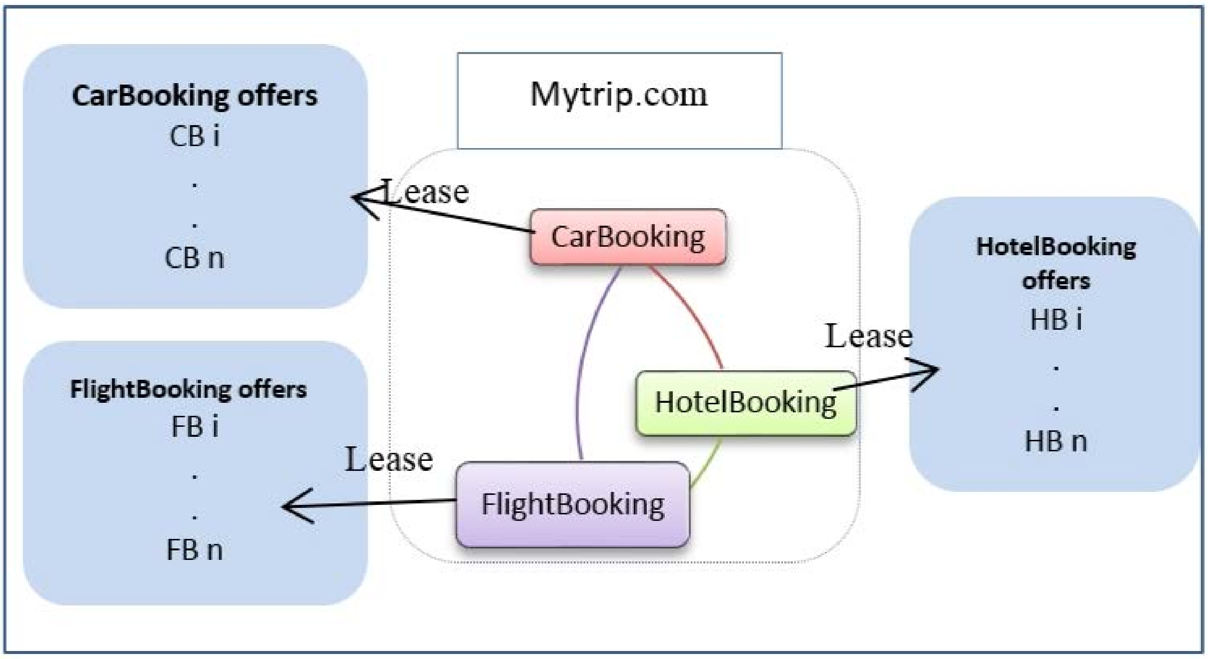


Figure 1.2: A Representation of the Cloud Service Composition (CSC) Problem.

1.2 Proposed Solution

This thesis views the cloud as a marketplace for trading instances of web services, which cloud-based applications can explore, trade and use as substitutable and composable entities in the architecture of a cloud-based service applications. That is, for a given abstract service S in a cloud-based service application, there exist multiple candidate services, $S_i......S_n$ in the market offering comparable functionality, but differing in their price and QoS provisions. This market-oriented perspective has been widely adopted by researchers and also constituted one of the enablers for the dynamic service selection and composition vision and automatic service operations. The perspective adheres to widely accepted practices as documented in the literature (e.g. [26], [27] [28] and [29]).

We view the selection of web service instances to architect an application from a cloud-based market as an optimisation problem, where the goal is to reduce probable risks of performance fluctuation while maintaining a set of QoS requirements. We look at such optimisation from the buyers (i.e. a cloud-based service application) perspective.

From a novel perspective, our stability-aware services composition advocates for the use of design diversity principles to reduce the performance fluctuation of cloud-based application. The idea is that by diversifying candidate services that share lowest possible correlation between their performances, we can improve performance stability and reduce the chance of performance fluctuation. We argue that a cloud-based application can utilize the Modem Portfolio Theory [30] to build a diversified portfolio of multiple instances of web services. The logic of the portfolio theory promotes the process of diversification, known as *'not putting all of your eggs in one basket'*, as a way to reduce the risk of the portfolio [31]. In the context of web services selection, diversification can be achieved by selecting web services from multiple providers that share a low correlation between their performances. In the case of low correlation, if the underlying factor for a drop on the services performance affects one provider, there is a chance that it will not occur in the other providers.

This is because cloud providers may vary in terms of the software, hardware, operating systems, virtualization mechanisms and physical location used among the other environmental factors. Assuming that the service provision tends to be functionally identical across candidate providers, the QoS characteristics tend to be sensitive to the underlying resources deployed to support the running of these instances. Henceforth, the strategy for considering more than one provider leans towards diversity. Unlike the reviewed classical design diversity, our portfolio-based approach links the diversification of candidate services to performance fluctuation and correlation between different candidate services. To the best of our knowledge, we are not aware of any service composition approach that explicates diversity in cloud-based application by the selection of sets of candidate cloud services with the aim of reducing the performance fluctuation.

In summary, the thesis attempts to answer the following research questions:

- *RQ1: Can the concept of design diversity be applicable to the case of cloud services selection and composition to reduce risk of performance fluctuation? How well can it perform compared to well-established services selection methods?*

- *RQ2: How can the approach be extended to self-adaptive mechanism, which can* *dynamically respond to changes in the market?*

- *RQ3: In the case of CSC, what are the scaling dimensions (e.g. number of web services, number of objectives, candidate solutions, frequency and volatility of change etc.) that we need to render a pragmatic solution ?*

1.3 Contributions of This Thesis

This thesis makes a number of novel contributions towards the objective of a stability-aware services selection and composition technique for the cloud environment. In particular, the thesis investigates how leveraging on the principles of design diversity, portfolio thinking and self-adaptivity can lead to a pragmatic technique that can serve the problem and stabilise performance in the cloud. The major contributions are as the following:

- **A literature review that covers the state of the art of QoS-aware service composition**: We review existing work on QoS-aware services composition. The objective of the review is to draw from the state of the art approaches, new insights that can assist the stability-aware selection of cloud-based services with the aim of reducing performance fluctuations. The review helped to identify a list of major challenges imposed by cloud environment on QoS-aware service composition. These challenges lead to the design of our novel portfolio inspired mechanism.

- **Review the existing design diversity solution**: We present a review that explores how design diversity is adopted in different solutions to improve the dependability and reliability of software systems. The aim of the review is to identify the advantages and limitations of current design diversity solutions and to motivate the need for a new approach to implementing diversity in service selection and composition.

- **A novel portfolio-based service selection algorithm**: We present a novel stability-aware service selection and composition algorithm that can be used to build cloud-based applications. The approach utilises the Modern Portfolio Theory to allow cloud-based applications to minimize the performance fluctuation while maintaining a set of required QoS constraints. The thesis uses two scenarios of application to illustrate the applicability and the effectiveness of the portfolio-based algorithm in improving performance stability.

- **Systematic Elaboration of Scalability Requirements for Portfolio-based Service Composition:** We adapt systemic methodology to conduct a scalability analysis for our approach. More specifically, this systemic methodology helps to identify the scalability dimensions of QoS-aware service composition that are more likely to affect the scalability of our approach. The used method has helped to reveal four scalability dimensions related to our scalability evaluation. These dimensions are the number of candidate services, the size of application workflow, the number of concurrent requests and the number of QoS in the application.

- **Conducting A Systematic Scalability Analysis**: By using controlled experiments, we systematically perform a scalability evaluation on the portfolio-based composition where we evaluate the sensitivity of time needed to find a solution to increases of the four scalability dimensions. This form of scalability evaluation can benefit other QoS-aware service composition approaches, which which require scalability analysis.

1.4 The Thesis Storyline

A survey of the QoS-aware service composition solution indicates that current approaches focus explicitly on QoS dimensions such as performance, cost, and security and only implicitly, if at all, on the performance fluctuation. Despite their concern with performance, these methods do not address the performance stability. Then, we looked at classical approaches that adopt the concept of design diversity to improve the reliability of application and motivate the need to a diversification technique that address the issue of correlated failures.

To bridge the gap, this thesis proposes an economics-driven approach for evaluating and minimising the performance fluctuation of a cloud-based application. It is assumed that architect of the application (i.e. a buyer) is a risk-averse investor that aims to select set of services that minimise the application performance fluctuation. The thesis then claims that using design diversity can help in reducing such fluctuation. In particular, the thesis argues that Modern Portfolio Theory ([32], [33]) is suited for assisting in the evaluation and minimisation performance fluctuation of cloud-based application. However, this raises the question: Why Modern Portfolio Theory? Portfolio theory was developed to deal with uncertainties revolving around investments and future returns in the financial markets. The theory presents a framework to form a stable portfolio of investment from set stocks that suffer price fluctuation [30]. This perspective is appealing to the problem of building a stable cloud-based application using a set of cloud services that suffer from performance fluctuation.

This thesis presents a portfolio-based services selection model that helps in evaluating and minimising the performance fluctuation of an application. Briefly, the model draws on a simple analogy with Modem Portfolio Theory, where services in the market present the investable asset and the cloud-based application as investor who owns the portfolio of services. The novelty of our portfolio-based model emanates from the ability of well-diversified portfolio by selecting an optimal set of services that share a minimum correlation between their performances to achieve more stable performance. The thesis describes how we have derived the portfolio-based services selection model: the analogy made, its formulation, the assumptions and its effectiveness by report on two scenarios of application for the model: services selection in scaling up scenario and a cloud services composition scenario.

The thesis complements the model with a self-adaptive mechanism that utilised the Monitor, Analyse, Plan and Execute (MAPE) control loop [34] to react to changes in runtime environment. The proposed mechanism makes an explicit trade-off between the cost and benefit of performing changes to the cloud-based application.

As scalability is of paramount importance for efficient dynamic and adaptive selection and composition, the thesis adapts the systematic method of [35] to identify the various scalability dimensions that can affect the working of the approach. We report on set of control experiments that evaluate the sensitivity of the approach to the identified scaling dimensions.

The thesis uses a set of control experiment to empirically evaluate the portfolio-based model and explore its effectiveness in addressing two scenario of applications. In the first scenario, we apply the portfolio-based model for building a simple cloud-based application for a scaling up scenario by selecting a number of identical services. We demonstrated the effectiveness of the self-adaptive portfolio-based selection for building and maintaining an optimal cloud-based application with minimum performance fluctuation.

The second scenario considers a more complex problem, where the CSC application requires the cooperation of multiple interconnected cloud services to satisfy their requirements. A set of simulated experiments were used to test the approach effectiveness in improving the performance stability of the CSC, analyse the performance of the approach under multiple correlation settings, evaluate the effectiveness of the self-adaptive mechanism in dynamic market; and perform the scalability analysis that covers multiple dimensions of the CSC problem.

In addition to the simulated experiments, we have implemented a prototype where the whole technique was realised in CloudSim environment [36]. The prototype shows consistent result with the findings of the simulated experiments, where the portfolio-based CSC outperforms the other composition algorithms in terms of the quality of selection (minimum performance fluctuation).

The thesis concludes by highlighting some open questions that could stimulate future research in stability-aware services selection and compositions.

1.5 Structure of The Thesis

The rest of the thesis is structured as following:

In Chapter 2: we start by presenting an overview that covers some of the basic concepts related to web service and web services composition. After that, we present a detailed survey that covers the current QoS-aware service composition approaches. The objective of the survey is mapping out the main activities used to support QoS-aware service composition in a dynamic environment and identify the gaps in current approaches. From the survey, we discovered that the reviewed approaches do not address the issue of performance fluctuation in cloud computing environment.

In Chapter 3: we present an overview that covers the concept of design diversity, followed by a review that explores current approaches of implementing design diversity. The review presents a number of different methods used to implement design diversity as well as presenting the requirements, challenges, benefits and trade-off of implementing design diversity.

In Chapter 4: we use a simple scenario to demonstrate the applicability of the portfolio-based approach. Particularly, we describe how the portfolio-based approach can be used to build a cloud-based application in scaling up a scenario with the aim of reducing the risk of performance fluctuation. First, we present an overview of the theory and some of its related concepts. Second, we present the formulation, analogy and the assumptions associated with the self-adaptive portfolio-based services selection solution. Last, we present a set of controlled experiments used to test the approach effectiveness in minimizing the risk of performance fluctuation.

In chapter 5: we illustrate the efficiency of our portfolio-based approach to tackle complex scenario which is CSC. First, we present adaptive CSC approach that leverages the principle of modem portfolio theory to construct a diversified CSC. Second, we present a set of controlled experiments used to test the approach effectiveness and self-adaptivity in minimizing the risk of performance fluctuation. Finally, prototype of the system is presented.

In Chapter 6: building on the work of the systematic elaboration of scalability requirements [35], we systematically identify the scalability requirements for the complex scenario of CSC. We start by presenting a background that covers part of methods used in our scalability analysis such as Goal-Oriented Modelling and Goals Obstacle Analysis. Secondly, we present the goal modelling and scalability requirements of the CSC problem. Finally, a set of experiment are presented to evaluate the scalability of the portfolio-based composition.

In chapter 7: we conclude the thesis with a discussion of the main findings and concluding thoughts about directions that this research can take, in the future.

1.6 Publications

Work presented in this thesis has been to a degree or completely derived from the following list of papers published during the course of the Ph.D. candidature. This thesis must be considered as the definitive reference of details and ideas, presented in these publications.

- Conference papers

1. F. Alrebeish and R. Bahsoon(2015). **Stabilising Performance of Cloud Services Composition Using Portfolio Theory.** Full Paper in the Research Track. The 22nd IEEE International Conference on Web Services (ICWS) June 27 - July 2, 2015, New York, USA (Selection rate 18%).

2. F. Alrebeish and R. Bahsoon(2013). **Risk-Aware Web Service Allocation in the Cloud Using Portfolio Theory.** Accepted as a full paper. The 10th IEEE International Conference on Services Computing (SCC 2013), In conjunction with IEEE Cloud 2013, ICWS 2013, and Services San Francisco, CA, USA.

3. F. Alrebeish and R. Bahsoon(2013). **Using Portfolio Theory to Diversify the Allocation of Web Services in the Cloud.** Genetic and Evolutionary Computation Conference (GECCO 2013), Amsterdam, the Netherlands. ACM Press.

- Journals

1. F. Alrebeish and R. Bahsoon(2015). **Implementing Design Diversity Using Portfolio Thinking to Dynamically and Adaptively Manage the Allocation of Web Services in the Cloud** (Accepted). *IEEE Transactions on Cloud Computing, vol.3, no.3, pp. 318-331.*

2. F. Alrebeish and R. Bahsoon(2015). **Portfolio-based Self-adaptive Mechanism for Stabilizing Performance of Cloud Service Composition** (Reviewing cycle). *IEEE Transactions on Service Computing.*

CHAPTER 2

QOS-AWARE SERVICE COMPOSITION: *STATE OF THE ART*

One of the critical dimensions which can not be undermined once we compose services is performance. Though there has been plenty of research in QoS-aware composition, these methods treat performance as an add-on dimension and do not explicitly handle its fluctuation overtime. The goal of our research is to present a QoS-aware service composition approach that achieves performance stability in cloud computing environments. The first step towards this goal consist of reviewing the state of the art of QoS-aware service composition.

The objective of the review is to draw from the state of the art of QoS-aware service composition a new insights that can assist the problem of stability-aware dynamic selection for cloud-based applications. In this survey, we reviewed the QoS-aware service composition approaches in the context of the traditional software environment, in general, while paying special interest on QoS-aware service composition approaches in the dynamic cloud environment. In dynamic environment, service composition algorithm can support QoS awareness by providing a set of key activities, such as QoS modelling, service composition and QoS-driven adaptation.

In this chapter, we start by presenting a background information that covers some of the basic concepts relating to web service and web services composition. Then, we will present a review of existing work on QoS-aware services composition. The aim of the review is to identify and present the main activities that support QoS-aware service composition in dynamic environment, such as QoS modelling and description, the scope of QoS constraints, problem modelling, selection strategies and techniques , and support for adaptation. Finally, we will conclude by presenting a list of challenges posed by cloud computing environment on the QoS-aware cloud service composition.

2.1 Basic Concepts and Related Standards to Web Services and Web Services Composition

2.1.1 Web Services

This section will present an overview of concepts related to web services. First, we start by defining what do we mean by web services and then we present the *Web Services Model* followed by a list of XML-Based Standards that are used for describing, discovering, and invoking web services. Finally, we list the benefits of presenting software applications as web services in comparison to the traditional software application.
As a first step towards defining web service, we start by presenting a general definition of the terms service. Then, this general definition is specialized to account for the case of web service. According to Gadrey [37] a service is defined as follow:*"A service is a set of activities that are performed and intended to bring about a change of state to either an entity that is owned or used by a consumer or to the consumer itself. The set of activities are performed by a provider or jointly by the provider and consumer. The outcome or resulting change of state is based upon a prior agreement between the consumer and provider, which aims at the co-creation of value".*

This definition emphasizes two key features of services. First, the intention of changing the state of an entity and secondly, it explicitly requires a prior agreement on terms of the delivered service. It is the responsibility of the services consumer to specify the appropriate terms of services to avoid overprovision. The provider mainly contributes by performing the activities required to satisfy the services agreement. This general definition covers any type of services from building a house, to leasing web services from cloud provider such as Amazon.

With the rise of cloud computing and rapid developments in web technologies, the environment of service provision and delivery has changed fundamentally. In this context, a new kind of services has emerged which is defined as web service. The main advantage of web services is that both of the inputs and outputs of the services are delivered by means of a network like mobile network or the Internet.

According to the World Wide Web Consortium (W3C) [38], a web service is *"a software system designed to support interoperable machine-to-machine interaction over a network. The web service has an interface described in a machine-processable format (specifically WSDL). Other systems should be able to interact with the Web service in a prescribed manner using language such as SOAP-messages"* . These messages are transmitted using HTTP or other web standards. Another definition of web service was presented by Berners-Lee et al [39] where he defined them as *"a software service, which can be accessed using a uniform resource identifier (URI), exposing a public interface based on Internet standards"*.

Both of these definitions stress two features of web service. First, the existence of well-defined interface is required to disclose the service to the public. Second, a web service should use a special web protocol (such as HTTP) to facilitate, communicate, and exchange data between the service provider and consumer. Both of these features are considered as key enablers of automated service selection and composition. A clear demonstration of web services interaction is the web services model presented in Figure 2.1 [40]. The web services model defines three primary activities for interacting with web services. These activities are publish, find and bind web services.

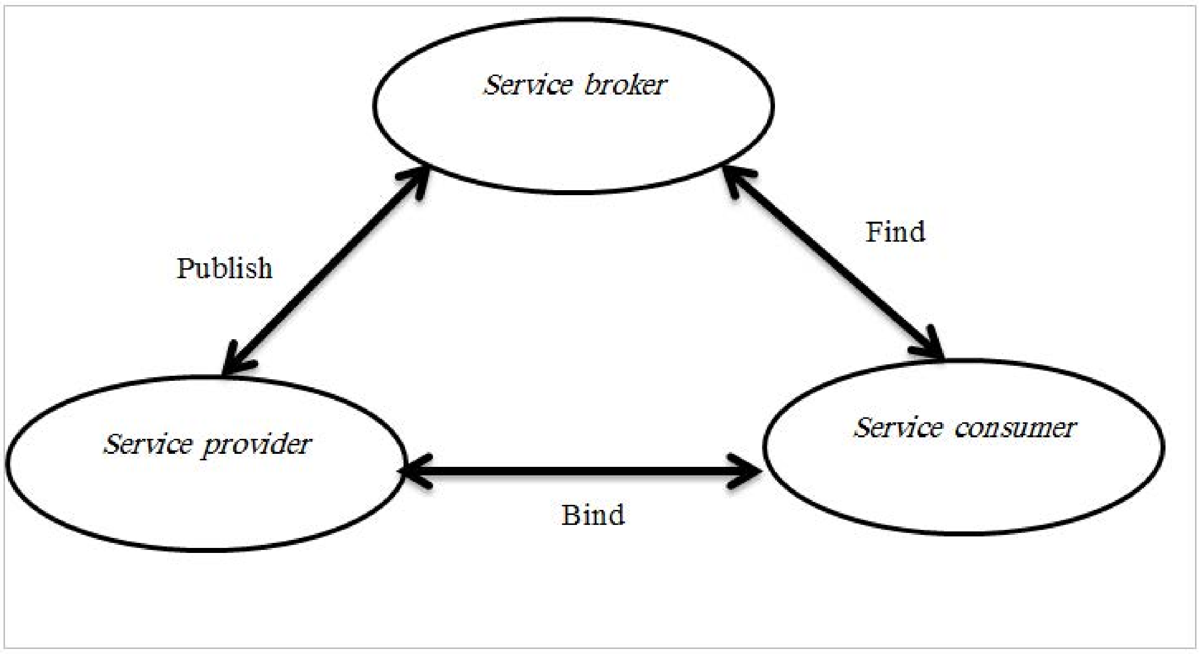


Figure 2.1: Web Service Model

The responsibilities of performing the activities presented in Web Service Model are assigned to three main agents: the service provider, the service broker and the services consumer [40].

- *The service provider* is an agent responsible for providing a specified software application as service. The service provider is also responsible for publishing and updating their services as well as their interface so that they are accessible on the Internet. From a business point of view, it is the owner of the service. From an architectural point of view, it is the platform hosting the implementation of the services.

- *The service consumer* is the agent that requires a certain function that can be fulfilled by a service published by a provider on the Internet. From a business point of view, this agent represents the business that needs certain services to be fulfilled. From an architectural point of view, this is the application that looks for service that meets its requirement. A consumer agent can be a human user accessing the service through a mobile phone or desktop; it can be a cloud-based application, or it can be another web services. The service consumer finds the required services by searching a service registry. Once the service consumer finds the appropriate service, then he can use the URI to bind to services hosted by a service provider.

- *The service broker* is the agent responsible for providing a searchable registry of service descriptions where service providers publish their services and service consumers find services and obtain their binding information.

Several advances in the field of web services and the introduction of multiple web service XML-based standards have facilitated the implementation of the web service model. Among these advances is the introduction of XML-based languages such as Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP) [40], the Universal Description, Discovery, and Integration (UDDI) [41] and the Business Process Execution Language for Web Service (BPEL4WS) [42].

The WSDL supports web services discovery activity by specifying properties of a web service, such as what it does, where it is located and how it is invoked. The SOAP facilitates the exchange of data with the services by specifying unified standard for sending data as part of messages and invoking remote procedure calls over the Internet [40], while the UDDI facilitates the activity related to publishing information about services in the registry. The BPEL4WS web service standard enables the integration and binding of service compositions [42].

In addition to these four core standards, there are some complimentary standards that provide additional support such as WS-Security, WS-Trust, WS-coordination and WS-policy [43]. All of these standards represent the minimum structure required for implementing the web service model and they are referred to as web services technology stack [43].

Currently, web services topologies are adopted by many organisations to make their traditional software applications available to the public as web services for different needs. These web services can be implemented based on various software modules [44]. For instance, a web service can be implemented as self-contained service, such as a money withdraw service or deposit service; or it can be implemented as a stand-alone application such as life insurance application or weather forecast application ; or as a resource enabler service that provides access to resources such as data storage, virtual machine or hardware platform.

The main benefits of implementing software as a web service in comparison to traditional services are as follows [40]:

- **Easy and fast deployment.** Developers can reduce the time and effort needed for developing a complex system by reusing and orchestrating some low-level web services.

- **Interoperability.** By using the standard interface definition language and protocol, any web service can interact and collaborate with any other web service which means that web services are truly language and platform independent. The interoperability of web services will enable the application developer to integrate their services with services implemented using different languages and allowing them to communicate with legacy applications.

- **Just-in-time integration.** Traditional software systems tend to be sensitive to change as a change in the implementation or output of a subsystem will often cause the static coupling of the subsystems to break down. Web service based systems promote the just-in-time integration of new service and applications.

- **Reduced complexity by encapsulation.** In the web service model, what is important is the behaviour that services provide not how they are implemented. This reduces the complexity of the implementation, as service consumers are concern with what the services do rather than how they are implemented.

2.1.2 Web Service Composition

One of the distinctive features of web service is the ability to integrate a number of web services to create an added value composition of services that satisfies the user requirements. The process of selecting and allocating a composition of service is called *Web Service Composition* [44]. For example, a website that provides holiday packages can be built by aggregating a flight booking services, car booking services, hotel booking services and card payment services.

Several methods have been proposed to facilitate and automate the task of web services compositions. These composition methods are categorised by Milanovic et al. [45] to six categories as follows: Semantic Web OWL-S models [46], Web Components approaches [47], Algebraic Process Composition [48], Model Checking and Finite-State Machines, Petri Nets approaches [49] and workflow based approaches [50]. The most popular is the workflow-based approach, and it has been used as a standard approach to implement services composition in business and scientific communities.

In workflow based composition, a workflow is used to describe the patterns of executing a collection of web services. The workflow helps to do the following:

1. Specify how the web services are combined to achieve the required functionality of the composition.

2. Define the composition control flow that specify the order for executing the web services and the control point where some of the activities may or may not be performed.

3. Define the composition data flow that specifies the exchange of data between the different services of the composition.

Many web service languages have been proposed to facilitate the integration of workflow based web services composition, such as the Web Services Business Process Execution Language (WS-BPEL) [51], Simple Conceptual Unified Flow Language (SCUFL) [52] and the XML Process Definition Language (XPDL) [53]. Using these workflow languages with other web services standards presented in the web services technology stack will enable a web application to automate the tasks of discovering, selecting and integrating web services to satisfy the user functional requirement [43].

2.2 QoS-Aware Service Composition

Given the potential existence of multiple web services that offer similar functionality in the cloud market. An application requirement can be fulfilled by one or more composite services, which offer similar functionalities but come with different QoS. In this context,the goal of QoS-aware service composition is to select and allocate a web service composition that: 1) achieves the functional requirements of the application, 2) satisfies the QoS constraint imposed on the composition, and 3) maximizes the overall QoS of the composition.

In this section, we will present an overview of the state of the art of QoS-aware web services composition. This overview structures the state of the art of QoS service composition into five areas: 1) QoS modelling and description, 2) the scope of QoS Constraints(global vs local constraint), 3) problem modelling, 4) selection strategy and technique for selecting the composition and 5) supporting an adaptive QoS-aware services composition.

2.2.1 QoS Modelling and Description

Our analysis of QoS-aware web services composition will start with QoS modelling and description which is an enabler for the selection process. There is an absence of unified definitions and standard models for QoS. With the absence of unified model, QoS-aware web services composition methods have used various models for specifying the QoS of the composition.

Generally, these QoS models can be classified based on the covered QoS into two main categories: Specific and Generic models. Specific QoS models tend to define a limited number of the commonly used QoS properties, such as cost, performance, security and reliability. The ASOB framework [54], HireSome-II model [55], BNQM [56] and the QoS model presented in [57], are examples of a Specific QoS models.

On the other hand, Generic QoS models are more comprehensive in the sense that they tend to define a larger number of QoS properties. AMIGO [58] and the QoS model proposed by Rosenberg [59] are examples of a generic QoS models. In AMIGO [58], the QoS model categorized the QoS properties into five distinctive categories: reliability, cost, transaction, security, and performance, where each one of these categories contains one or more QoS.

Regarding the cloud environment, Generic QoS models are more appropriate choices to address QoS-aware services selection and composition as they cover a larger number of properties that may be required by both the service providers and consumers.

Another way of classifying the QoS models is based on the scope of the QoS model. On that base, the QoS models can be divided into *Services focused QoS models* and *End-to-End QoS models*. In Services focused QoS models, the scope of the QoS model is limited as it focuses on QoS properties that affect the application only such as availability, price and response time. A representative example of the services focused QoS models is the framework presented by Christos et al. [60].

On the other hand, End-to-End QoS models have a wider scope as they cover all the factors that influence the QoS delivery to the user. These factors include QoS properties that affect the application, network and the provider infrastructure, which host the application. A typical example of End-to-End QoS Models are the framework proposed by Yang et al [61]. In his framework, Yang modelled both service QoS properties, such as performance, availability, cost as well as network QoS properties such as devices availability and reliability. Another interesting End-to-End QoS model was presented by Chang et al [62]. In this model, QoS were divided into three categories: 1) QoS for service; 2) QoS of content delivery such as correctness of delivered information; and 3) QoS of the hardware that cover aspects such as processing power, memory and power consumption.

In a cloud environment, both of the network used to connect end users to services and the hardware used to host the application services may considerably affect QoS of composition services. This is because a busy network can affect the response time of services and heavily shared hardware resource can reduce the service throughput. Therefore, considering QoS on an end-to-end basis is required when dealing with web service selection and composition in a cloud environment.

Finally, QoS models can be further divided based on the level of the specification they provide to BlackBox and WhiteBox models. In BlackBox QoS models, values of QoS are associated with services which represent a black box entity without having any prior knowledge about the operations or the structure of these services (i.e. framework presented by Christos et al [60]). In WhiteBox QoS models, QoS values are associated with more refined elements of services. Specifically, QoS values are associated with operations and tasks that represent the functional behaviour of the services. An example of WhiteBox QoS models is the PERSE framework [63] where services are represented as a set of operations linked by different types of control structures, such as loop, parallel and sequences and the QoS values are associated with these operations.

Another example of WhiteBox model is the work by Rosenberg [59] in which he defined a three layer QoS model for service composition. The first layer focuses on QoS proprieties of individual services in the composition. The second layer focuses on the peer-to-peer QoS between the services within the composition such as Services level agreement. The third layer provides global view of QoS for the service composition where the QoS of all the services in the composition is aggregated to calculate the global QoS for the composition.

Regarding the cloud environment, adopting a WhiteBox approach can enrich the QoS model. However, the WhiteBox modelling is considered to be a complex approach as it requires additional specification of services structure and behaviour. This is not possible in some cases as some service providers are not willing to share these information. Therefore, considering BlackBox model is deemed to be a suitable choice for web service selection and composition in a cloud environment.

2.2.2 The Scope of QoS Constraints

One of the goals of QoS-aware web services composition is to select a set of candidate services that satisfy a set of QoS constraints. There are two types of QoS constraints used in the literature of web services composition: Global and Local QoS constraints. The Global QoS constraints are constraints that are imposed in the whole web services composition, whereas the local QoS constraints have a limited scope that covers only individual services in the composition.

The type of QoS constraints have a significant impact on the level of complexity of the QoS web services composition problem. While QoS web services composition under local constraints creates a problem with linear complexity [61], QoS-aware web services composition under global constraints is a NP-hard problem [64].

Most of the algorithms proposed in the literature of QoS-aware service and composition(e.g., HireSome-II [55], PERSE [63] , SanGA [65], Clobmas [66]) model the problem using global QoS constraints which is challenging task when compared to selection under local constraint. For that reason, this thesis will consider the selection of services under global constraints.

2.2.3 QoS-Aware Web Service Composition Models

The modelling of QoS-aware web services composition aims to enable a formal specification of the problem which is the first step on that path of finding an appropriate composition. In the literature of QoS-aware web services composition, three models have been used to specify the problem: *Mixed Integer Linear Program (MILP)*(i.e. Ardagna et al [67] and Alrifai et al [68]), *Multi-dimension Multi-choice Knapsack Problem (MMKP)* (i.e. Jaeger et al [69] and Yu et al [70]) and *Multi-Constraint Optimal Path* (MCOP) (i.e. Yu et al [70]).

- *MultiConstraint Optimal Path (MCOP).* In this model, web services composition is presented as a directed graph of nodes and links. The goal of MCOP is finding an optimal path that starts from the root node to the end node that maximise the attributes while maintaining multiple constraints, such as limited cost. QoS-aware web services composition problem can be modelled as MCOP by creating a graph where service candidates are formulated as nodes and the workflow between them as links. The goal of QoS-aware web services composition is finding an optimal path of service candidates that have highest QoS values while maintaining the QoS constraints.

- *Multi-dimension Multi-choice Knapsack Problem (MMKP).* This model assumes that there is a set of items, and there are a number of alternative resources. Each item will require a certain amount of each resource and it generates a value that depends on the selected resources. The goal of MMKP is to select an optimal subset of items to put into a knapsack with limited resource capacity that maximizes the sum of the values of the included items, while the size of all selected resources is less than or equal to the knapsack capacity. QoS-aware web services composition problem can be formulated as MMKP by mapping composition to knapsack where the QoS constraints represent knapsack capacity; candidate services represent items and QoS represents the values of the items.

- *Mixed Integer Linear Program (MILP).* The goal of MILP is to select an optimum solution that maximizes or minimizes an objective function that aggregates a number of variables. At the same time, the solution needs to comply with a set of constraints represented by linear equations. QoS-aware web services composition problem can be presented as a mixed integer linear program by defining the aggregated QoS of the service composition as the objective function and QoS constraints as the linear constraints in the model.

Due to the poor scalability of the Mixed Integer Linear Program methods [71], MILP becomes an unsuitable choice for modelling QoS-aware services composition in large scale and dynamic environment such as the cloud. Both Multi-Constraint Optimal Path and

Multi-dimension Multi-choice Knapsack Problem represent a practical method for modelling QoS-aware composition in large scale environment [70]. However, due to the additional effort required for modelling the service composition as a directed graph in MCOP. This thesis will use an MMKP to model a QoS-aware composition in the cloud.

2.2.4 Selection Strategy to Allocate the Composition

QoS-aware service composition approaches can adopt different strategies for exploring the search space. As all algorithms are aiming to find a combination of items, QoS- Aware service composition algorithms can be categorized into two general strategies: *brute force algorithms* and *heuristic algorithms.*

The aim of *brute force algorithms* is to find an optimal services composition by exploring all the candidate services in the registry to ensure optimality. The service composition algorithms presented by Mokhtar et al. [58], Yu et al. [70] and Zeng et al. [72] are categorized as *brute force algorithms* as they ensure optimality by considering all possible compositions. However, that optimality comes with a high computational cost.

To overcome the high computational cost, some composition solution adopt lightweight *heuristic algorithms* that seek near-optimal composition (e.g [73], [74] and [65]). These *heuristic algorithms* do not perform an exhaustive search that explores all the possible compositions; However, they seek near optimal solution by using different heuristics. The main goal of using these heuristic is to provide a systematic way to explore a subset of the search space that is more likely to lead to finding a satisfying solution. Thus, enabling to reduce the computational time needed for running these algorithms.

In the literature of web service composition, different heuristics have been adopted by the selection algorithms. These heuristic-based selection algorithms can be categorized into *Greedy* and *Discarding subsets* algorithms [69]. In the *Greedy* algorithms, for each abstract service in the workflow, one candidate service that has the highest QoS score is selected and the rest of candidates are ignored.

Elhabbash et al. [75] presented a greedy algorithm for web service composition in a volunteer environment. The authors used the value QoS utility for each candidate as a heuristic to reduce the search space and provide a near optimal solution in a timely manner. Another *Greedy* algorithm for web service composition is presented by Zeng et al. [72]. In their local algorithm, they employed *Simple Additive Weighting* (SAW) technique as heuristic to identify the optimal candidate in terms of QoS for each abstract service. Another interesting work is the *Greedy* algorithm introduced by Yang et al. [61]. The authors use a QoS based aggregation function to identify the optimal candidate for each abstract service in the workflow. The main limitation of adopting a Greedy algorithm, is its inability to guarantee the globe QoS constraint [69].

To cope with this limitation, other set of *heuristic-based* algorithms that maintain global constraints have been proposed, such as *Discarding subsets* algorithms [69]. In *Discarding subsets* algorithms, selection is performed over several phases. In each phase, a subset of candidates are nominated to move on to the next phase and the rest are discarded. In the *Discarding subsets* algorithms adopted, Alrifai et al. [76] uses QoS based clustering of the candidate services as heuristic to nominate the dominating services that are more like to lead to a near optimal solution. Similar work was presented by Mabrouk et al. [77]. The authors use k-means clustering of the candidate services to nominate the services that are going to move to the next phase.

Other solutions rely on bio-inspired algorithms, such as *Particle Swarm Optimization* (PSO), *Bee Colony Optimisation* (BCO) and *Ant Colony Optimization* (ACO) to select a suitable subset of candidates services. The bio-inspired algorithm developed by Wu et al. [73] applied the principles of ant colony optimization to find a near optimal composition. The latter modelled the problem of web services composition as a constrained directed acyclic graph with a start point and a target point. The author used the starting point of the problem as a nest of the ants and the target point as the food source. A QoS based phenomenon are employed as heuristic to guide the ant in selecting the most attractive candidates for each abstract services in the graph.

The selection algorithm presented by Lartigaua et al. [78] use *Artificial Bee Colony*

(ABC) optimisation where QoS and physical location are used as heuristic to select optimal candidate services for each abstract service in the composition. An example selection algorithm that relies on *Particle Swarm Optimization* (PSO) is depicted by Wang et al. [79].

Overall, the adopted selection strategy and technique varies from one composition algorithm to another. The choice of the selection strategy and technique depends on the type of constraints imposed in the selection (i.e. Global vs Local constraint) and the user preference in terms of receiving spontaneous result or optimal solutions.

2.2.5 QoS-Driven Service Composition Adaptation

Self-adaptation is a key requirement that needs to be considered when developing QoS-aware service composition approaches. It enables the QoS-aware service composition to react to the changes in the dynamic cloud environment and maintain a satisfying solution. Several factors can trigger the need for adaptation of web services composition. Bucchiarone et al. [80] categorised those factors to changes in: the functionality of the services, QoS of the services, the business context, the computational context, and user preferences. However, the focus of this section will concentrate on adaptation trigger by QoS changes which is known as QoS-driven services composition adaptation.

The goal of self-adaptive QoS-aware services composition is to adaptively change the service composition in order to maintain QoS constraint and/or optimise the global QoS of the composition [66]. To achieve self-adaptation, several solutions have been proposed for QoS-aware web services composition algorithms. Several surveys have covered software adaptations in generals [81], [82] and [80]. The authors presented detailed taxonomy and criteria for comparing and classifying adaptive software solutions in general. In the light of these surveys, we will adopt the following criteria to assess and classify the proposed QoS-driven composition adaptation solutions: *Adaptation model* and *Adaptation approach* [81] and [82].

Adaptation Models are concerned with how adaptation problem is formulated. We were able to identify three models: 1) *mathematical based* 2) *graph based* and 3) *policy based models*. Zeng et al. [72] relied on a *mathematical-based model* to self-adaptively manage QoS composition in a dynamic environment. The self-adaptive algorithm reacts to the changes of QoS that occur during the execution of a composite service, by revising the selected

Yan et al. [83] employed a *graph model* to enable adaptation. This model symbolizes the services as a tuple*(in, out)* where *in* is the service input data and *out* is the service output. For instance, if an input of a service matches the output of another service, thus these two services can be merged into a composition. When one of the services of the composition did not meet the requirements, a greedy search process will be activated to find an alternative composition on the graph model that satisfies the users needs. MASC middleware [84] is using a policy based self-adaptive mechanism. The model relies on *Event-Condition-Action* (ECA) rules to detect and trigger adaptation that maintains a satisfying composition.

The main advantage of policy based models in comparison with other models is that policies are represented with higher-level abstractions. As a result, the software developer can easily specify them. For that reason, a *Policy-Based Model* will be used to model self-adaptively our QoS-aware web services composition.

The *Adaptation Approach* deals with how and when the adaptation decision can be planned and constructed. Related to this, we categorise *Adaptation Approach* into: *(1) Static,and (2) Dynamic Adaptation Approach.* In *Static Adaptation approach* (e.g.work of Mokhtar [63]), future needs of adaptation are anticipated, and the adaptation plan is hard-coded in advance, detailing the required changes of the service compositions. These hard-coded plans are activated when a need for adaptation is triggered. Static adaptation approaches allow a fast reaction to change in the environment. However, these static plans do not consider the recent changes in the environment at the time of adaptation (e.g. changes in QoS or the addition of new services to the cloud market after adoption plan is hard-coded).

In *dynamic approach* (e.g. work Nallur [66], Elhabbash [75]) the adaptation planning is taking place dynamically at run time just after adaptation is triggered. For that reason, dynamic approach considers the current state of the environment where QoS-aware service compositions operate. However, the delay in dynamic adaptation may be significant when compared to static approach as adaptation plans are created at runtime. Despite their computational cost, we consider using a dynamic approach as a suitable choice for implementing the self-adapting mechanism in the high dynamic cloud environment as they reflect the current state of the market.

2.3 Discussion and Summary

In this chapter, we have presented a survey that covers a number of existing methods for QoS-aware service composition in both traditional software environment and cloud environment. Based on the review, we can argue that QoS-aware service composition addressing cloud environment are closely related in general to QoS-aware service composition in traditional software environments. As shown in our survey, QoS-aware service composition in both environments use the same taxonomy, models and adopt similar strategies to solve the services selection problem. However, we deem that these composition solution are not sufficient to cope with QoS awareness challenges in cloud environments.

Although the fundamentals of selecting services in the cloud and the traditional environments could appear to be similar, but there are some differences: Cloud-based markets tend to be dynamic and volatile, in situations where cloud providers continuously update their provision of services, QoS, and price. In such model, competition is respected, in situations where cloud providers continuously compete for providing better services, QoS, and price.

For this reason, research efforts should be devoted to designing a novel QoS-aware service composition method that takes into account the challenges presented by the cloud environment. To the best of our knowledge, we are not aware of algorithms that dealt with major challenges imposed by the cloud environments on stability-aware service composition that aim to reduce performance fluctuation. During the review, we identified the major challenges imposed by cloud environment on stability-aware service composition. These challenges can be classify into five main areas: QoS modelling and description, supporting an adaptive QoS-aware services composition, problem modelling, selection strategy and the scope of QoS constraints.

- *QoS modelling and description:* In cloud environment, using End-to-End generic QoS models are more appropriate choices to address QoS-aware services selection and composition as they cover a larger number of properties that may be required by both the service providers and consumers. Moreover, most of the existing service composition algorithms focus on finding an optimal composition of services among a set of candidate services based on promised performance published by services providers. Conversely, in a dynamic environment the actual performance delivered by services may fluctuate because of the changes that may occur in the cloud environment (e.g. limited resources, peak in demand). To cope with this issue, service composition algorithm should additionally:

  1. Evaluate performance stability for each service. This requires a continuous monitoring of the performance of all the candidate services, which is hard to achieve considering the large number of services that exist in the cloud market.
  2. The service composition algorithm should consider reducing performance fluctuation of the selected services.

  We appeal to the concept of design diversity as a solution to address the problem of improving the performance stability of services composition. In particular, we see that diversifying the selection of services by using different providers to avoid a single point of failure is beneficial. The concept of design diversity and a brief review of the state of the art of diverse software systems are discussed in chapter 3.

- *Support for self- adaptivity*: Cloud-based market are highly dynamic where new services may be added to the market at any time, the QoS of existing service may change, or the service may become unavailable at any time. Therefore, the QoS-aware service composition should dynamically adapt to changes in the market. One approach towards self-adaptivity consists of replacing services that deliver unsatisfactory QoS with alternative services that deliver better QoS. We think that using and adaptive policy based solution is more appropriate for cloud environment. The main advantage of policy based models in comparison with other models is that policies are represented with higher-level abstractions. As a result, the software developer can easily specify them.

  Compared to existing self-adaptive solutions (e.g. Nallur [66] and Zeng et al. [72]) where service forming the composition are replaced in alternative service resulting with better QoS, this thesis goes beyond the state of the art of self-adaptive solution :

  1. Considering performance fluctuation as a major driver for adaptation.
  2. Considering a cost-efficient adaptation as our adaptive algorithm makes an explicit trade-off between the cost of replacing the services and benefits gain by the changing them.

- *Problem modelling:* In the literature of QoS-aware web services composition, three models have been used to specify the problem: *Mixed Integer Linear Program (MILP)*, *Multi-dimension Multi-choice Knapsack Problem (MMKP)* and *Multi-Constraint Optimal Path* (MCOP). Due to the poor scalability of MILP methods it becomes an unsuitable choice for modelling QoS-aware services composition in large scale and dynamic environment such as the cloud. Both MCOP and MMKP methods present a practical mean for modelling QoS-aware composition in large scale environment. However, due to the additional effort required for modelling the service composition as a directed graph in MCOP. This thesis will use an MMKP to model a QoS-aware composition in the cloud.

- *Selection strategy and the scope of QoS constraints:* The selection strategy used in the literature QoS- Aware service composition can be categorized into two general strategies: brute force algorithm and heuristic algorithms. However, in this thesis we are seeking optimal selection and because of that we will use a brute force algorithm. Regarding the scope of the QoS constraints, most of the algorithms proposed in the literature of QoS-aware service and composition model the problem using global QoS constraints which is challenging task when compared to selection under local constraint. For that reason, this thesis will consider the selection of services under global constraints.

  Combining a brute force algorithm with a global QoS constraints will make optimising for QoS a NP-hard problem. An important issue is how the QoS-aware services composition approaches scale with regard to different scaling dimensions (Number of QoS, Number of candidate, and Number of users). One would expect that problem of scalability in dynamic service composition to be fully addressed in a consistent way in the literature. However, the proposed solutions in the literature have used a limited and inconsistent range of scaling dimensions, which make it very difficult to evaluate the claim of scalability. To overcome this problem, we will use Scalability Goal-Obstacle analysis to identify the scaling dimensions that are relevant to the scalability of our QoS-aware web services composition. The details of this scalability analysis and it findings are presented in Chapter 6.

CHAPTER 3

DESIGN DIVERSITY: *BACKGROUND ON SOFTWARE DIVERSITY*

In the previous chapter, we presented a survey that covered the state of the art of the QoS-aware service composition. We also discussed performance fluctuation as one challenge imposed by cloud computing environment. In addition, we motivated the need for a stability-aware service composition method that appeals to the concept of design diversity to address the problem of performance fluctuation.
The aim of this chapter is to present a brief overview of the concept of design diversity. We present a review that explores how design diversity is adopted in different solutions to improve the dependability and reliability of software systems. We look at different methods used to implement design diversity, as well as review the requirements, challenges, benefits and trade-offs of implementing design diversity.

3.1 Overview of Design Diversity Concept

In nature, the coexistence of many species is often referred to as diversity. While in society diversity often refers to gathering a group of people from different backgrounds and cultures. In both domains, diversity is considered as a source for stability and resilient [85]. In software we take another perspective, we want to achieve properties such as stability and resilient by trying to engineer software diversity [86]. The question here is how to engineer and implement such diversity.

Design diversity can be implemented by creating two or more independent versions of the same service, where all of the independent versions tend to meet the specification. However, each independent version has its unique design decisions and is implemented in a distinctive way [87]. In this case, if a fault occurs in one of the versions, there is a great chance that the other versions will continue to be intact. Different research groups have studied design diversity. Prominent examples are the Centre for Software Reliability at City University and Dependable-Computing and Fault Tolerance Laboratory at University of California Los Anglos (UCLA).

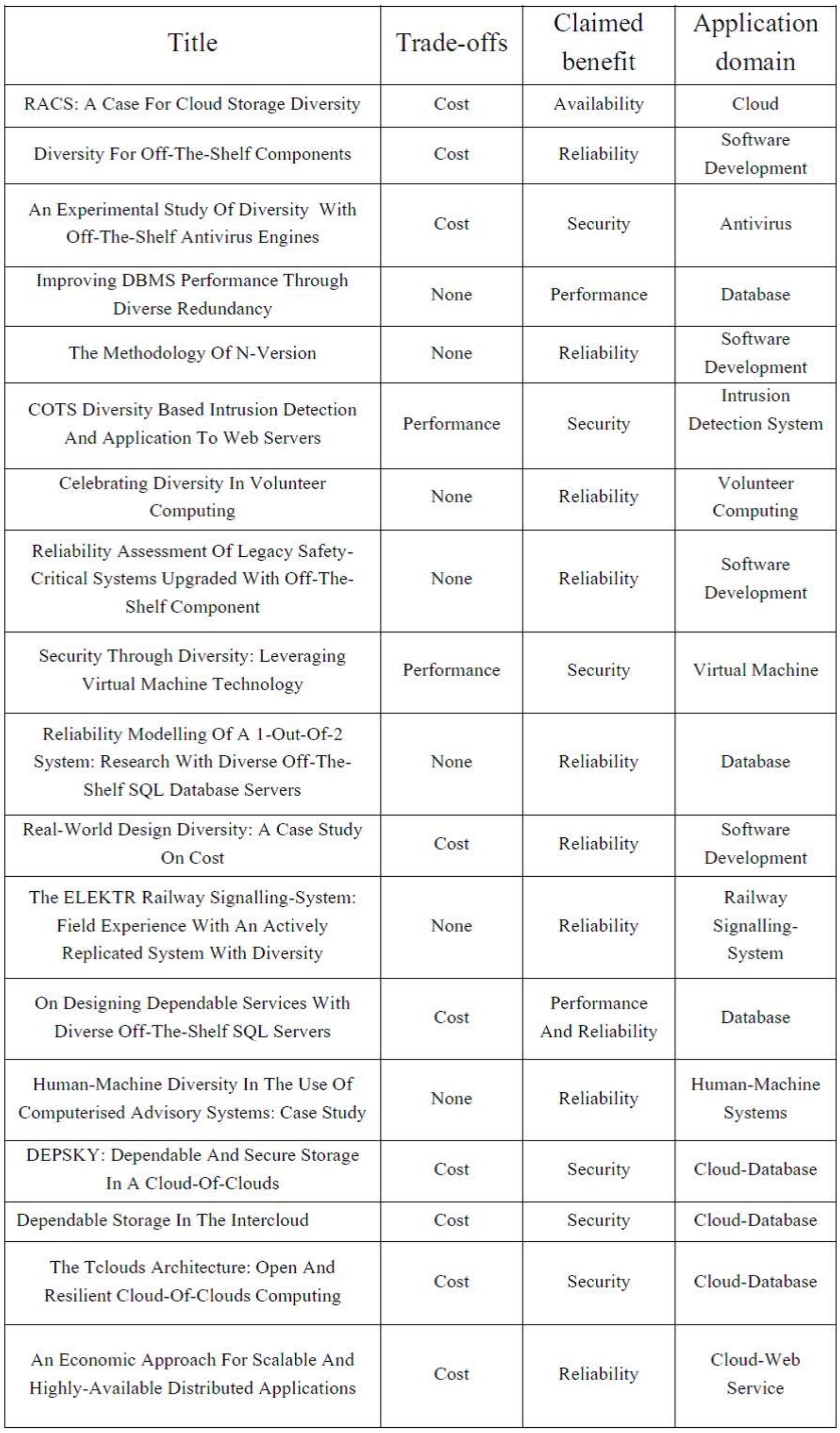
Diversity in design is a mature topic [88] and it has been used as a strategy to increase the reliability and dependability of software systems, such as in [89] and [90]. During the nineties, techniques adopted for implementing design diversity were widely criticized for their high cost as each independent version of the software has to be developed from scratch, which can double the implementation cost [91]. As a result, the applications of these techniques were limited to the critical systems (e.g. airplane [92], trains [93]) where failure can lead to financial disasters or losses in human lives. However, in the context of services hosted on cloud market, there are many services that provide the same functionality using different implementations and hardware, thus making diversity-based techniques more practical and cost efficient [94].

3.2 Questions

Researchers and engineers have developed a huge body of work on designing and implementing diverse systems. However, it is not clear how the research results have contributed to improvements of designing diversity in a cloud-based application. Meanwhile, there have been several research efforts on dynamic QoS- aware selection and composition (discussed in Chapter 2). However, these solutions do not implement the concept of design diversity.

In this review, we will study and summarize some of the existing research efforts related to designing diverse systems and shed light on the claimed benefits, trade-off and limitations of applying design diversity. In particular, we aim to answer the following questions: what are the different techniques and methods that have been used to implement design diversity?, what are the claimed benefits and trade-offs of applying design diversity?, what are the reported concerns and limitations that need to be considered when implementing design diversity solutions? and what are the requirements and challenges of implementing diverse systems using inspiration from nature?. An overview of papers included in this review are presented in table 3.1.

Table 3.1: An Overview Of the Covered Design Diversity Papers



Title	Trade-offs	Claimed benefit	Application domain
RACS: A Case For Cloud Storage Diversity	Cost	Availability	Cloud
Diversity For Off-The-Shelf Components	Cost	Reliability	Software Development
An Experimental Study Of Diversity With Off-The-Shelf Antivirus Engines	Cost	Security	Antivirus
Improving DBMS Performance Through Diverse Redundancy	None	Performance	Database
The Methodology Of N-Version	None	Reliability	Software Development
COTS Diversity Based Intrusion Detection And Application To Web Servers	Performance	Security	Intrusion Detection System
Celebrating Diversity In Volunteer Computing	None	Reliability	Volunteer Computing
Reliability Assessment Of Legacy Safety-Critical Systems Upgraded With Off-The-Shelf Component	None	Reliability	Software Development
Security Through Diversity: Leveraging Virtual Machine Technology	Performance	Security	Virtual Machine
Reliability Modelling Of A 1-Out-Of-2 System: Research With Diverse Off-The-Shelf SQL Database Servers	None	Reliability	Database
Real-World Design Diversity: A Case Study On Cost	Cost	Reliability	Software Development
The ELEKTR Railway Signalling-System: Field Experience With An Actively Replicated System With Diversity	None	Reliability	Railway Signalling-System
On Designing Dependable Services With Diverse Off-The-Shelf SQL Servers	Cost	Performance And Reliability	Database
Human-Machine Diversity In The Use Of Computerised Advisory Systems: Case Study	None	Reliability	Human-Machine Systems
DEPSKY: Dependable And Secure Storage In A Cloud-Of-Clouds	Cost	Security	Cloud-Database
Dependable Storage In The Intercloud	Cost	Security	Cloud-Database
The Tclouds Architecture: Open And Resilient Cloud-Of-Clouds Computing	Cost	Security	Cloud-Database
An Economic Approach For Scalable And Highly-Available Distributed Applications	Cost	Reliability	Cloud-Web Service

3.3 Techniques and Methods for Implementing Software Diversity

In this section, we will present an overview of the different techniques for implementing software diversity. Software diversity techniques can be divided into two main categories [86]: *Created Diversity* and *Managed Natural Diversity*. In created diversity: we look at approaches that try to implement diversity by creation. This, for example, can be through the creation of duplicated versions of the same system. On the other hand, in natural diversity the environment provides ready means for diversification, where research effort is heavily concerned with how to exploit the primitives for diversification and consequently manage them.

3.3.1 Created Diversity

In *Created Diversity*, software diversity is engineered at design phase to cope with probable accidental faults. There are three main techniques to implement Created diversity:

1. *N-version programming:* First introduced by Avizienis et al. [87] in 1977. In this design diversity technique, three or more versions of a program are independently developed. Those programs share the same functionality. Then all of the independent versions are executed in parallel. A majority voting logic is used to compare the results produced by all of the versions and to report one of the results that is presumed correct. This technique has been applied to improve fault tolerance in number of domains, such as software development [95] and system security [96].

2. *Recovery block:* This technique was first introduced by Horning et al [97], where only one active version of the program is executed at a time. When a failure of the active version is detected by an acceptance test, the program will recover the failure via roll-back and retry to run program using a different version. The main differences in the recovery block technique from N-version programming are the number of active versions running at the same time and the use of acceptance test

instead of relying on majority voting to get the correct result. The recovery block technique has been applied to improve a software fault tolerance in the cloud [98].

3. *N-self checking programming:* First introduced by Laprie et al. [99]. A self-checking component is a version of the program with an acceptance test or a pair of versions of the program with an associated comparison test to detect mismatched results. Both of the versions and their acceptance tests are developed independently from common requirements. Fault tolerance is achieved by executing more than one self-checking component in parallel. N-self checking technique has been used in [100] to improve reliability in volunteer computing system.

3.3.2 Managed Natural Diversity

By natural diversity, we refer to the existence of different services that share the same functionality. In this case, diversity can be easily achieved by selecting a set of these out of the off-the-shelf software. Natural software diversity exists in several computing environments such as the grid and cloud computing. The cloud computing comes with built-in primitives and underlying resources, which can help with designing of diverse systems. These may be part of one or more layers within the cloud ecosystems. For example, in Software as a Service (SaaS), it may be possible to design for diversity through selecting multiple instances from multiple providers. The idea is that by diversifying the allocation, we can improve dependability and reduce the probable risks.

This is due to the fact that risks tend to vary with cloud providers. The dependability and reliability of the solutions tend to be sensitive to the underlying resources used as well as their locations. This is because cloud providers may vary in terms of the software, hardware, operating systems, virtualization mechanisms and physical locations. Assuming that the service provision tends to be functionally identical across candidate providers, the non-functional characteristics tend to be sensitive to the underlying resources deployed to support the running of these instances. Henceforth, the strategy for considering more than one provider leans towards diversity.

Section 3.6 provides coverage of indicative existing work on diversity in environments, such as the cloud, the grid and volunteer computing. In such naturally diverse environments, three general properties can be exploited to enhance the level of diversity among the selected services: Geographical diversity, Ecological diversity and Modal diversity [101]. Table 3.2 presents a brief description of the diversity properties as well as a simple example of how diversity is implemented.

Table 3.2: A Brief Description of the Diversity Properties in Natural Diversity

Diversity properties	Description	example
Geographical Diversity	The extent to which physical elements hosting the application are distributed in different geographical location.	This can be achieved by selecting services from providers in different locations. This geographical distribution will make the application less sensitive to the incidents that can affect local location such as network failure or power outage.
Ecological Diversity	the extent to which the application is hosted in a heterogeneous system environment	This can be achieved by selecting services that are hosted on machines that use the different operating system. As heterogeneous systems are less vulnerable to common-mode failures caused by bugs, which often tend affect a single type of operating systems.
Modal Diversity	The extent to which any application functions is designed to utilize multiple modes of operation.	If a weather application is required to send floods warning to the public, Modal Diversity can be achieved by selecting services adopt different technologies to deliver the warning such: SmS and email. This will make the application less sensitive to problem affecting one of these technologies.

3.4 Claimed Benefits and Trade-Offs of Applying Design Diversity

To identify the claimed benefits and trade-offs of using design diversity, we used data extracted from Table 3.1. Overall, we found that all of the claimed benefits were related to QoS. The most claimed QoS are reliability (60%), security (30%), performance (10%)

and cost saving (5%). On the other hand, we found that the reported trades offs of using design diversity were cost (50%) and performance (10%). Figure 3.1 summarizes the claimed benefits versus trade-offs of using design diversity.

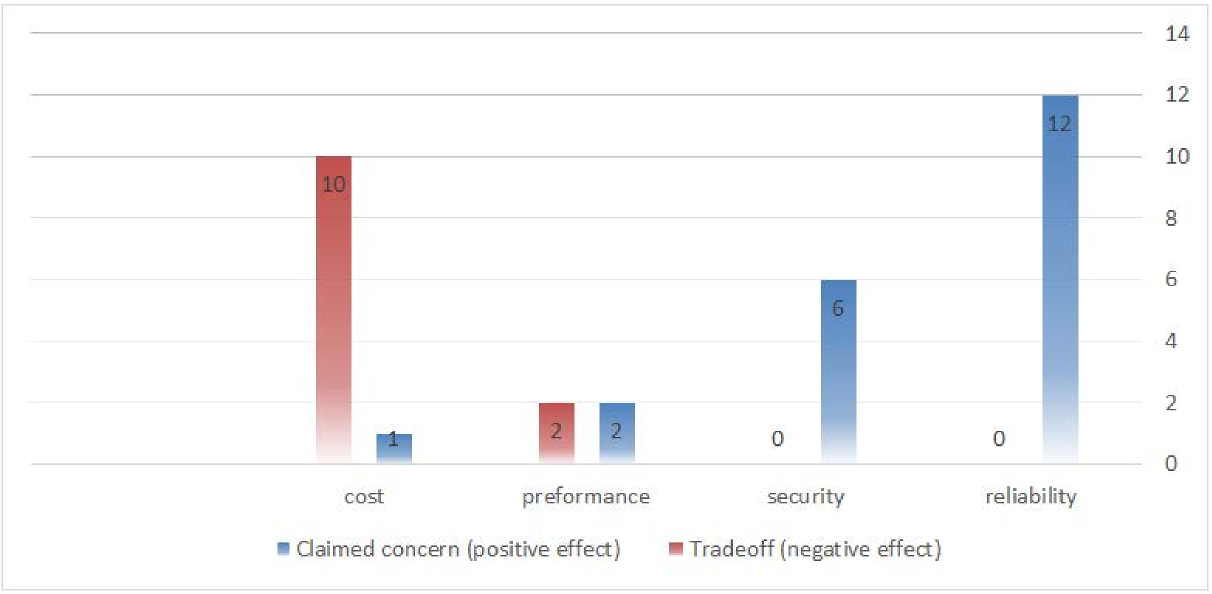


Figure 3.1: Claims Versus Trade-offs of Design Diversity. Bars Show the Total Number of Reported Concerns.

3.5 Concerns and Limitations Needed to be Taken into Account when Implementing Design Diversity Solution

The use of design diversity techniques have been advocated by [87], [95] and [96] as a mean of achieving fault tolerance in software systems. A common key assumption is that by using multiple versions of independently developed software, we will have uncorrelated failures. However, the work of Knight [102] and Eckhardt [103] indicated that software failures may be correlated in the independently developed software systems. Another concern shared by the majority of reviewed papers is the high cost of developing diverse systems. However, implementing a cost-efficient diversity system in the cloud market is feasible due to the low cost of reusing services offered by different providers.

3.6 Requirements and Challenges of Implementing Diverse Systems Using Natural Diversity

Software diversity, naturally emerges in software markets as the services running in heterogeneous systems such as the cloud, grid and volunteer computing. Recently, several researchers have proposed methods to increase the reliability and security by implementing design diversity techniques in order to exploit the natural software diversity.

Anderson and Reed [100] proposed using diversity as fault tolerance technique to improve reliability in grid environment. The paper presents an adaptive approach that consists of two main steps. The first step is the estimation of error rates for each host. Then in the case of high levels of error rate, a random replication will be performed to reduce the incidence of errors.

Abu-Libdeh et al [104] have presented a middleware to stripe user data across multiple cloud providers to reduce the cost of switching between the providers, and to better tolerate provider outages or failures. However, they have not addressed how to evaluate the dependability between different providers before distributing the data among them.

Similarly, Bessan at al [105] have presented DEPSKY, a system that improves the availability, and integrity of data stored in the cloud through the encryption and replication of the data on diverse clouds that form a cloud-of-clouds. The system will perform a full replication of the data which increases the cost of the system. Furthermore, they have not discussed how to evaluate the different providers before distributing the data among them.

Bonvin et al [106] have introduced a cost-efficient approach for dynamic and geographically diverse replication of web service composition in a cloud computing infrastructure that offers service availability guarantees. Their approach is similar to the two steps mentioned in [100]. The first step is to evaluate the availability of the web service and when it goes below a certain level, a replica will be activated. In the second step, the algorithm selects the provider of the new replication based on net benefit-based policy, the geographically closest and least loaded, different replications are allocated to different physical locations.

The solution by Zheng et al. [107] try to improve the fault tolerance of web service by using a user-collaborated QoS model. The solution by Anatoliy et al [108] has proposed a method of improving the dependability of web service composition by updating some of the services from an online market. They used the *probability of failure on demand* as a mean to systematically measure the dependability of both the services and the web services composition. A similar solution for improving the dependability of web service composition has been presented by Mansour et al [109] where they employed hybrid the reliability model to evaluate the dependability of the services in the composition.

3.7 Recommendations

Based on the reviewed diverse systems, we offer the following recommendations for implementing diverse resilient systems in a dynamic environment, such as the cloud with explicit focus on stabilising performance in the service-oriented composition.

1. To have Adaptive component for managing diversity such as in [100] and [106] for accommodating changes in the dynamic environment.

2. To consider the correlation between the performance of the selected services . All of the reviewed diverse solutions [100], [104], [105], [108] , [109] and [106] have failed to evaluate the correlation between the diverse solutions. While [104], [105] have not addressed the issue, [100] allocated the replication randomly. Solutions presented in [108] and [109] have considered the dependability of the services in the composition. However, they have not considered correlation between the services. The work presented by [106] has used net benefit based policy that allocates the new replication at the geographically closest and least loaded resource. This policy has also ignored the correlation of failures between the different locations.

3.8 Summary

In this chapter, we highlighted the concept of design diversity followed by a review of the current design diversity solutions. In this context, we discovered that ignoring the possibility of correlated failure of the selected services can lead to a poor diversification of system. We also recommended that an effective diversification decision should be linked to the correlation between the candidate services of the applications. In the next chapter, we introduce an economics-driven web services selection approach to implement design diversity in scaling up scenario using the *Modern Portfolio Theory*.

CHAPTER 4

PORTFOLIO BASED WEB SERVICES SELECTION: *IMPROVING PERFORMANCE STABILITY IN SCALING UP SCENARIO*

This chapter describes our portfolio-based approach for selecting cloud-based services in a scaling up scenario with the aim of reducing the risk of performance fluctuation of the cloud-based application. First, we will provide a background on the Modern Portfolio Theory that is necessary to understand our approach. Then, we will introduce the logic behind the theory and how effective diversification can be implemented. After that, we present the concept of effective diversification in the context of web services selection followed by the presentation of the assumptions of our model. Then, we formulate the portfolio-based services selection model. We finally present an evaluation to illustrate the applicability of the approach for services selection in scaling up scenario.

In the evaluation, a set of controlled experiments are conducted to (1) test the approach effectiveness in minimizing the risk of performance fluctuation; (2) simulate the dynamic and adaptive behaviour of the approach in responding to changes in the market conditions and risks of performance fluctuation; (3) evaluate the sensitivity of the allocation decisions to risk of performance fluctuation and its correlation with performance of the other candidates and (4) evaluate the scalability of the approach and its ramifications on risk reduction under extreme scenarios.

4.1 Introduction

We pursue an economics-driven web service selection approach that implements design diversity. We adapt a novel model that exploits Modern Portfolio Theory [30] , [33] to improve performance stability of the selected services. The model builds on Harry Markowitz [33], Modern Portfolio Theory for reducing the risk of software performance fluctuation. Portfolio theory has found its way in numerous applications. Among them is the optimisation in the selection of software projects [110], the electricity generation planning [111], and a cost-aware virtual machine management in cloud computing from the provider perspective [112].

We argue that portfolio thinking can be used to implement diversity when selecting web services from cloud-based markets. Unlike the reviewed design diversity solutions that share the assumption of uncorrelated failures, the portfolio-based design diversity is correlation-sensitive; it explicitly links the selection of services to performance fluctuation and accounts for correlation. The following section will present a brief background.

4.2 Modern Portfolio Theory: Brief Background

Central to the modern portfolio theory is the concept of investment risk. Markowitz [30] states that investment risk is a silent feature of investment, which measures the uncertainty of future return. Investment risk can emerge as a result of market, industry or company risks [113]. All these risks lead to uncertainty about future returns. What measures the level of uncertainty of the future return is how fluctuating is the asset return. The higher the fluctuation of asset return, the higher the uncertainty about their expected return. The standard deviation of historical asset return is used to estimate the risk of each asset.

4.2.1 What Problems Do Modern Portfolio Theory Address?

Critics recognise the limitations of qualitative methods to predict and manage the risk of investment. In these qualitative methods, the selection on investment assets is made based on personal experience with company and industry analysis. Mandelbrot et al [114] maintain that it is hard to provider an accurate prediction the effect of events on stock prices by using such personal experience.

Markowitz [30] acknowledged the poor accuracy of the traditional qualitative methods in predicting and evaluating financial risk and suggested the modern portfolio theory as a quantitative method to predict and reduce the investment risk by allocating investments among diversified group of assets. It is a mature theory as several researchers have investigated the theory frameworks e.g. [115], [116] and [117]. The logic of the portfolio theory promotes diversification, which goes by the saying *'Do not put all your eggs in one basket'*, is an effective way to reduce the risk of the portfolio [31].

The Modern portfolio theory has three major advantages. First, it provides an estimation of the investment risks based on quantitative analysis of asset prices. Second, it presents a systematic way to implement an effective diversification of the selected assets in the portfolio. Third, unlike the reviewed classical design diversity approaches, the modern portfolio theory links the diversification of assets to both risk and correlation between different assets.

4.2.2 Modern Portfolio Theory

The foundation of Modern Portfolio Theory [30] was developed by the Nobel Prize winner Markowitz in 1950. The aim of Modern Portfolio Theory is to develop a formal procedure that support the decision-making process of allocating capital to a portfolio of multiple investment assets. The portfolio in this theory is a weighted composition of the assets. The weight represents how much an investor should allocate from the capital to those assets.

The Modern Portfolio Theory helps the investor to decide how much of the available capital (s)he should invest in each of the available assets in order to maximise the expected return and minimise the investment risk of the portfolio. This can be achieved by calculating the expected return and risk for every possible portfolio that can be constructed from the available assets. The expected return and risk will evaluate the efficiency of every portfolio. Several possible portfolios are present in a plot chart that has the vertical axis as the expected return and the horizontal axis as the risk (see Fig. 4.1).

Fig. 4.1 shows that the uppermost point that form the curve presents the efficient frontier. The efficient frontier represents portfolios that achieve the maximum expected return for a certain level of risk. For the risk adverse investor, the optimal choice will be a portfolio with the lowest level of risk on the efficient frontier.

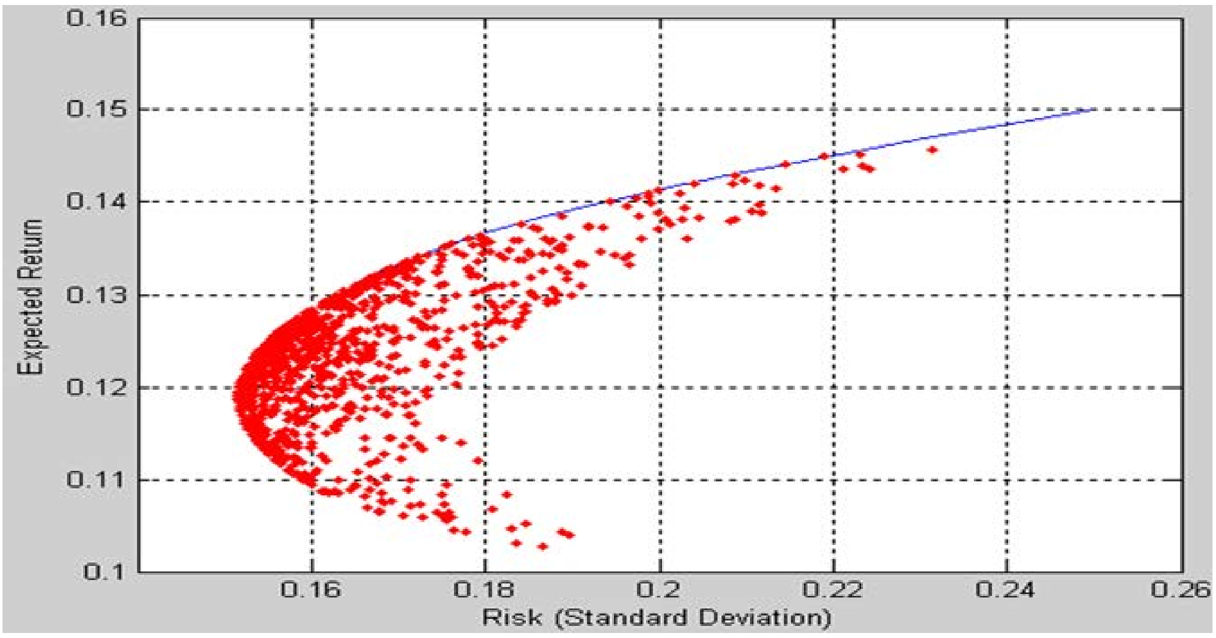


Figure 4.1: Efficient Frontier For the Portfolio of Two Assets. Courtesy Of Mathworks [118].

The expected return of portfolio E_p that of m assets can be calculated as in Equation 4.1 with one constraint, presented in Equation 4.2, where w_i denotes the weight of the capital invested in asset i, and Ea_i represents the expected return on investing in asset i.

$$E_p = \sum_{i=1}^{m} w_i Ea_i \qquad (4.1)$$

$$\sum_{i=1}^{m} w_i = 1 \qquad (4.2)$$

To evaluate the risk of a portfolio of investments, it is necessary to measure the risk of each asset in the portfolio. This can be statically calculated based on the historical return of the asset. Under the modern portfolio theory, the risk of the portfolio R_p is affected by three factors: the weight of the capital invested in each asset w_i, the risk associated with each asset Ra_i, and p_{ij}, which is the correlation between assets. The risk of the portfolio R_p is calculated as in Equation 4.3

$$R_p = \sqrt{\sum_{i=1}^{m} w_i^2 \; Ra_i^2 + \sum_{i=1}^{m}\sum_{j\neq i}^{m} w_i w_j \; Ra_i \; Ra_j \, p_{ij}} \quad (4.3)$$

4.2.3 Effective Diversification

Modern Portfolio Theory utilises the concept of diversification to reduce the risk of investment. Markowitz [30] states that it is possible to reduce the portfolio fluctuation by selecting a set of diversified assets. The key to achieving effective diversification is to understand how assets returns behave when combined in a single portfolio [113].

Markowitz viewed correlation as a tool to understand how combined assets behave. Correlation among returns of different assets represents a silent feature of investment. Similar to other financial quantities, asset returns tend to move up and down in the same or in the opposite direction. For that reason, when forming a portfolio, it is important to take into consideration how different assets returns interact together [113]. The correlation measures the direction and the strength of the relationship between two assets returns.

The correlation can be statistically evaluated based on historical assets return. Equation 4.4 can be used to calculate the correlation between the returns of assets i and j, where Ra_i is the fluctuation risk of the expected return of asset i, Ra_j is the fluctuation risk of the expected return j, p_{ij} represents the correlation between the return of asset i and the return of asset j, Ea_i is the expected return of asset i, Ea_j is the expected return of asset j and $cov(Ea_i\ ,Ea_j)$ represents the covariance between assets i and asset j.

$$p_{ij} = {}^{\mathrm{cov}(Ea_i\ ,Ea_j)}\Big/_{(Ra_i\ \mathrm{x}\ Ra_j)} \quad (4.4)$$

The correlation is represented as a number between +1 and -1, where +1 denotes a strong relationship with a similar direction, -1 represents a strong relationship with an opposite direction and 0 correlations indicates that there is no relation between the two assets.

The strength of the correlation is indicated by a number that varies from 0 to 1, where 1 represents a strong relationship. As the number moves closer to 0, it indicates a weaker relation. The direction of the relationship is denoted by a sign, where (+) indicates that the assets move in the same direction and (-) indicates that the assets move in the opposite direction [30]. Fig. 4.2 shows scatter plots of different types of correlations.

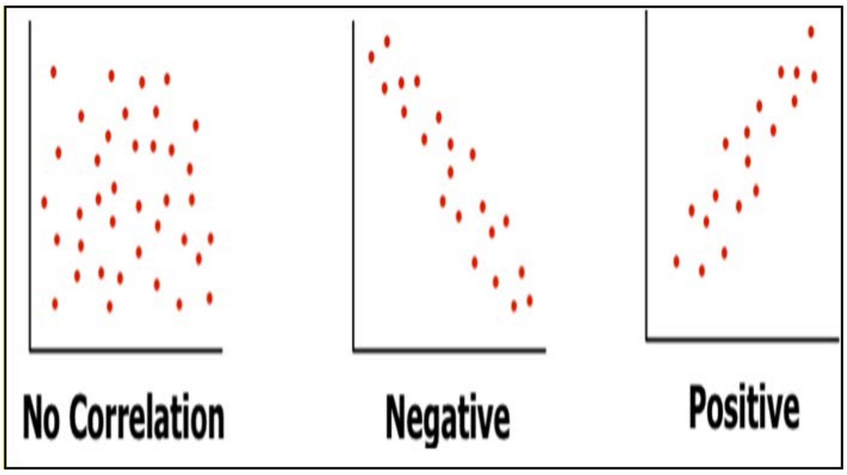


Figure 4.2: Scatter Plot of the Different Types of Correlation [119].

In the case of a strong positive correlation +1, the returns of the two assets move together in the same direction. If the return of first asset increases, the other asset will increase by a similar percentage. If a portfolio is formed of positively correlated assets, the portfolio risk will not be minimised as the assets will follow the same behaviour and these assets will not present any diversification of the portfolio.

In the case of a strong negative correlation -1, the returns of the two assets move in the opposite direction. If the return of the first asset increases, the return of the other asset will decrease by a similar percentage. Consequently, a portfolio that is formed from negatively correlated assets is a well-diversified portfolio. The portfolio risk will decrease as the asset will behave differently [32]. To find a minimum risk portfolio, the investor needs to combine a group of risky assets that have a minimum correlation between them [30].

To sum up, the risk reduction of a portfolio depends on the correlation between the assets. The lower the correlation between the assets (avoiding a positive correlation), the greater the risk reduction achieved by diversification

4.3 Effective Diversification in the Context of Web Service Selection: Analogy and Mapping

A cloud marketplace will facilitate the process of buying and selling instances of web services, which are offered with different prices and QoS. When selecting and subsequently executing web services to perform a specific task, the performance stability plays a detrimental factor to achieve user satisfaction [2]. However, as the cloud is a multi-tenant environment with shared resources and fluctuating demands, it is easy to see how a web service hosted on the cloud can be vulnerable to performance fluctuation.

Similar cases of volatility occur in the financial markets. Stocks in the financial market are vulnerable to price fluctuation caused by a change in supply and demand. Financial investors view this fluctuation as a measurement of investment risk [113]. Modern Portfolio Theory is used in the financial markets with the aim to reduce investment risk by forming a diversified portfolio that has low fluctuation.

The main actors in the cloud-based market are agents acting on behalf of the buyer (cloud-based application) and sellers (cloud service providers):

1. **Seller:** A cloud provider is offering SaaS S_i in the market for the price .

2. **Buyer:** Is exploring the market to select and allocate multiple concrete instances of abstract services that satisfy its required level of QoS and are bounded by a price limit that cannot be exceeded.

3. **Market regulator:** An independent agent who is responsible for monitoring trading and QoS for services exchanged in the market.

We now define some of terms used in the thesis.

- **AbstractService:** The functional specification of a task. A Cloud-based application may require multiple abstract services to implement all of its functional specification.

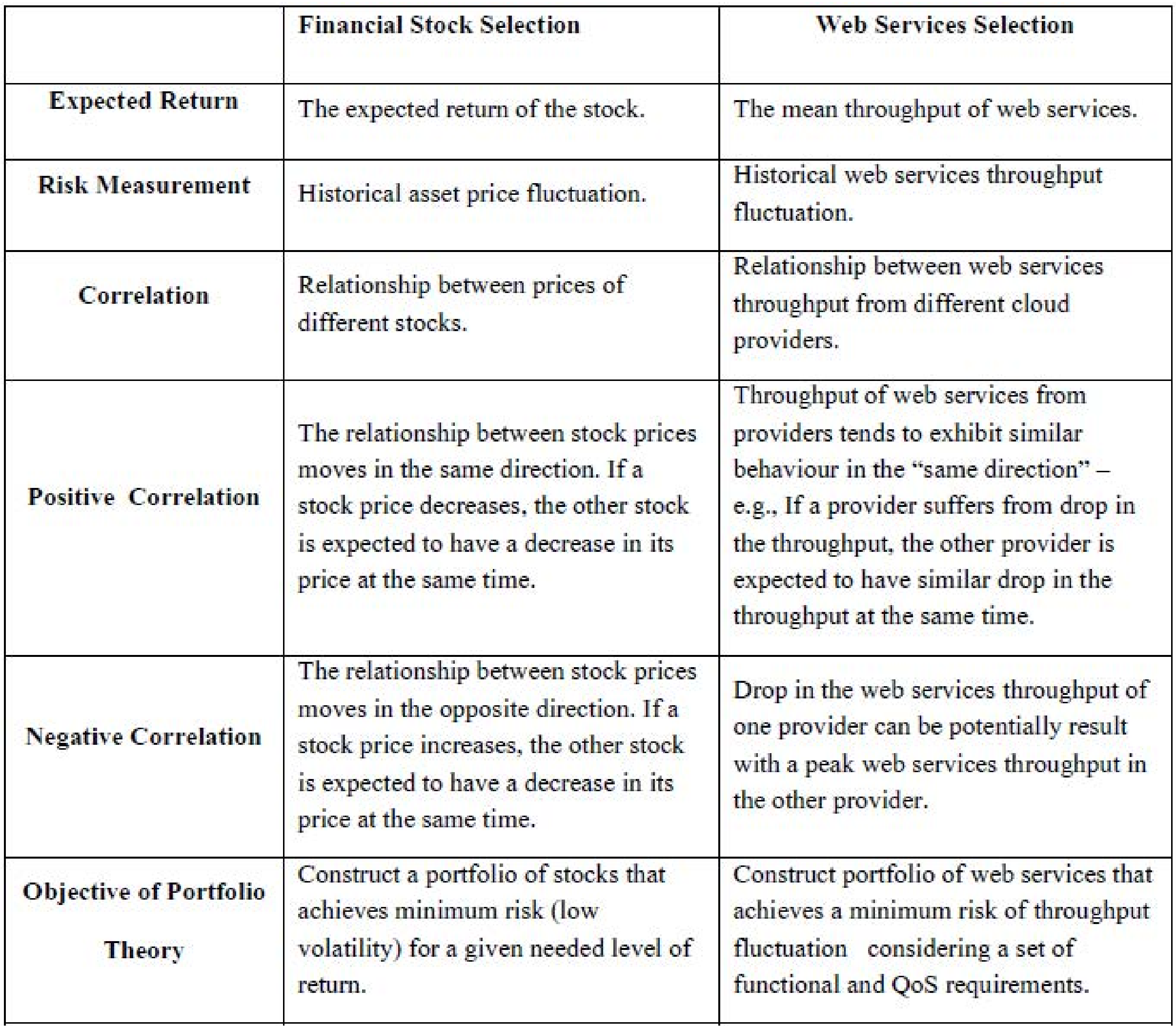
- **CandidateServices**: A SaaS implementation of abstract services that satisfy all the QoS constraints required by the buyer.

- **Workflow:** The description of the set Abstract Services and how they are composed to build an application.

In the context of cloud-based market, we argue that buyers of web services can implement design diversity by using portfolio theory to select web services from multiple sellers in a cloud-based market. A prerequisite for our diversification approach is that an abstract service has to be short-listed as a candidate for diversification and it has compatible candidate services in the marketplace. Such perspective is novel and has the potential to reduce risks of performance fluctuation and improve compliance. Table 4.1 illustrates the correspondence of the portfolio selection problem to both the financial stocks and the scaling up web services selection problem.

Previous research has used auction-based methods to allocate all the instances of web service from a single provider or multiple providers that have the lowest price and optimal QoS dynamically [24]. In contrast, our approach attempts to secure the instances by constructing a diversified portfolio of multiple instances of web services from multiple providers in the cloud-based market. The objective is to minimise the risk of performance fluctuation through diversification. Portfolio theory has found its way in numerous applications (e.g. [110], [120] [121]). However, it has not been used to improve the performance stability of cloud-based application.

Table 4.1: Correspondence of the Portfolio Selection Problem to both Financial Stock Selection and the Scaling Up Web Services Selection Problem.



	Financial Stock Selection	**Web Services Selection**
Expected Return	The expected return of the stock.	The mean throughput of web services.
Risk Measurement	Historical asset price fluctuation.	Historical web services throughput fluctuation.
Correlation	Relationship between prices of different stocks.	Relationship between web services throughput from different cloud providers.
Positive Correlation	The relationship between stock prices moves in the same direction. If a stock price decreases, the other stock is expected to have a decrease in its price at the same time.	Throughput of web services from providers tends to exhibit similar behaviour in the "same direction" – e.g., If a provider suffers from drop in the throughput, the other provider is expected to have similar drop in the throughput at the same time.
Negative Correlation	The relationship between stock prices moves in the opposite direction. If a stock price increases, the other stock is expected to have a decrease in its price at the same time.	Drop in the web services throughput of one provider can be potentially result with a peak web services throughput in the other provider.
Objective of Portfolio Theory	Construct a portfolio of stocks that achieves minimum risk (low volatility) for a given needed level of return.	Construct portfolio of web services that achieves a minimum risk of throughput fluctuation considering a set of functional and QoS requirements.

4.4 Assumptions of our Portfolio-based Web Service Selection in Scaling Up Scenario

One can see that the cloud marketplace has similarities to the financial market. On the other hand, when portfolio theory is used to support the web service selection process in scaling up scenario, a few assumptions need to be taken into account:

- The cloud-based application is a risk averse and target to select a set of services that help to reduce throughput fluctuation given a set of QoS constraints.
- The expected return E_i of investing in web service S_i is equal to the expected

throughput of web service S_i, which is measured as the mean throughput during specific period of time.

- The weight of investment w_i for each candidate service S_i present to the number of instances needed to be allocated from that candidate service.

- The market regulator, independent agent that is part of the market infrastructure, will provide an evaluation of the risk of throughput fluctuation Ri and the mean throughput of web service E_i for each web service S_i based on the historical record of delivered performance.

- We rely on recent and relevant historical records of throughput of web service S_i to predict the likely future risk of throughput fluctuation. The Risk of throughput fluctuation R_i is quantified as the standard deviation of throughput; it measures the amount by which the delivered throughput of web service S_i vary from the mean throughput of web service E_i. The higher the risk of throughput fluctuation R_i The less reliable and less stable is the throughput of service S_i.

- We assume that the market regulator will complement the allocation decisions by calculating the extent to which the candidate services are statistically correlated. One of the statistical models of [122] can be used to calculate the correlation p_{ij} based on historical records of throughput of services S_i and S_j. The correlation p_{ij} describes the direction and strength of the relationship between web service S_i and web service S_j in terms of their delivered throughput. The correlation is represented as a number between +1 and -1. In the case of positive correlation p_{ij}, if a drop in throughput takes place on web service S_i, there is great chance that it will also happen on web service S_j. A typical scenario for a positive correlation can take place when services S_i and S_j are hosted in the same data centre and/or share a common infrastructure. For example, peaks on demand on the data centre could trigger drop on throughput of both web services at the same time. However, in the case of negative correlation p_{ij}, if the drop in throughput takes place on web service S_i, there is a great chance that it will not occur in web service S_j. A typical scenario for a negative correlation can take place when the services S_i and S_j are hosted in different data centres located in two different time zones such as Japan and the U.S., where a peak at one of the locations could be reflected as a non-peak at the other location. If a drop in throughput is triggered by peaks on demand at the data centre located in the U.S.,the data centre located in Japan will probably have an off-peak demand and will have high throughput at that time.

4.5 Model For Portfolio-Based Web Service Selection in Scaling Up Scenario

Taking into account the assumptions presented in the previous section, we can apply the portfolio theory where a cloud-based application is an investor buying web services from the cloud marketplace. The cloud-based application needs to build a diversified portfolio of multiple instances of web services. The multiple web services offered in the market represent the asset. Each web service S_i will have its own risk of throughput fluctuation R_i, mean throughput E_i, price C_i and correlations p_{ij} with the other web service in the market.

Based on these values, we can then decide how many instances of a given web service need to be allocated in constructing a portfolio of services with the aim of minimising the risk of the portfolio R_p subject to a set of QoS constraints. For simplicity of exposition, we consider the cost, throughput and security as the three dimensions of QoS used to demonstrate our approach; nevertheless, the model is extensible to accommodate other QoS dimensions in the analysis:

1. Cost of a web service C_i represents the amount of money paid to allocate web service S_i.

2. Throughput of a web service E_i measures the number of users requests that web service S_i can support during unit of time.

3. Security of web service S_{ei} represents the level of security provided by the web service.

We assume that the prices and the security level are fixed for each service. On the other hand, throughput of a web service hosted in the cloud tends to fluctuate [8]. In this context, the throughput of each service S_i will be modelled by mean throughput E_i , fluctuation rate of throughput R_i. The objective is to select a set of web services that minimises R_p, where R_p is the fluctuation rate of throughput of the selected services in the portfolio, represented by equation 4.5. The minimization should also satisfy the following constraints on the selected portfolio of services:

1. The portfolio mean throughput should not be less than minimum throughput level E_{min}.
2. The price of the selected services should be less or equal to max price C_{max}.
3. Each service in the portfolio should exceed the minimum security level Se_{Min}.
4. Number of selected services should be equal to the number of required services which is 100 in this scenario.

These constraints are represented by equations 4.6, 4.7, 4.8 and 4.9. Table 4.2 shows a brief description of the variables.

$$min(R_p) = min \sqrt{\sum_{i=1}^{m} w_i^2 \; R_i^2 + \sum_{i=1}^{m} \sum_{j \neq i}^{m} w_i \, w_j \; R_i \, R_j \; p_{ij}} \quad (4.5)$$

Subject to

$$E_{min} \leq \sum_{i=1}^{m} w_i E_i \quad (4.6)$$

$$C_{max} \geq \sum_{i=1}^{m} C_i \quad (4.7)$$

$$\forall \; s_i \epsilon \, Portfolio : \; Se_{min} < Se_i \quad (4.8)$$

$$\sum_{i=1}^{m} w_i = w_r \quad (4.9)$$

Table 4.2: Variables Description

Variable	Description
R_p	Fluctuation Rate of throughput of selected services in the portfolio
w_i	Weight of investment w_i which is equal to the number of selected services from wen servies S_i.
w_r	Required number of services in the portfolio.
E_i	Mean throughput of service S_i.
R_i	Standard deviation of throughput of service S_i.
FR_j	Standard deviation of throughput of service S_j.
p_{ij}	Correlation of throughput between service i and service j.
E_{min}	Minimum throughput of services accepted by the user.
Se_i	The level of security of the services S_i.
Se_{Min}	Minimum accepted level of security accepted by the user .
C_{max}	Maximum Cost of service that the buyer is willing to pay.
C_i	Cost of allocating services CS_i.
m	The number of providers for the web services in the portfolio.

In order to construct the low-risk portfolio R_p, we need to determine how many instances should be selected from each web service S_i. To find the weights of the portfolio, we have used *quadprog*, a well-known solver for optimization problems; it is part of the MATLAB optimization package. The goal of this optimisation process is to find the optimum number of web services instances from a specific provider to construct portfolio with minimum risk of throughput fluctuation. The process of mapping our problem to *quadprog* is done by using equation 4.5 as a fitness function and equations 4.6, 4.7, 4.8 and 4.9 as constraints. The result will be a vector containing the recommended number of services in the portfolio.

4.6 Self-Adaptation Mechanism for Portfolio-Based Web Service Selection in Scaling Up Scenario

In this section, we will present a self-adaptive strategy for realising and implementing the portfolio-based approach in highly dynamic market settings, where multiple sellers and buyers can continuously trade web services with dynamic prices and QoS. The system will be self-adaptive to changes that may happen in the market. The system has to construct an optimum portfolio of web services and modify it in response to the perception of the market in a timely manner.

According to De Lemos et al. [34], adaptation control can be achieved by a sequence of four components: *Monitoring, Analysing, Planning and Executing* (MAPE). These components, when put together, form the building blocks for feedback control systems as explained in control theory. Because scalability is a key concern for our system, we will use a decentralised pattern, where each buyer agent will have its local M, A, P and E components and will interact with other peer agents directly, as represented in Fig. 4.3.

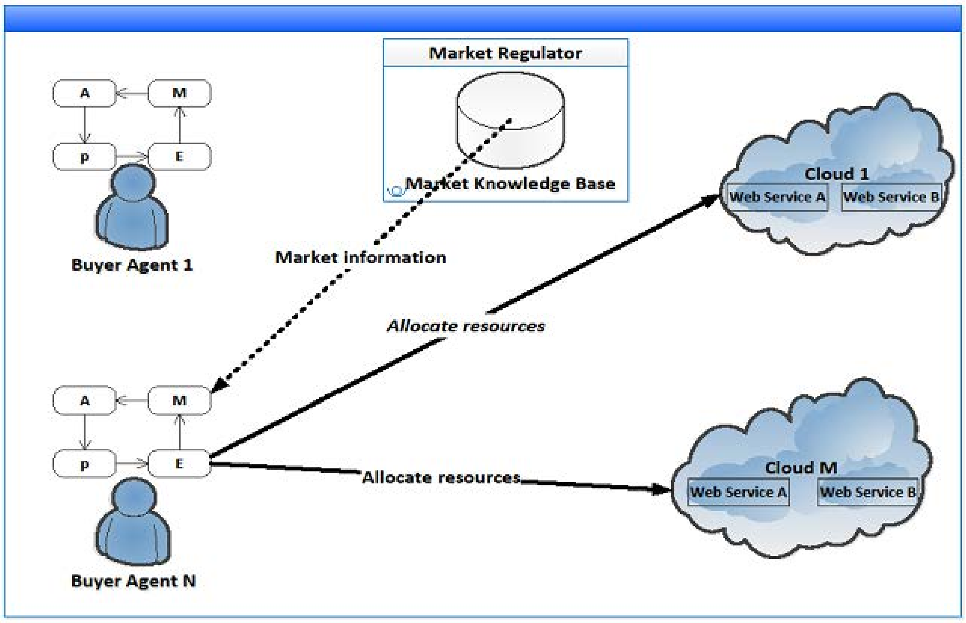


Figure 4.3: Overview of Self-Adaptive System Implementation

In order to achieve a global consistent view of the market status, we use a Knowledge-Base which is going to be implemented in a publish/subscribe architectural style. The KnowledgeBase coordinates knowledge of traded web services in relation to QoS, price and risks. Such architecture has the potential of scalability, where the KnowledgeBase will eliminate the overheads of potential interaction between the buyers and the sellers in each trading cycle. The monitoring of QoS in real time are beyond the scope of this thesis. The works of Keller [123] , Michlmayr [124] and Zhang et al. [125] are prominent examples of online QoS monitoring.

In a decentralised architecture, the KnowledgeBase will be used as a reference point by the buyer agents. For each abstract web service offered in the market, the KnowledgeBase will provide the following information:

- A list of the different offers in the market with the prices, QoS and number of available services.

- Historical data related to the throughput and its fluctuation rate for each web service.

There are two important questions in realising the adaptation mechanism: What are the dynamics, which trigger observations? and, what does trigger the adaptation?

The Monitoring and Analysis

In this phase, we attempt to answer what triggers observation in our system. Therefore, we will use an event-based observation, which will help our system to gain a lightweight interaction between the agents and market KnowledgeBase. First, the buyers agent subscribes to the abstract web service which he is interested in (e.g. *FlightBooking* web services or *PhotoStorage* web services). Subsequently, a change in the prices or risk of these abstract web services will trigger an observation and activate the control loop.

In the monitoring component, the buyers agent retrieves all the key information about the web services from the market KnowledgeBase. Subsequently, the buyer analyses the information to get the services that matches his preference by using Equations 4.6, 4.7 and 4.8. This will make the agent ready to start the planning and execution phases.

The Planning And Execution

Here we attempt to determine the answer to the second question regarding what triggers adaptation in our system. The first step is to evaluate the currently allocated portfolio risk $R_{pcurrent}$ and the optimum potential portfolio risk $R_{poptimum}$ that we could allocate based on the new market state by using Equation 4.5.

I_c presents the level of improvement that the system could gain by allocating the new optimum portfolio $R_{poptimum}$. In other words, I_c represents the potential improvement in risk between the current portfolio risk $R_{pcurrent}$ and the new optimum portfolio $R_{poptimum}$, I_c is calculated as in Equation 4.10. A positive number represents an improvement in the portfolio risk.

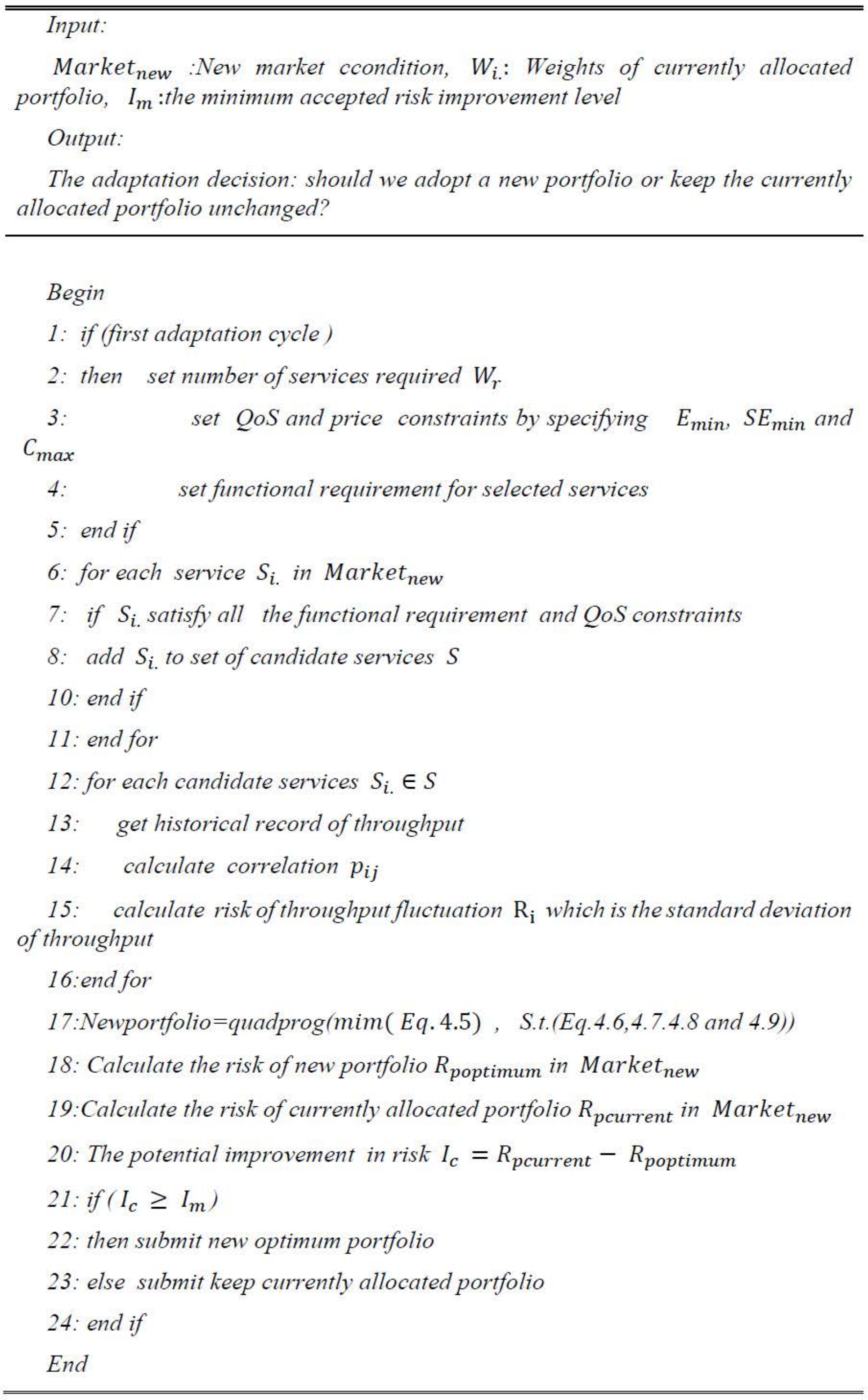
$$\boldsymbol{I_c = R_{pcurrent} - R_{poptimum}} \qquad (4.10)$$

Furthermore, with the reallocation of web services there will be an extra cost in the form of penalties for breaking the contract. Therefore, the cost of the adaptation process will be the cost of allocating the new web services plus the cost of penalties for breaking any contract. A buyer will set I_m, which represents the minimum accepted improvement level in the new portfolio, to trigger the adaptation and replace the current portfolio with a new optimum one. This will make the adaptation of the new optimum portfolio subject to the satisfaction of two conditions:

- The cost of constructing the new optimum portfolio plus the penalties is less than the price limit C_{max} .
- The level of improvement I_c in the new optimum portfolio is greater than or equal to the minimum level of accepted improvement I_m .

A cloud-based application is expected to perform adaptations when there is a change in the market prices or risks. The basic steps of our self-adaptive portfolio-based optimisation are listed in table 4.3.

Table 4.3: Algorithm of Self-Adaptive Portfolio-Based Optimisation for the Scaling Up Scenario



```
Input:
$Market_{new}$ :New market ccondition, $W_{i.}$: Weights of currently allocated portfolio, $I_m$ :the minimum accepted risk improvement level
Output:
The adaptation decision: should we adopt a new portfolio or keep the currently allocated portfolio unchanged?

Begin
1: if (first adaptation cycle )
2: then   set number of services required $W_r$
3:           set QoS and price constraints by specifying $E_{min}$, $SE_{min}$ and $C_{max}$
4:           set functional requirement for selected services
5: end if
6: for each service $S_{i.}$ in $Market_{new}$
7:  if $S_{i.}$ satisfy all the functional requirement and QoS constraints
8:  add $S_{i.}$ to set of candidate services $S$
10: end if
11: end for
12: for each candidate services $S_{i.} \in S$
13:     get historical record of throughput
14:     calculate correlation $p_{ij}$
15:     calculate risk of throughput fluctuation $\mathrm{R_i}$ which is the standard deviation of throughput
16:end for
17:Newportfolio=quadprog($mim(Eq.4.5)$ , S.t.(Eq.4.6,4.7.4.8 and 4.9))
18: Calculate the risk of new portfolio $R_{poptimum}$ in $Market_{new}$
19:Calculate the risk of currently allocated portfolio $R_{pcurrent}$ in $Market_{new}$
20: The potential improvement in risk $I_c = R_{pcurrent} - R_{poptimum}$
21: if ( $I_c \geq I_m$ )
22: then submit new optimum portfolio
23: else submit keep currently allocated portfolio
24: end if
End
```

The algorithm in the first adaptation cycle asks the buyer to set the minimum accepted QoS , the required number of web services and the maximum price that s/he is willing to pay for the web service in line(1-5). Then, the algorithm identifies the web services S_i which offer the functionalities required by the user and satisfy the QoS and cost constraints in line (6-11). After that, the buyer agent will access the KnowledgeBase, to retrieve throughput historical record and calculate throughput fluctuation risk R_i and correlation matrix p associated with each candidate web service in line (12-16).

The quadprog function, provided by the MatLab optimization toolbox, will return a vector that contains the weights of each web service in a new optimum portfolio. After that, we calculate the risk of the new optimum portfolio and the currently allocated portfolio in line (17-19). Finally, we calculate the level of improvement in risk I_c and if it exceeds the minimum accepted improvement level I_m, an adaptation will be triggered to allocate new set of web services based on the recommended weights from the new optimum portfolio in line (20-24).

4.7 Evaluation

We demonstrate how a cloud-based application can reduce the throughput fluctuation risks associated with web service allocation through simulating a hypothetical cloud market. We compared the risk reduction achieved by portfolio-based selection with the following non-diverse auction-based mechanisms of [24] , where auctioning is based on risk or price algorithm and classical design diversity algorithm [100] :

1. *Risk-Based Auction*: that allocates all the required services from a single cloud provider that have lowest risk of throughput fluctuation.

2. *Price-Based Auction*: that allocates all the required services from a single cloud provider that have the lowest price.

3. *Classical Design Diversity*: that implements diversity by evenly distributing the ser-

vices among multiple providers without considering the risk or correlation between the providers.

The evaluation is designed to test how our portfolio-based selection can be applied to a scaling up scenario, for experiments 1-3, we have considered a scenario involving three candidate services for simplicity of exposition; nevertheless, the argument and the technique can be applied to more than three services. Furthermore, the number of considered providers tends to be limited to a few, as dealing with different providers may come with costs and overheads. Experiment 4 is aimed at stress-testing the mechanism for scalability. We have used 400 candidate services. The choice of 400 is aimed to test/stress for scale and exhibit an extreme scenario. A systematic literature review had revealed that the current approaches typically use 20 candidate services in dynamic selection and composition problem [126]; henceforth, 400 goes far beyond the classics.

The following is a description of the rationale for the choice of each experiment:

1. The first experiment is designed to evaluate the effectiveness of the portfolio-based allocation in reducing the risk of throughput fluctuation in comparison to price-based auction, risk-based auction and classical design diversity.

2. The second experiment evaluates the self-adaptive behaviour of the portfolio-based allocation by changing the risk of throughput fluctuation in the cloud market.

3. The third experiment is designed to assess the sensitivity of the approach to changes in correlation between providers in the optimal portfolio selection. We compare the approach to the classical design diversity approach.

4. The fourth experiment is designed to test the scalability of the approach. It attempts to determine how the risk of throughput fluctuation and execution time of the portfolio selection tend to be affected by increasing the number of candidate services in the portfolio.

4.7.1 The Effectiveness of the Portfolio-Based Allocation

Recall the case of the flight booking website, Flight.com, which provides the online booking web service FlightBooking. As mentioned previously, the variants of the web service instances from various providers tend to provide the same core functionalities, but they differ in price and the way they address QoS.

In high seasons, Flight.com has decided to scale up its services to support an anticipated load in the number of users through selecting and subsequently allocating 100 instances. Let us assume that the minimum accepted value of QoS is as follows: 68 for throughput and 80 for security. Moreover, let us assume that the maximum price that they are willing to pay is $32 for each instance.

Table 4.4 shows a snapshot of the current FlightBooking web service offers that are available in the simulated cloud-based market which satisfy the QoS and price constraints as well as the risk of throughput fluctuation associated with each web service. Table 4.5 shows the correlation of throughput between the current FlightBooking web services. The QoS represents a normalised QoS values inspired from the work of Zeng et al. [72] and the fluctuation rate are driven from the work of Dejon et al. [8].

Table 4.4: Current Flightbooking Web Service Offers Available in the Cloud Market at Condition 1.

Web Service	Security	Throughput	Price	Risk of throughput fluctuation
FlightBooking 1	81	70	\$30	14%
FlightBooking 2	87	69	\$32	7%
FlightBooking 3	85	72	\$31	12%

Table 4.5: The Correlation Matrix Of Throughput Between The Current Flightbooking Offers Available in The Cloud Market at Condition 1.

Web Service	FlightBooking 1	FlightBooking 2	FlightBooking 3
FlightBooking 1	1	.4	-.3
FlightBooking 2	.4	1	0
FlightBooking 3	-.3	0	1

For this experiment, the simulation methodology consists of two steps:

1. Performing web services allocation to the simulated market state using:
    - Portfolio-based allocation.
    - Classical design diversity allocation.
    - Price-based allocation (minimum price).
    - Risk-based allocation (minimum risk).
2. Allocating the web services in the simulated market and calculating the risk of throughput fluctuation. The results shown are the averaging of 30 repetitions.

Web Service Allocation Using The Portfolio-Based Method

When we use portfolio-based optimisation to allocate instances of web services, the first step is to find the number of instances that we should allocate from each of the candidate services in order to achieve the minimum portfolio risk. We can find the numbers by applying portfolio optimisation, where the goal is to minimise the fluctuation risk in Equation 4.5 while Equations 4.6, 4.7, 4.8 and 4.9 are constraints. The optimum weights of the optimisation process are depicted in figure .4.4.

These weights imply that we will be able to construct the minimum risk portfolio of a web service by allocating 70 service instances from FlightBooking2 at the cost of $32 for each service and 30 services from FlightBooking3 at the cost of $31 for each service. This makes the total cost of constructing this portfolio to total $3170. After allocating the web service as in the optimum portfolio and running the simulation 30 times, a box plot shows the percentage of throughput fluctuation of the portfolio-based methods in figure 4.5. The average throughput for the portfolio was 70.6 with a risk of throughput fluctuation of 5.14%.

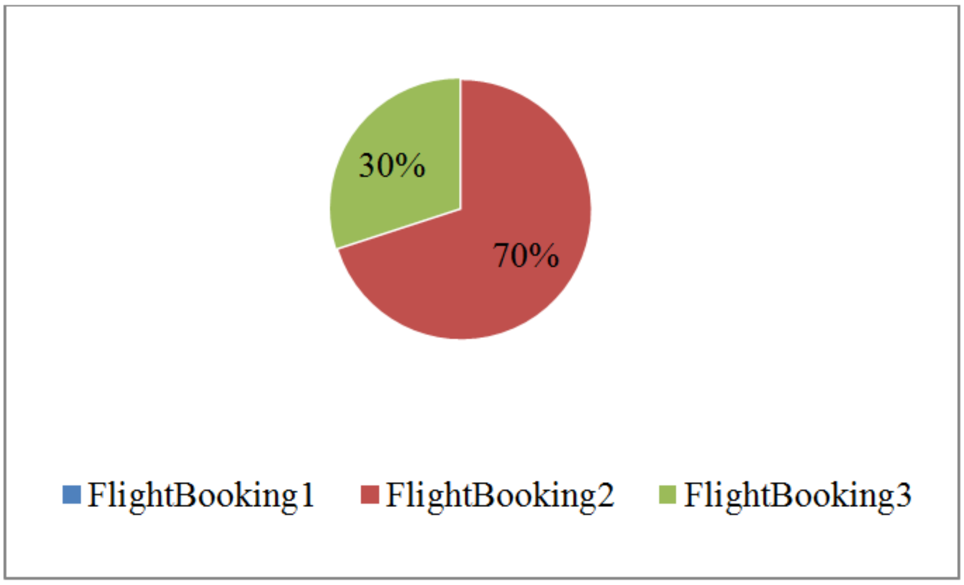


Figure 4.4: Optimum Weight of Allocation for Each of the Traded Web Services in Cloud Market in Condition 1.

Web Service Allocation Using The Price-Based Method

When Flight.com used a price-based auction method to select the web service with the lowest price, the result was the allocation of 100 web service instances from FlightBooking 1 with a cost of \$3000 and the average throughput was 70 with a risk of throughput fluctuation of 14%.

Web service Allocation Using the Risk-Based Method

Conversely, when Flight.com used the risk-based auction method to select the web service with lowest risk, the result was the allocation of 100 web service instances from FlightBooking 2 at the cost of \$3200, and the average throughput was 69 with a risk of throughput fluctuation of 7%.

Web Service Allocation Using The Classical Design Diversity Method

When Flight.com used the classical design diversity method in order to evenly distribute the allocation among candidate web services, the allocation was as follows: 33 web services from FlightBooking1 and another 33 web services from FlightBooking 3. The remaining web services (i.e. 34)was allocated from FlightBooking 2. The total cost of allocation was \$3101. Based on the simulation result, the average throughput was 70.33 with a risk of throughput fluctuation of 6.7%.

The risk of throughput fluctuation, average throughput and cost for the web service allocation process for the Flight.com example using the price-based auction, risk-based auction, classical design diversity and our portfolio-based allocation are shown in Fig. 4.6.

We can see from the results of the allocation process in Fig. 4.6. that the portfolio-based approach has achieved the minimum risk of throughput fluctuation because it utilises the portfolio concept to diversify the allocation of web services among two providers instead of relying on one provider to allocate all the needed instances of web services as in the auction-based mechanism.

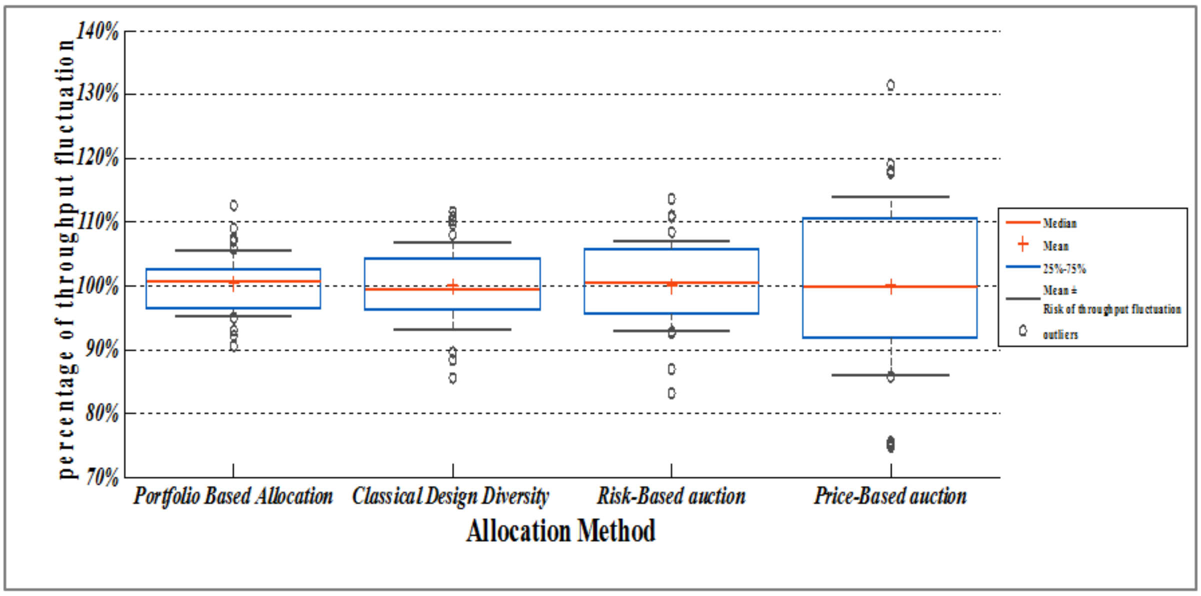


Figure 4.5: Box Plot Depicting Throughput in Terms of Mean, Median And Risk of Throughput Fluctuation of Price-Based Auction, Risk-Based Auction, Classical Design Diversity And Portfolio-Based Allocation.

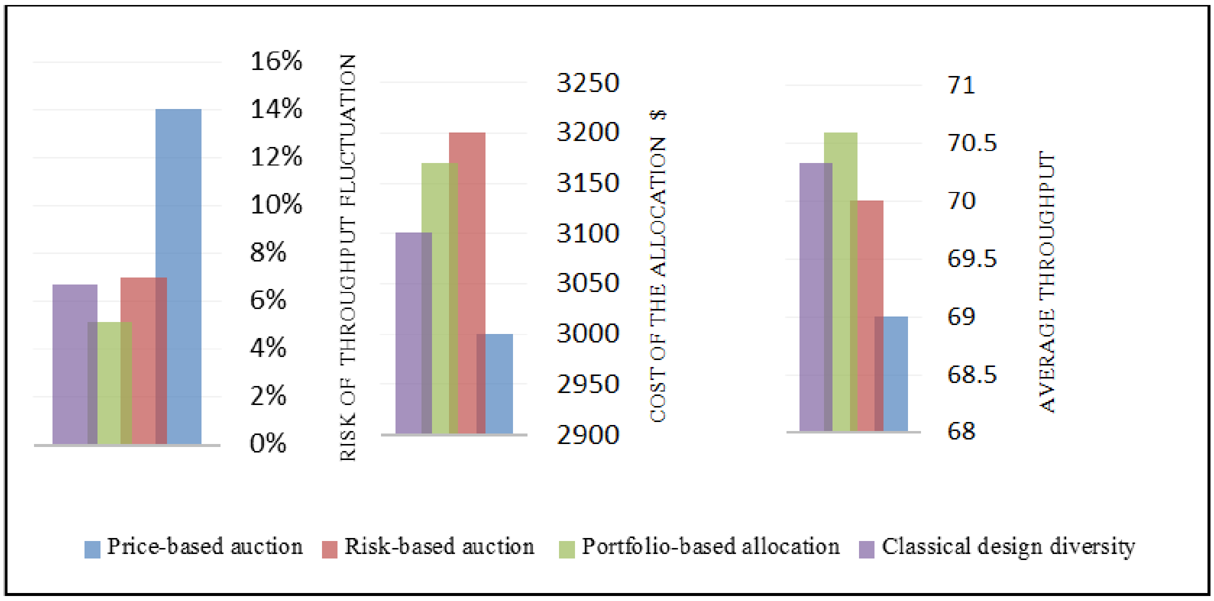


Figure 4.6: Results of the Web Service Allocation Process for Flight.com in the Current Cloud Market.

Moreover, using a portfolio to diversify the instance allocation has outperformed the classical diversity mechanism as the diversification of resources in the portfolio-based optimisation is directly linked to the risk and correlation between the providers. Portfolio theory maintains that effective diversification can be achieved by seeking a low correlation between the web services providers.

The portfolio-based selection diversified the allocation of web services among the two low correlation providers (FlightBooking 2 and FlightBooking 3) and did not consider the third provider (FlightBooking 1) that shared a higher correlation with the other providers (0 with FlightBooking 3 and +.4 with FlightBooking 2). The portfolio-based allocation deals with the cost and level of aggregated QoS as constraints that need to be satisfied. However, it does not seek the optimality on either of them. In terms of cost, the portfolio-based allocation is not the cheapest option. However, the risk reduction would justify the additional cost from the point of view of a risk-averse buyer.

4.7.2 The Effectiveness of the Self-Adaptive Portfolio-Based Allocation

In this experiment, we evaluate the effectiveness of the self-adaptive behaviour where the approach should systematically evaluate the risks of the current portfolio and compare it to the optimal traded portfolio at a given time. It should then dynamically decide on a new portfolio and adapt the application accordingly, as detailed in table 4.3 (see Section 4.6).

We assume that currently the allocated portfolio is as the one displayed in Fig 4.4 . To test our self-adaptive approach, we assume that we have two new market conditions where cost and QoS of web services remain unchanged, as in Table 4.4, and their correlation, as in Table 4.5. However, the risk of throughput fluctuation is changing, as shown in Table 4.6.

Table 4.6: Risk of Throughput Fluctuation for Flightbooking Web Service Offers Available in the Cloud Market on Conditions 1, 2 and 3.

Web Service	**Current Risks (Condition 1)**	**Risks (Condition 2)**	**Risk (Condition 3)**
FlightBooking 1	14%	15%	6%
FlightBooking 2	7%	12%	14%
FlightBooking 3	12%	11%	12%

For each new market condition, we will evaluate how a change in the risk of throughput fluctuation at a new market condition can affect the previously allocated portfolio in market condition 1. This experiment assumes that Flight.com requires 4% improvement in the risk of throughput fluctuation to adopt a new optimum portfolio. This implies that we will not change the currently allocated portfolio unless the new portfolio presents more that 4% as risk reduction.

Change in Risk of Throughput Fluctuation in the Cloud Market from the Current Condition (Condition 1) to Condition 2.

First, we re-run the portfolio-based optimisation to find the new optimum portfolio weights according to market condition 2. Fig. 4.7 shows the weights of the currently allocated portfolio (allocated according to condition 1) and the optimum portfolio (allocated according to condition 2 and condition 3).

Moreover, Fig. 4.8 shows a box plot representation for the percentage of throughput fluctuation of the currently allocated portfolio and new optimal portfolio in market condition 2. In market condition 2, the risk of the currently allocated portfolio is 8%, and the new optimum portfolio risk is 6.7%. We can gain 1.3% improvement as risk reduction if we switch from the currently allocated portfolio and adopt the new optimum.

However, Flight.com requires a minimum 4% improvement in risk to relocate the portfolio. As a result, the change from the current market condition (condition 1) to condition 2 will not cause any changes in the currently allocated portfolio.

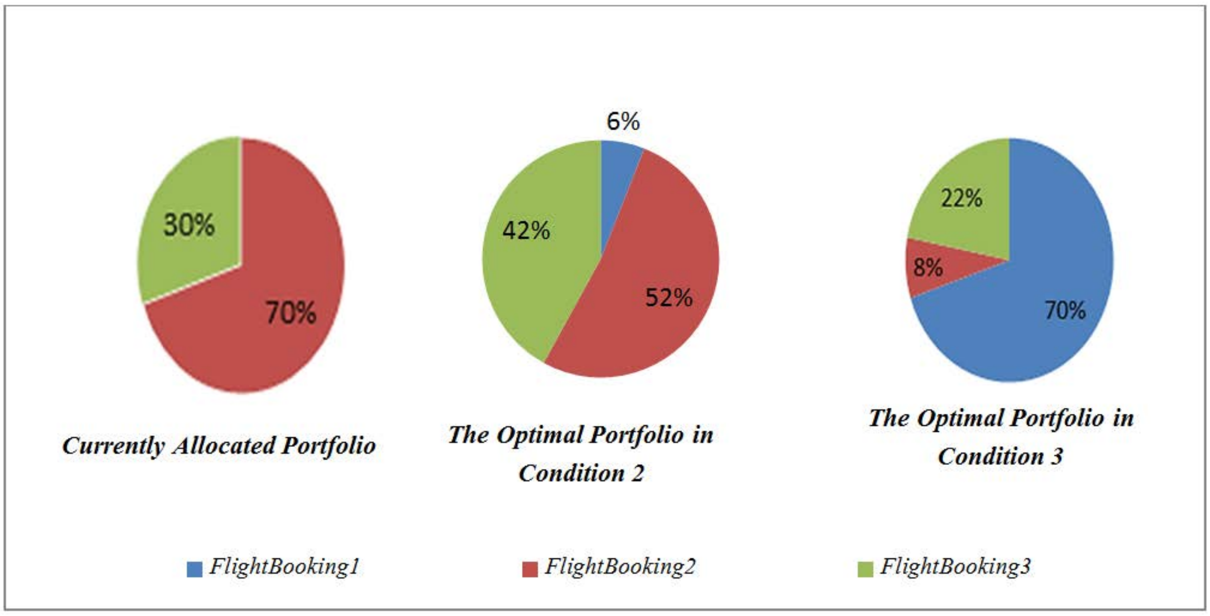


Figure 4.7: The Currently Allocated Portfolio and the New Optimum Portfolios in Market Conditions 2 and 3.

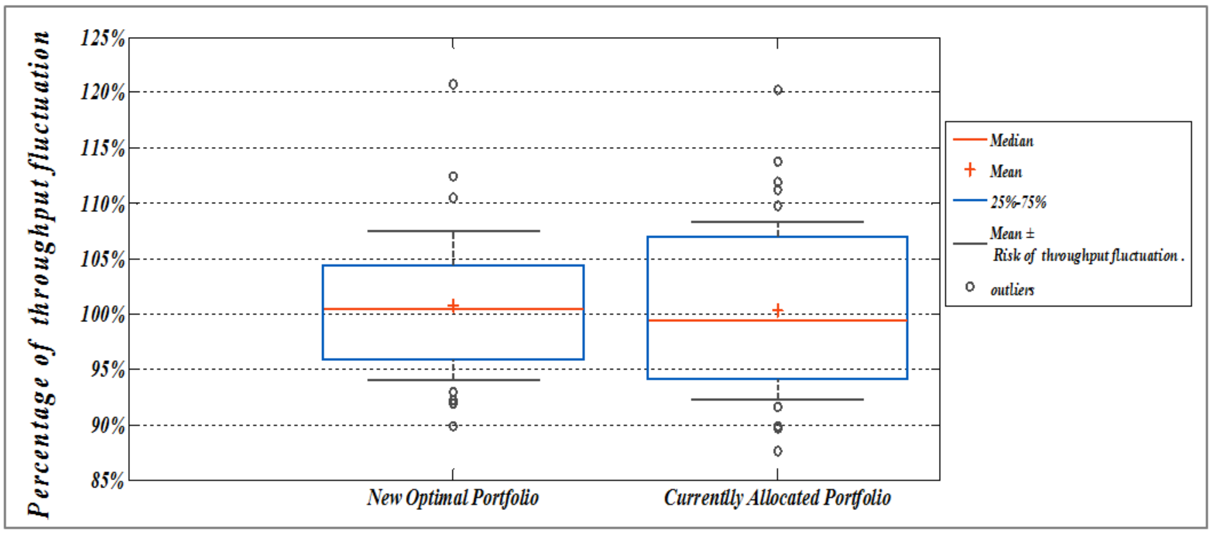


Figure 4.8: Box plot Depicting Throughput in terms of Mean, Median and Risk of throughput fluctuation of the Currently Allocated portfolio and the New Optimal Portfolio in Market Condition 2.

Change in Risk of Throughput Fluctuation in The Cloud Market From The Current Condition (Condition 1) To Condition 3.

We will re-run the portfolio-based optimisation to find the new optimum portfolio weights according to market condition 3. In Fig. 4.7, we can see both the distribution of the currently allocated portfolio and the new optimum portfolio (allocated according to condition 3). Moreover, in Fig. 4.9 we can see a box plot presenting the risk of throughput fluctuation of the currently allocated portfolio and the new optimal portfolio in market condition 3.

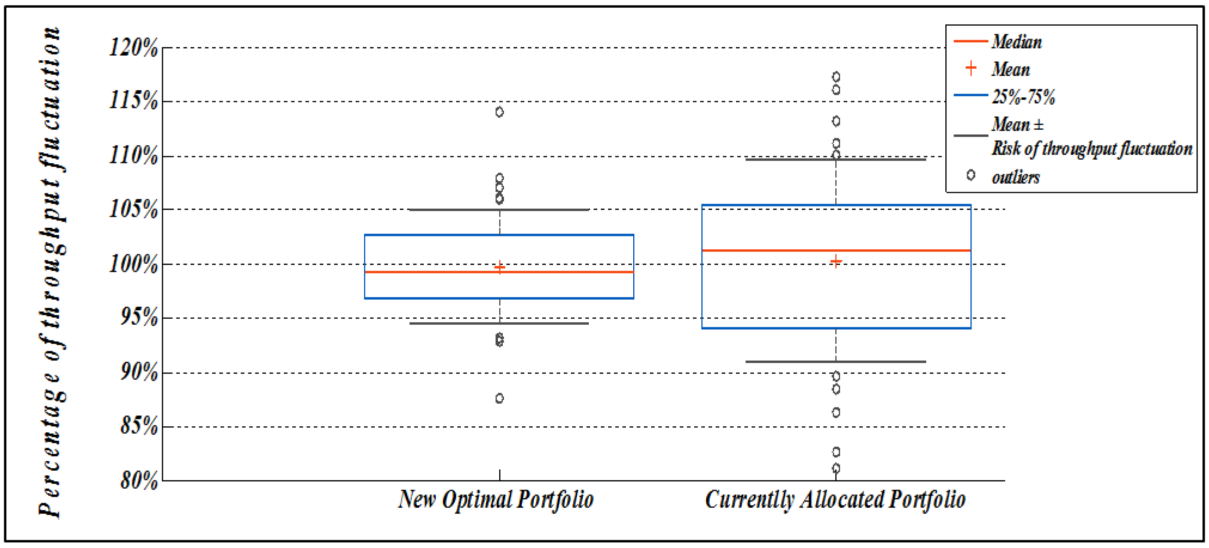


Figure 4.9: Box Plot Depicting Throughput in Terms of Mean, Median and Risk of Throughput Fluctuation of The Currently Allocated Portfolio and New Optimal Portfolio in Market Condition 3.

In market condition 3, the risk of the currently allocated portfolio is 9.37%, and the new optimum portfolio risk is 5.2%. We can gain a 4.17% improvement in risk if we adopt the new optimum, which satisfies the adaptation requirement. As a result, the change from the current market condition to condition 3 will trigger the adoption of the new optimal portfolio by allocating 70 services from FlightBooking 1, 8 services from FlightBooking 2 and 22 services from FlightBooking 3.

4.7.3 Correlation Sensitivity in Portfolio-Based Allocation and Classical Design Diversity Allocation

In this experiment, we will evaluate the effect of change in the correlation between providers on the selection process using portfolio-based allocation and classical design diversity allocation. To do this, we run 90 simulations with the configuration, as in Table 4.4, except that correlation will change between FlightBooking 3 and FlightBooking 1. The correlation will vary in each experiment by moving from +0.90 to -0.88, stepping 0.02 at each simulation.

In portfolio-based allocation, as we moved toward negative correlation, the allocation of resources changes. More and more resources were allocated to FlightBooking 1 and 3, as shown in Fig. 4.10, as the low correlation makes them an attractive option to improve the diversification of the portfolio. As the allocation changes, the risk of throughput fluctuation and mean changes; the results are displayed in Fig. 4.11.

Unlike the portfolio-based approach, a change in correlation will not cause any change in the classical design diversity allocation, which is not a correlation sensitive method. In classical design diversity, the allocation will be distributed evenly among candidate web services; the result will be allocating 33 web services from FlightBooking 1 and another 33 web services from FlightBooking 3. The remaining web services (i.e. 34) have been allocated from FlightBooking 2. The throughput mean for this allocation will not be affected by the correlation change and will remain at 70.33, as shown in Fig. 4.11(a). However, as the correlation changes, the risk of throughput fluctuation changes (see Fig. 4.11(b)).

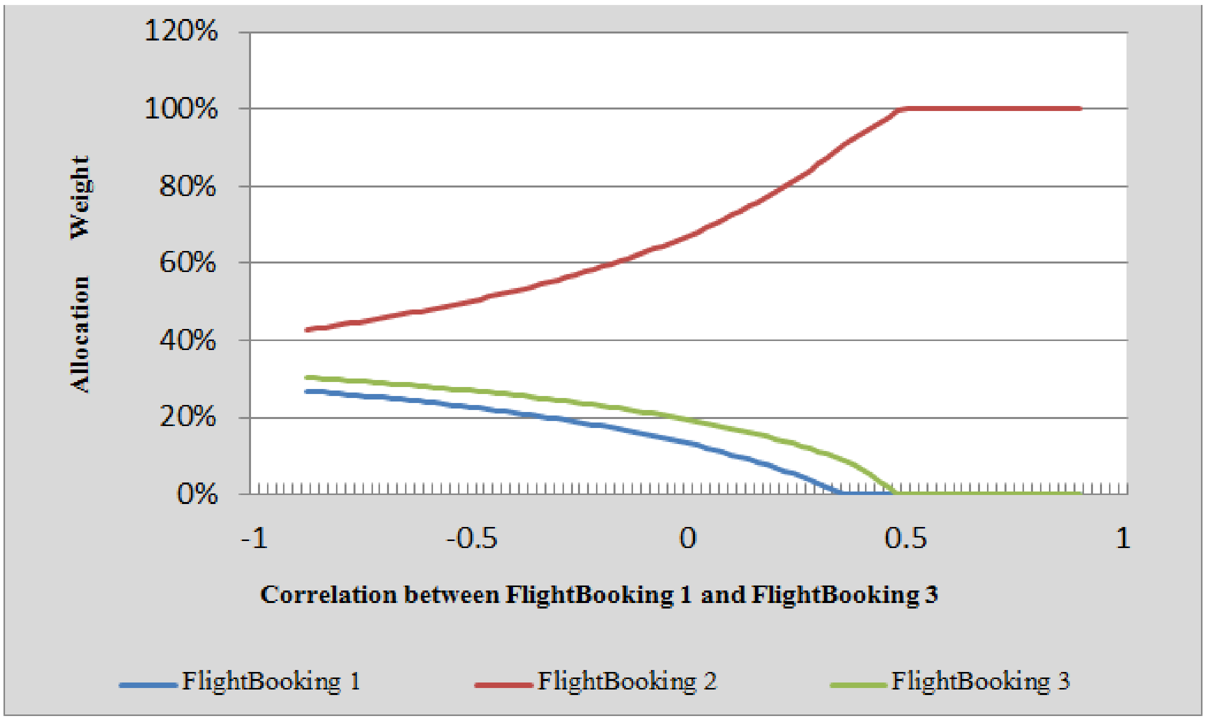


Figure 4.10: Changes in the Allocation of the Optimum Portfolio as the Correlation Between FlightBooking 1 and 3 Change from a Positive to Negative Correlation.

In the case of portfolio-based allocation, when we analyse the results displayed in

Figs. 4.10 and 4.11 and when the correlation between FlightBooking 1 and FlightBooking 3 was highly positive(+1 to +.5), all of the services were allocated to one provider, which is Flightbooking 2 as it has a 7% risk of throughput fluctuation, while providers FlightBooking 1 and FlightBooking 3 have higher risk (12% and 14%, respectively). When the correlation between FlightBooking 1 and FlightBooking 3 decreases below +.5, the portfolio-based optimisation recommends allocating more and more resources from FlightBooking 1 and FlightBooking 3.

As we move toward a negative correlation, we managed to reduce the risk of throughput fluctuation of the portfolio below 7% until it reaches 0% risk of throughput fluctuation when the correlation between FlightBooking 1 and FlightBooking 3 reach -0.60. After this point, the improvement in the portfolio is limited to the throughput mean, as seen in Fig. 4.11.

In the case of classical design diversity, there were no changes in terms of the allocation, but the risk of throughput fluctuation is reduced as the correlation between FlightBooking 1 and 3 changed from a positive to negative correlation. In the beginning, the risk was around 11% and subsequently started to decrease until it reached 0% when the correlation between FlightBooking 1 and FlightBooking 3 reach -0.66.

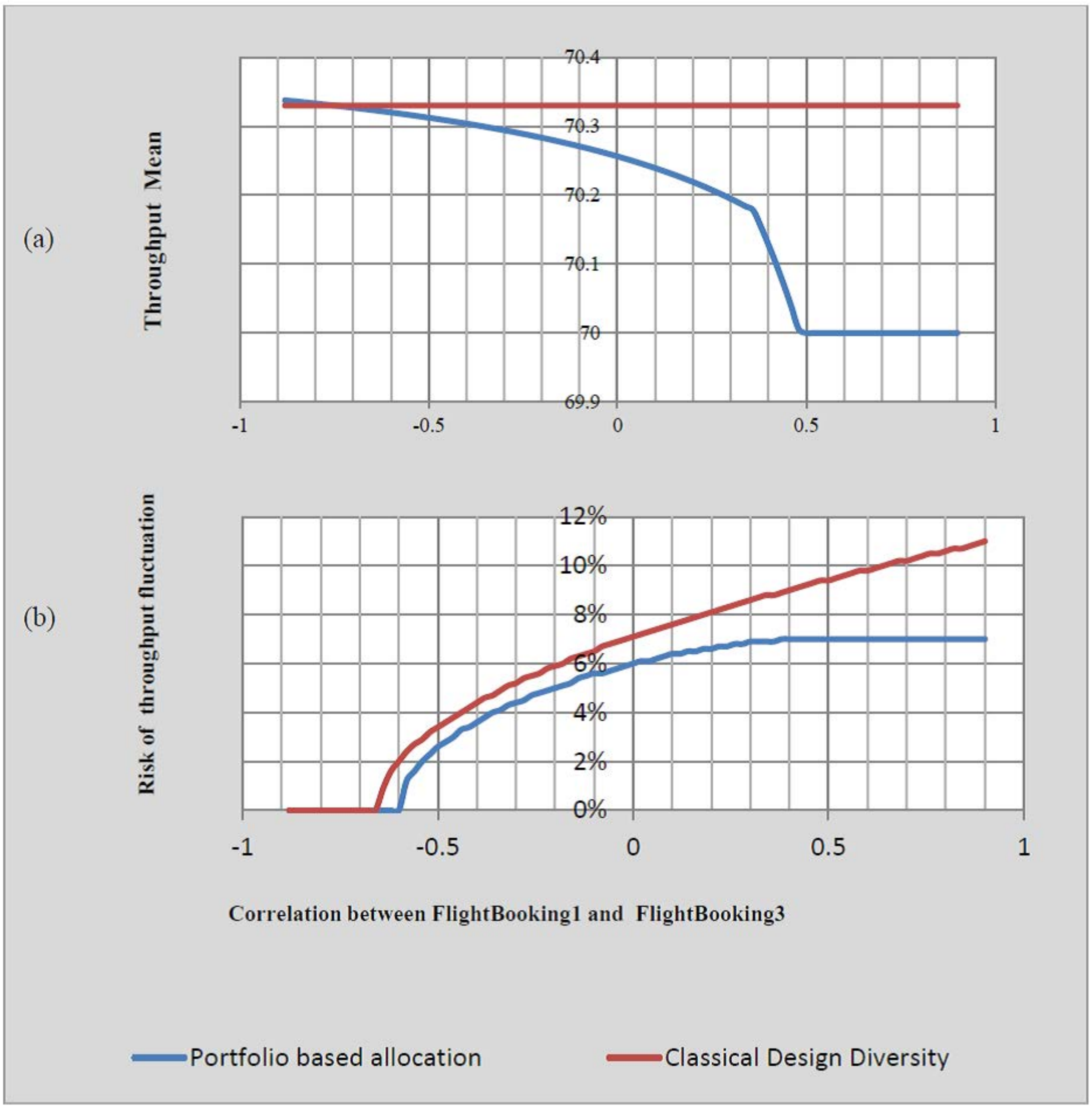


Figure 4.11: Change in the Throughput Mean and Risk of Throughput Fluctuation for Portfolio-Based Allocation and Classical Design Diversity Allocation as the Correlation Between Flightbooking 1 and 3 Change from a Positive to Negative Correlation.

We can see in Fig. 4.11 (a) that the portfolio-based approach has outperformed the classical design diversity in terms of risk reduction because it reacts to changes in correlation by reallocating the resources. As the priority in the portfolio-based approach is for risk reduction, the classical design diversity had a small lead in terms of the achieved throughput mean. However, when the risks reached 0%, the focus of our approach was shifted towards the throughput mean improvement, and it outperformed the classical design diversity in terms of throughput mean when the correlation was -0.76.

The logic behind risk reduction in both of the allocation methods is related to correlation change. A positive correlation represents a strong relationship between different providers with a similar direction. If a drop in throughput affects one provider, there is a great chance that it will happen to the other. On the other hand, a negative correlation indicates a strong relationship with an opposite direction. In the case of negatively correlated providers, if a drop in throughput occurs in one provider, there is great chance that it will not happen to the other. As a result, a negatively correlated portfolio will outperform a positively correlated portfolio in terms of risk. In conclusion, we found that a well-diversified portfolio should have a negative correlation between the throughputs of the different providers.

4.7.4 The Scalability and the Effect of Increasing the Number of Web Candidate Services on the Risk of throughput Fluctuation of the Portfolio-Based Allocation

In this experiment, we evaluate the scalability of the approach and the effect of increasing the number of candidate services on the risk of throughput fluctuation and execution time of portfolio-based allocation. We assume that the overhead of dealing with different providers is minimal, and it is possible to distribute our allocation over 400 candidate services from different providers.

We will run 397 simulations starting with three candidate services in the first simulation. Then, we will be adding additional providers at each run to finish with 400 candidate services in the last simulation. All of the providers satisfy the cost and QoS constraints and have the same 10% risk of throughput fluctuation. The correlation between the different providers is equal to 0 (independent providers) in this experiment. The result of the experiment is displayed in Fig. 4.12.

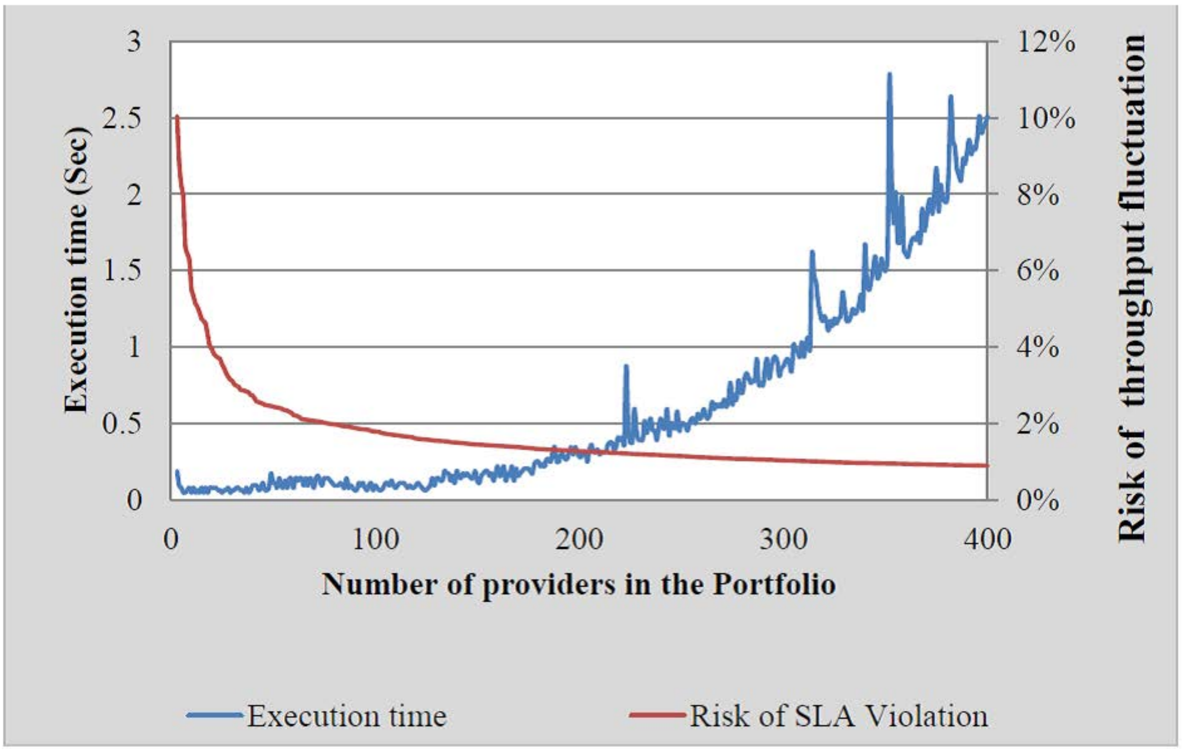


Figure 4.12: Change in the Risk of Throughput Fluctuation and the Execution Time of the Portfolio Selection as the Number of Candidate Services Increases from 3 to 400.

In this experiment, we have demonstrated the scalability of our approach by stress testing our approach by using 400 candidate services, which exceeds the 20 candidate services that is typically used in the literature of dynamic resource allocation [126]. Fig 4.12 shows that the execution time of the portfolio selection was less than 0.5 seconds when the number of providers was 200. However, when the number of providers was 400, the execution time of the portfolio selection jumped to 2.5 seconds.

Moreover, the result shows that the risk decreases as the number of different candidate services in the portfolio increases. The logic behind this is as we increase the number of providers, the portfolio becomes more diversified. If a drop in throughput occurs in one of the candidate services, there is a great chance that it will not have any effect on the other resources as they are allocated to different independent providers. So, as the number of providers increases, the portfolio becomes more diversified and will have a lower risk.

Furthermore, we assumed in this experiment that the cost of dealing with a large number of providers is minimal. However, in real life there is a trade-off that needs to be considered between the overhead and the increasing maintenance and operations costs of dealing with a large number of providers against the likely benefits that we could gain in risk reduction. Our experiments, for example, have shown that as the number of providers increase, the risk tend to decrease through diversification. The improvement, however, comes with overheads, which can be case specific and can be weighed against the benefits.

Indeed, trade-offs are the norm in cloud-based architectures, where trade-offs between maintenance costs and risks reductions are among the many others that architect should consider in the diversification decisions. However, we envision that the solution can leverage on cloud federation models benefiting from a federation managers to inform these trade-offs and to absorb much of the cost through memberships with shared standards, operation and maintenance schemes.

4.8 Summary

In this chapter, we have introduced a novel, dynamic and adaptive design diversity approach for web services selection in scaling up scenario using portfolio thinking. We have viewed the cloud as a marketplace for trading instances of web services, which cloud-based applications can explore, trade and use as substitutable and composable entities. In particular, we have used a portfolio-based optimisation to improve the throughput fluctuation rate by diversifying the selection and consequently the allocation of traded instances of web services from multiple providers. Unlike the reviewed classical design diversity solutions that share the assumption of uncorrelated failures, our portfolio-based design diversity is correlation-sensitive; it explicitly links the distribution of resources to risks and accounts for correlation between different providers.

We have reported on four experiments to: (1) test for the approach effectiveness in minimizing the risk of throughput fluctuation; (2) simulate the dynamic and adaptive behaviour of the approach in responding to changes in the market conditions and risk; (3) evaluate the sensitivity of the allocation decisions to risk and its correlation with other candidates and (4) evaluate the scalability of the approach and its ramifications on risk reduction under extreme scenarios. Our approach presents an efficient and effective solution for adaptively investing in diversity while reducing the risk of throughput fluctuation, as demonstrated in our evaluation. To the best of our knowledge, neither portfolio-based design diversity nor dynamic adaptive systems have addressed the problem that we explore in this chapter. The combination of portfolio thinking with web instance diversification is a promising approach for dynamically and adaptively improving QoS and reducing risk of throughput fluctuation.

This chapter has considered a simple scenario where an application is satisfied by a single type of cloud services. However, we have not discussed a more complex scenario where the application requires the cooperation of multiple interconnected cloud services to satisfy their requirement. Dealing with such complex scenario will be addressed in chapter 5.

CHAPTER 5

PORTFOLIO-BASED CLOUD SERVICES COMPOSITION : *IMPROVING PERFORMANCE STABILITY IN A CLOUD SERVICE COMPOSITION*

5.1 Introduction

As we discussed earlier, the increasing number of services available in the cloud market make them plausible and attractive for building Cloud Service Compositions (CSC). However, performance fluctuation is common in the cloud environment due to the changes in the supply and the demand of the shared computational infrastructure and resources. Candidate compositions are vulnerable to such instability.

In chapter 4, we have considered a simple scenario where an application is satisfied by a single type of cloud services. However, we have not discussed a more complex scenario where the application requires the cooperation of multiple interconnected cloud services to satisfy their requirement. we refer to this complex scenario as Cloud service composition CSC. In this chapter, we propose a novel approach to improve performance stability by leveraging on the principles of design diversity in the complex scenarios that target the problem of CSC.

We present a self-adaptive approach that leverages the principle of Modern Portfolio Theory to construct a diversified composition of candidate services. The self-adaptive approach makes an explicit trade-off between the cost and benefit of performing changes to the CSC. We use a hypothetical simulation and a prototype of the system to illustrate the applicability of the approach. Controlled experiments are used to (1) test the approach effectiveness in improving the performance stability of the CSC; (2) analyse the performance of the approach under multiple correlation settings and (3) evaluate the effectiveness of the self-adaptive mechanism in dynamic market.

5.2 Effective Diversification in the Context of Cloud Service Composition

A web services composition is typically concerned with selecting combination of functionalities provided by outsourced services usually denoted as candidate services to satisfy users requests. The selected set of services is often referred to as a composite service. With the increasing popularity of cloud computing, it is expected to find several composite services that are able to provide the same functionality requested by the user. In addition, web services that are hosted on the cloud are vulnerable to performance fluctuations due to changes in demand of the shared resources. As a consequence, candidate CSC becomes vulnerable to such performance fluctuation.

In this section, we present a portfolio-based composition algorithm that aims to reduce performance fluctuation of the CSC. Table 5.1 illustrates the correspondence of portfolio selection problem to both the financial stocks and the CSC selection problem.

Previous research focused on selecting an optimal composition of services among a set of compatible candidates based on constraints on the Quality of Service (QoS) i.e. [55], [74] , [9], [10] [78] and [14]. However, they have not explicitly considered the fluctuation of performance in their models. Contrary to the existing methods, the objective of our approach is to minimise the performance fluctuation through diversity. It attempts to secure an optimal CSC by constructing a diversified portfolio of candidate services.

In the context of CSC, diversification can be achieved by selecting candidate cloud services that share a minimum level of correlation among their performances. In the case of negative correlation, if high demand would affect one candidate service, there is a great chance that it will not affect the performance of the other candidate services in the CSC. We intend to show, that by composing services with low-performance correlation, can result in better performance stability of the composition.

Table 5.1: Correspondence of the Portfolio Selection Problem to Both Financial Stock Selection and the CSC Problem.

	Financial Stock Selection	***Cloud Services Composition(CSC)***
Expected Return	The mean return of the stock.	The mean response time of the of candidate services.
Risk Measurement	The standard deviation of Historical stock price.	The standard deviation of historical response time of the candidate services.
Correlation	Relationship between prices of different stocks.	Relationship between response times of different candidate services.
Positive Correlation	A same direction relationship between stock prices. If a stock price decreases, the other stock is expected to have a decrease in its price at the same time.	A same direction relationship between candidate services in terms of response time. If a candidate service suffers from low performance, the other candidate service is expected to have low performance at the same time.
Negative Correlation	An opposite direction relationship between stock prices moves in the opposite direction. If a stock price increases, the other stock is expected to have a decrease in its price at the same time.	An opposite direction relationship between candidate services in terms of response time. If a candidate service suffers from low performance, the other candidate service is expected have a high performance at the same time.
Objective of Portfolio Theory	Construct a portfolio of stocks that achieves minimum risk (low fluctuation) for a given needed level of return.	Construct portfolio of interconnected candidate services that achieve a minimum fluctuation of response time considering a set of functional and QoS requirements.

5.3 Assumption for the Cloud Service Composition

A Cloud services composition problem has similarities with constructing portfolio of multiple investment assets as it is shown by table 5.1. However, when portfolio theory is used to support the CSC, few assumptions are need to be taken into account:

- The cloud-based application is a risk averse and target to select a set of candidate services that help to reduce response time fluctuation of the CSC given a set of QoS constraints.
- The expected return E_i of investing in a candidate service CS_i is equal to the mean response time. The mean response time for each candidate service can be calculated based on the historical records.
- The response time fluctuation rate FR_i of candidate web service CS_i is calculated as the standard deviation of response time. The standard deviation of response time can be calculated based on historical records of that candidate service.
- The weight of investment w_i for each candidate web service CS_i is equal to the mean response time of the candidate service divided by the mean response time of the CSC.
- For two candidate services CS_i and CS_j, P_{ij} is the performance correlation of these services and can be calculated using one of the statistical models (e.g. [122]) based on historical response time of CS_i and CS_j.

5.4 Model for Portfolio-Based Cloud Service Composition

Taking into account the above assumptions, we can apply the portfolio theory, where a CSC is an investor, searching to invest in a set of candidate services compatible with abstract services in the workflow. The goal of the CSC is to select an optimal set of candidate services that help to achieve stable performance (measured in response time of the CSC). The candidate web services offered in the cloud represent assets. Previous literature has considered 1 to 9 QoS parameters as optimisation constraints for the web services composition problem, where three being the norm [126]. For simplicity of exposition, we look at responses time, price and security as three dimensions of QoS to demonstrate our approach.

1. *Response Time*(RT_i): measures the delay in seconds between the moment the web service is requested till the moment a reply is received.

2. *Security*(Se_i): represents the level of security of the candidate services. It varies from 1 to 5, where 1 indicates weak security and 5 indicates high security level.

3. *Price*(C_i): is the amount of money that has to be paid to lease the candidate services CS_i.

We assume that prices and security level are fixed for each candidate services. On the other hand, response time of web services hosted in the cloud tends to fluctuate [8]. Each candidate service CS_i response time will be modelled by mean response time RT_i, fluctuation rate of response time FR_i and correlations P_{ij} with the other candidate services. Based on these values, we can decide which candidate services are to be selected to construct a portfolio of service composition. The objective is a set of candidate services that minimizes FR_p , where FR_p is the fluctuation rate of response time of CSC, shown in equation 5.1. The minimization should also satisfy constraints on the selected composition:

1. The composition mean responses time should not exceed RT_{max}.

2. Total price of all the selected services should be less than or equal to max Price C_{max}.

3. Each candidate service in the composition should exceed the minimum security level Se_{min}.

These constraints are represented by equations 5.2, 5.3 and 5.4. Table 5.2 presents a brief description of the variables.

$$\min(FR_p) = \min \sqrt{\sum_{i=1}^{m} w_i^2\ FR_i^2 + \sum_{i=1}^{m}\sum_{j\neq i}^{m} w_i w_j\ FR_i FR_j\ p_{ij}} \quad (5.1)$$

Where

$$w_i = RT_i/RT_p$$

Subject to

$$RT_{max} \geq RT_p = \sum_{i=1}^{m} RT_i \quad (5.2)$$

$$C_{max} \geq C_p = \sum_{i=1}^{m} C_i \quad (5.3)$$

$$\forall\ CS_i \in CSC:\ Se_{min} < Se_i \quad (5.4)$$

Table 5.2: Variables Description

Variable	*Description*
FR_p	Fluctuation Rate of response time of CSC.
w_i	Weight of investment w_i of candidate service CS_i.
RT_i	Mean response time of candidate service CS_i.
FR_i	Fluctuation Rate of response time of candidate service CS_i.
FR_j	Fluctuation Rate of response time of candidate service CS_j.
p_{ij}	Correlation of response time between candidate i and candidate j.
RT_{max}	Maximum mean response time of service composition accepted by the user.
RT_p	Mean response time of service composition CSC_i.
C_p	Cost of allocating service composition CSC_i.
Se_i	The level of security of the candidate services CS_i.
Se_{Min}	Minimum accepted level of composition security accepted by the user.
C_{max}	Maximum Cost of service that the buyer is willing to pay.
C_i	Cost of allocating candidate services CS_i.
m	The number of abstract services in the web services composition

5.5 Self-Adaptation Mechanism for Portfolio-Based Cloud Services Composition

Self-adaptation is a key requirement that needs to be considered when developing CSC approaches. It enables the CSC to react to the changes in the dynamic cloud environment and maintains a satisfying solution. Several factors can trigger the need for adaptation of web services composition.

In this section, we present a decentralized self-adaptive strategy for realising and implementing the portfolio-based CSC in highly dynamic market settings. First, the self-adaptive mechanism evaluates the currently allocated portfolio fluctuation rate $FR_{CSCcurrent}$. After that, it will evaluate the fluctuation rate $FR_{CSCoptimum}$ of the optimum portfolio fluctuation rate that we could allocate based on the new market state. I_c represents the level of improvement that the system can gain by allocating the new optimum CSC. In other words, I_c resembles the potential improvement in fluctuation rate, if we replace the current CSC with a new optimum CSC. I_c is calculated according to equation 5.5. A positive number will indicate an improvement in the fluctuation rate of CSC.

$$I_c = FR_{CSCcurrent} - FR_{CSCoptimum} \qquad (5.5)$$

A buyer will set I_m, which denotes the minimum accepted improvement level in the new CSC, to trigger the adaptation and replace the current portfolio with a new optimum. This will make the adaptation of the new optimum CSC subject to the satisfaction of the following condition. The level of improvement I_c in the new optimum CSC is greater than the minimum level of accepted improvement I_m.

A cloud-based application is expected to perform adaptations when there is a change in the market prices, QoS or fluctuation rates. The basic steps of our self-adaptive portfolio-based CSC are listed as in the algorithm presented in table 5.3.

The algorithm in the first adaptation cycle asks the buyer to set the minimum accepted level of security, Se_{min} , the maximum response time that (s)he can tolerate RT_{max} and the maximum price that (s)he is willing to pay, C_{max} (see line 1-4). Then the algorithm identifies the set of service compositions, $CSC_{candidate}$, which offer the functionalities required by the user and satisfy the global QoS constraints presented by equation 5.2, 5.3 and 5.4 (see line 5-10).

After that, the buyer agent will access the *KnowledgeBase*, which is maintained by the market regulator, to retrieve fluctuation rate, FR_i , QoS values and correlation matrix P associated with each service in candidate compositions. For each candidate composition, the buyer agent will calculate investment weight for each candidate services and the composition fluctuation rate FR_p using equation 5.1 (see line 12-15).

Next, the CSC that provides the minimum fluctuation rate will be selected as optimum $FR_{CSCoptimum}$ in the market. The algorithm then calculates the fluctuation rate of the currently allocated CSC (see line 16-17). Finally, the algorithm calculates the level of improvements in fluctuation rate I_c using equations 5.5 and if I_c exceeds the minimum accepted improvement level I_m, an adaptation will be triggered to allocate new set of web services as recommend by the new optimum CSC (see line 18-22).

Table 5.3: Self-Adaptive Portfolio-Based CSC Optimization Algorithm.

```
Input:
$CSC_{available.}$: list of the avalible compiation new market condition, $\mathbf{I_m}$ :the minimum accepted improvement level
Output:
The adaptation decision: should we adopt a new portfolio or keep the currently allocated portfolio unchanged?

Begin
1: if (first adaptation cycle )
2: then
3: set QoS constraints by defining $\mathbf{RT_{max}}$, $\mathbf{C_{max}}$ $and$ $\mathbf{Se_{min}}$
4: end if
5: for $CSC_{i.} \epsilon$ $CSC_{available.}$ in the market
6: calculate aggregated $C_i$ and $RT_i$ for $CSC_{i.}$
7: if $CSC_{i.}$ satisfy all constraints presented in equations 5.2,5.3 and 5.4.
8: add $CSC_{i.i.}$ to set of alternative candidate services composition $CSC_{\text{candidate.}}$
9: end if
10: end for
11: for candidate services composition $CSC_{i.} \epsilon$ $CSC_{\text{candidate.}}$
12: get correlation $\mathbf{p_{ij}}$ and fluctuation rate $FR_{i.}$ For each service in the composition from the KnowledgeBase
13: Calculate weight of investment $w_i$ for candidate service $CS_{i.}$ in the composition $CSC_{i.}$ Using equation 5.2
14: Calculate fluctuation rate of the CSC $\mathrm{FR_p}$ using equation 5.1.
15: end for
16: $FR_{CSCoptimum} = MinFluctuationRate\ (CSC_{\text{candidate.}})$
17: Calculate the Fluctuation Rate of currently allocated CSC using equation 5.1
18: Calculate the potential improvement in risk $I_c$ using equation 5.5.
19: if ( $I_c > I_m$ )
20: then submit the new optimum CSC
21: else keep the currently allocated CSC
22: end if
End
```

5.6 Evaluation

We demonstrated how portfolio-based composition can improve the stability of the CSC response time by simulating a hypothetical cloud market. We compared the performance stability achieved by portfolio-based composition with the following algorithms:

1. *Non-diverse composition* that uses auction-based mechanism as in [126] and [36] to allocate all the required candidate services from a single cloud provider that have the lowest fluctuation rate of throughput.

2. *Random Geographical diverse composition* inspired by [22] and [23] where diversity is neither linked to stability (fluctuation rate) nor correlation between candidate services. Instead, diversity is implemented by selecting random composition of candidate services from multiple cloud providers located in different geographical locations.

3. *Non-correlated portfolio-based composition* proposed in [127] where diversity is linked to the stability (fluctuation rate) of candidate services from multiple cloud providers. However, it does not consider the correlation between the candidate services.

In addition to simulation, we have implemented a prototype where the whole technique is integrated with CloudSim [128] computing environment. CloudSim supports both system and behaviour modelling of Cloud computing components such as data centres, physical machines, virtual machines and CloudLet web services. In the prototype, we extended generic implementation of CloudSim to implement the portfolio-based CSC. We evaluate the performance stability of a CSC application built by portfolio-based composition and compare it with CSC application built by Random Geographical diverse composition and Non-Correlated portfolio-based composition.

5.6.1 Simulation Settings

Mytrip.com, which is a CSC application that provides an on-line service for booking travel packages, is used here as an example. We have considered a typical scenario involving services composition of three AbstractService: FlightBooking, CarBooking and Hotel-Booking services. Each AbstractServices will have four candidate services for simplicity of demonstration; nevertheless, the argument and technique can be applied to more than four candidate services. Furthermore, we have used up to 50 candidate services to stress test the mechanism for scaling. The choice of 50 to test scale is justified as a systematic literature review of dynamic services composition has shown that the literature has typically used 20 candidate services [126]. To simulate a realistic performance stability setting of the cloud, we need the following data for each candidate service: 1) mean response time 2) degree of stability measured by fluctuation rate of response time of the candidate services, and 3) response time correlation between candidate services. We have assumed that the mean response time will vary between 10 and 18 seconds.

In terms of stability, an analysis of Amazon EC2 performance [8] shows that the fluctuation rate of some Amazon EC2 instances reached 71%. For that, we assume that the candidate service fluctuation rate of response time vary between 30% and 71%. Moreover, the number of the considered QoS constraints is set to 3 (cost, response time and security level). Cost and security level of the candidate services are randomly created. In addition, correlation will vary between 1 and -1. Each simulation reported, unless otherwise specified, was run with the following default parameters as shown in table 5.4.

In this evaluation, four main sets of experiments were conducted with the following descriptions as rationale for our choice:

1. The first set of experiments are designed to evaluate the effectiveness of the portfolio-based composition in improving the response time stability of CSC in comparison to Non-diverse, Random Geographical diverse composition and Non-correlated portfolio-based composition.

Table 5.4: The Default Simulation Parameters

Simulation Parameters	***Value***
Workflow size (number of Abstract Services in composition)	3
Candidate Services per Abstract Service	4
QoS attributes	3
Candidate Services mean response time	10 to18 Seconds
Candidate Services fluctuation rate	30% to 71%
Correlation of response time between the Candidate Services	1 to -1

2. The second set of experiments evaluates the effectiveness of the algorithms under three correlation settings for the environment positive, negative and weak correlations.

3. The third set of experiments is designed to evaluate the effectiveness and the stability of the proposed self-adaptive mechanism under different market setting.

5.6.2 The Effectiveness of the Portfolio-Based Composition

The goal of this experiment is to evaluate the performance stability of a CSC application built by portfolio-based composition and compare it with CSC application built by Non-diverse, Random Geographical diverse composition and Non-Correlated portfolio-based composition. The performance stability of the composition is measured by the fluctuation rate of the response time.

To ensure a fair comparison between the different composition algorithms, we ran nine cases with the same parameters as in table 5.4. Each case is different, as it will have different setting for fluctuation rate, mean and correlation of response time. Fig. 5.1 depicts for each of the nine cases, the fluctuation rate of response time of web composition selected by the Non-diverse, Geographical diverse and composition , Non-correlated and correlated portfolio composition algorithms. From the results in Fig. 5.1, it is clear that

Non-diverse composition has the worst results (highest fluctuation rate) in six out of the nine cases, because it does not implement design diversity but selects all the required candidate services from a single cloud provider. That means a high positive correlation between the candidate services as high demand on that cloud provider will affect all candidate services of the composition at the same time.

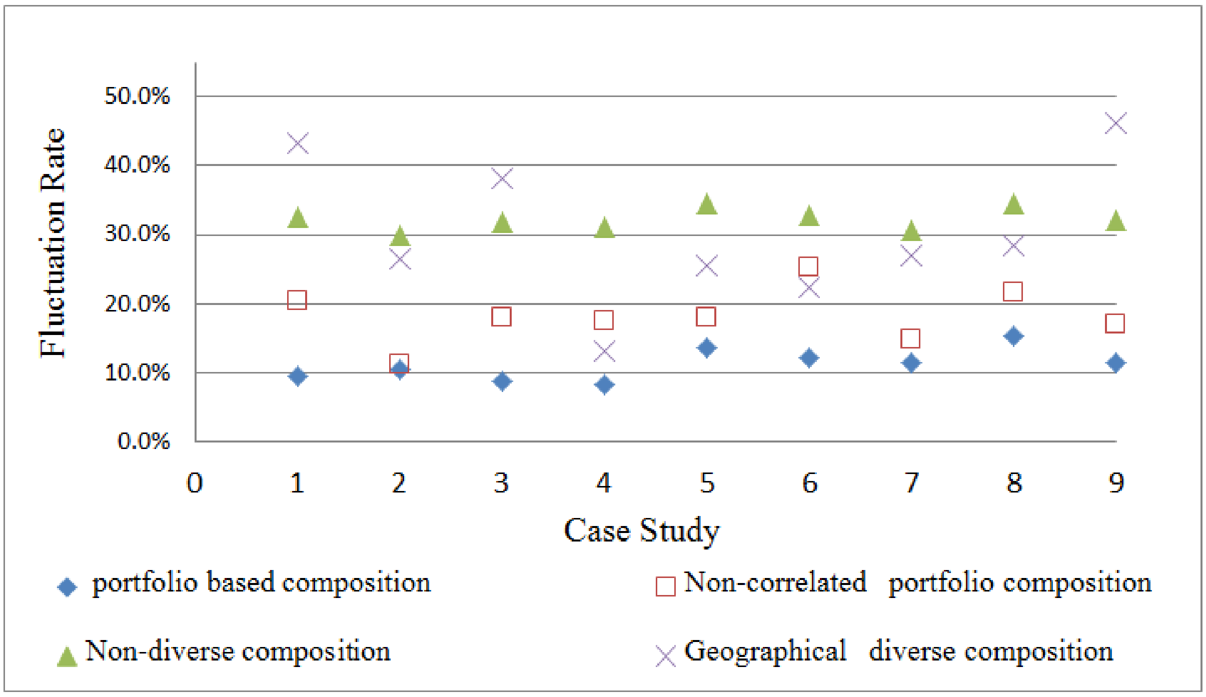


Figure 5.1: Fluctuation Rate of CSC for the Four Considered Algorithms.

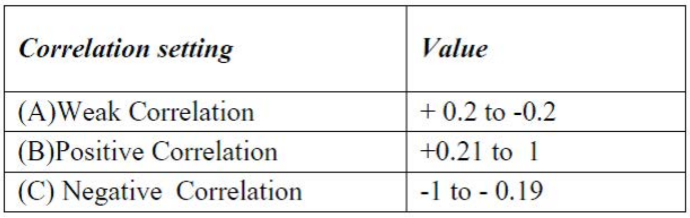
Random Geographical diverse composition gives the second worst results (second highest fluctuation rate). This is because the diversity is randomly created and not linked to neither the fluctuation rate of the candidate services nor the correlation between them. Non-correlated portfolio emerged as the second best algorithm in seven of the cases studied.

The Non-correlated portfolio composition considers fluctuation rate of candidate services. However, it ignores performance correlation between them. The portfolio-based composition has achieved the best results in all of the nine cases (lowest level of fluctuation rate) because it utilizes the portfolio concept to diversify the selection of candidate services. The diversification is directly linked to both of the fluctuation rate and correlation between the candidate services in the composition.

5.6.3 Effectiveness of the Composition Algorithms Under Positive, Negative and Weak Correlations

The goal of this experiment is to investigate the effect of the different correlation settings on the quality of the solution for each of the four algorithms. As was in experiment 1, we ran 9 cases with the same parameter setting of Table 5.4 except for the correlation setting. Instead, we have used three settings for correlation such as Weak, Positive and Negative correlation as shown in Table 5.5.

Table 5.5: Simulation Correlation Settings



Correlation setting	*Value*
(A)Weak Correlation	+ 0.2 to -0.2
(B)Positive Correlation	+0.21 to 1
(C) Negative Correlation	-1 to - 0.19

The result in Fig.5.2 shows that the ranking of the composition of each of the four algorithms remains as in previous experiment. The lowest fluctuation rate was achieved by the portfolio-based composition followed by non-correlated portfolio with the Random Geographical diversity. The non-diverse composition was ranked as the forth. In terms of the effectiveness of the diversification process, we can see from the results that the quality of the diversification can be affected by the level of correlation between the candidate services. A positive correlation indicates a high dependency between the candidate services and weak correlation indicates a performance independence of the candidate services.

We can measure the effect of diversification by the difference in fluctuation rate between the Non-diverse composition and the portfolio-based composition. In case (A), where we have nine cases with positive correlation setting, the average difference fluctuation rate between the Non-diverse composition and the portfolio-based composition was 6.3%. The difference in fluctuation rate increased to 12.5% for weak correlation and peaked for positive correlation reaching 22.9%.

From the results, we can conclude that the quality of portfolio-based composition is closely related to the correlation between different candidate services. The lower the correlation between the candidate services (avoiding a positive correlation), the greater the fluctuation rate reductions achieved by diversification.

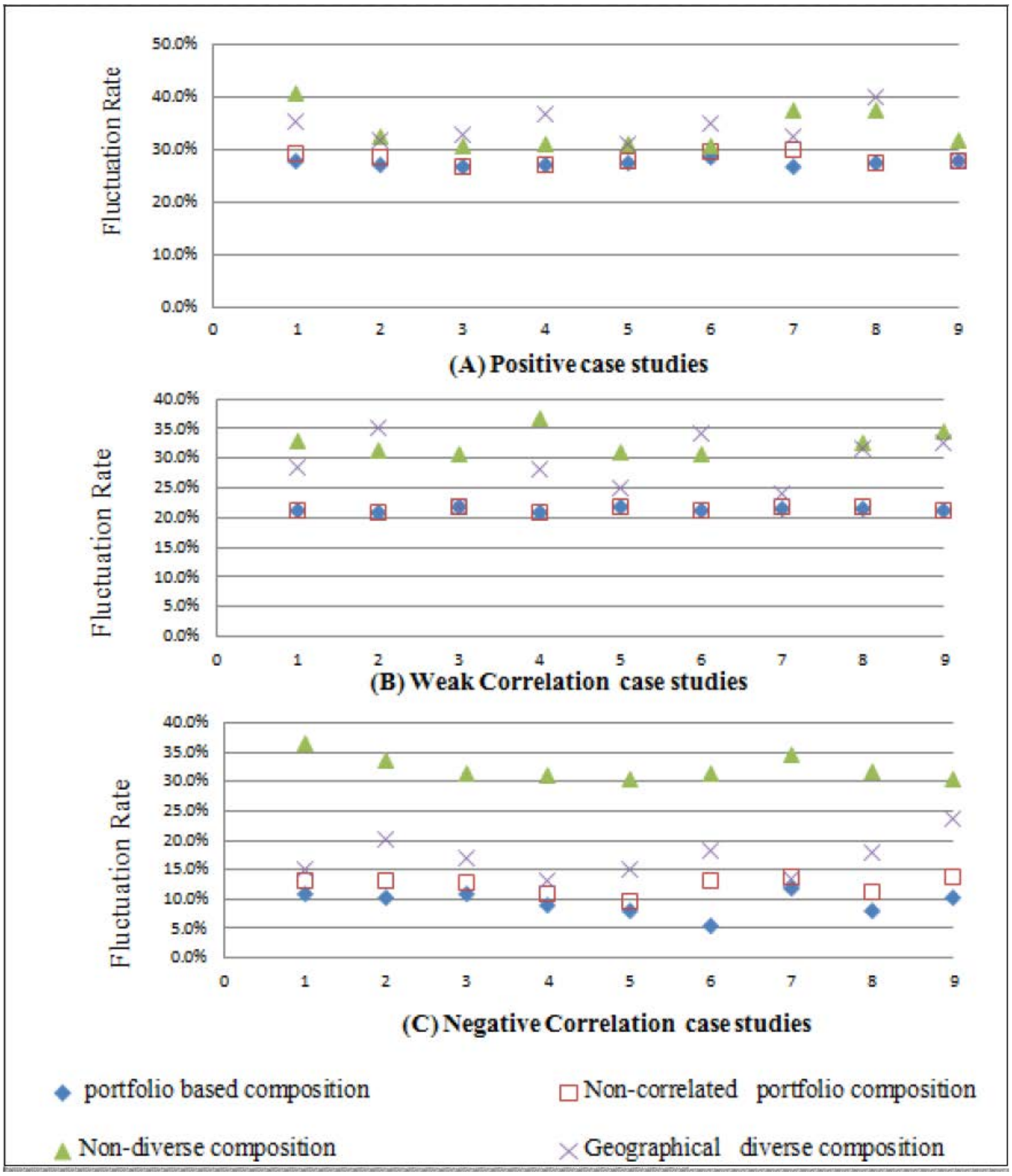
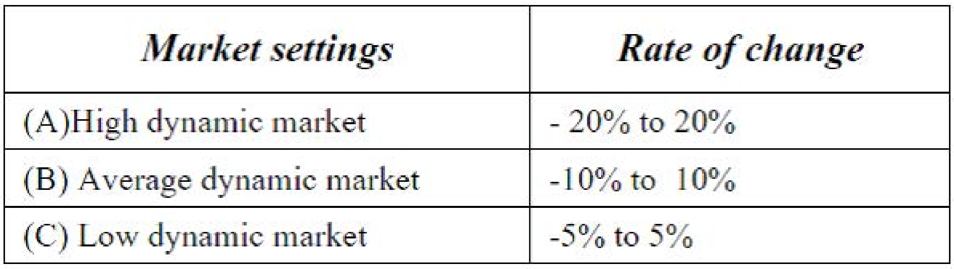


Figure 5.2: Fluctuation rate of CSC in Positive, Weak and Negative correlation.

5.6.4 The Effectiveness and the Stability of the Self-Adaptive Mechanism

In this experiment, we will evaluate the efficiency of the self-adaptive behaviour where the approach should systematically evaluate the fluctuation rate of the allocated CSC and compares it to the optimal traded CSC at the current market condition. It should then dynamically make trade-offs decisions between the benefits of adopting a new optimal CSC, where benefits correspond to the fluctuation reduction versus the cost of frequently changing the services in the adaptation process, as detailed Section 5.5. To test the self-adaptive mechanism, we assume that the QoS and fluctuation rates of web services in the cloud market change over time. In the experiment, we model different dynamics of change by simulating three market settings as in table 5.6. The rest of the simulation parameter are the same as the values presented in table 5.4.

Table 5.6: Simulation Market Settings



Market settings	*Rate of change*
(A)High dynamic market	- 20% to 20%
(B) Average dynamic market	-10% to 10%
(C) Low dynamic market	-5% to 5%

For each market setting, we ran 100 adaptation cycles, where each cycle will cause a change in QoS of the services. After each adaptation cycle, we evaluate how a change in the market condition affects the optimality of previously allocated CSC. As demonstrated by algorithm depicted in table 5.3, the adaptation decision is based on I_m, which is the minimum required improvement level in fluctuation rate to trigger an adaptation to allocate the new optimum CSC. Using different values for I_m can affect both of the quality and frequency of adaptation.

To evaluate the quality and the stability of the self-adaptive mechanism, we compare the performance of four algorithms in term of optimality and number of adaptation required to achieve that level of optimality. The four algorithms are as follow:

1. *Non-adaptive method.* This algorithm will not make any changes to the previously selected CSC.

2. *Unconditional adaptation algorithm.* This algorithm will trigger an adaptation whenever a new optimum CSC emerges in the market. That is because it does not specify any minimum required improvement ($I_m = 0\%$).

3. *A 5% conditional adaptation algorithm.* This algorithm will trigger an adaptation in case a new optimum CSC presents an improvement level greater than 5% (I_m=5%).

4. *A 10% conditional adaptation algorithm.* This algorithm will trigger an adaptation in case a new optimum CSC present an improvement level greater than 10% (I_m =10%).

The change of fluctuation rate of CSC for each of unconditional adaptive, conditional adaptive and non-adaptive algorithms during the 100 adaptation cycle is displayed in Fig. 5.3. Moreover, Fig. 5.4 shows the average fluctuation rate of CSC and the number of adaptations performed by each algorithm. One observation from the results in Fig. 5.4 is that the adaptive algorithms, in general, have outperformed the non-adaptive algorithm because they react to changes in the market by selecting new optimum services for the CSC.

In comparison to the non-adaptive algorithm, the use of unconditional adaptation algorithm has contributed to reductions in fluctuation rate of CSC by 24.6% and 16.6% in high and average dynamic market settings, respectively. However, the number dramatically drops to 2.6% in the market in low dynamics setting. We can say that the benefits of using self-adaptive scheme decrease as we move from high dynamic market to a low dynamic market.

In terms of quality of the adaptive algorithms, we can see in Fig. 5.4 (a) that the unconditional adaptation algorithm have achieved the lowest level of fluctuation rate in comparison to the 5% and the 10% conditional algorithms. On the contrary, the performance superiority of unconditional adaptation algorithm comes with a high cost as shown in Fig. 5.4 (b). The figure shows that the unconditional adaptation algorithm requires a high number of adaptations reaching 80, 56 and 33 for the high, average and low dynamic market, respectively. This is because the unconditional adaptation algorithm will continuously change the selection of the services regardless of the level of improvement gained by that change.

On the other hand, the 5% conditional algorithm requires only 41, 16 and 3 adaptations for the high, average and low dynamic market, respectively. Furthermore, the 10% conditional algorithm requires even a lower number of adaptations which was 13, 5 and 2 for the high, average and low dynamic market, respectively. One can see in Fig. 5.4 (a) that the quality gap between unconditional adaptation and the 5%, 10% conditional adaptation was less than 6% and 3%, respectively.

So, despite the high cost associated with the unconditional adaptation, the quality gap between the unconditional and conditional adaptation algorithms is relatively small. Keeping that in mind, one can argue that using conditional adaptation is more effective as they make an explicit trade-off between the cost and benefit of performing change to CSC.

To sum up, the conditional adaptation algorithm can be customized to cater for different users preferences by using different values for an improvement level I_m. Choosing a large value for I_m will help to limit the number of adaptations, whereas choosing a smaller number will aid in reducing the fluctuation of the selected CSC.

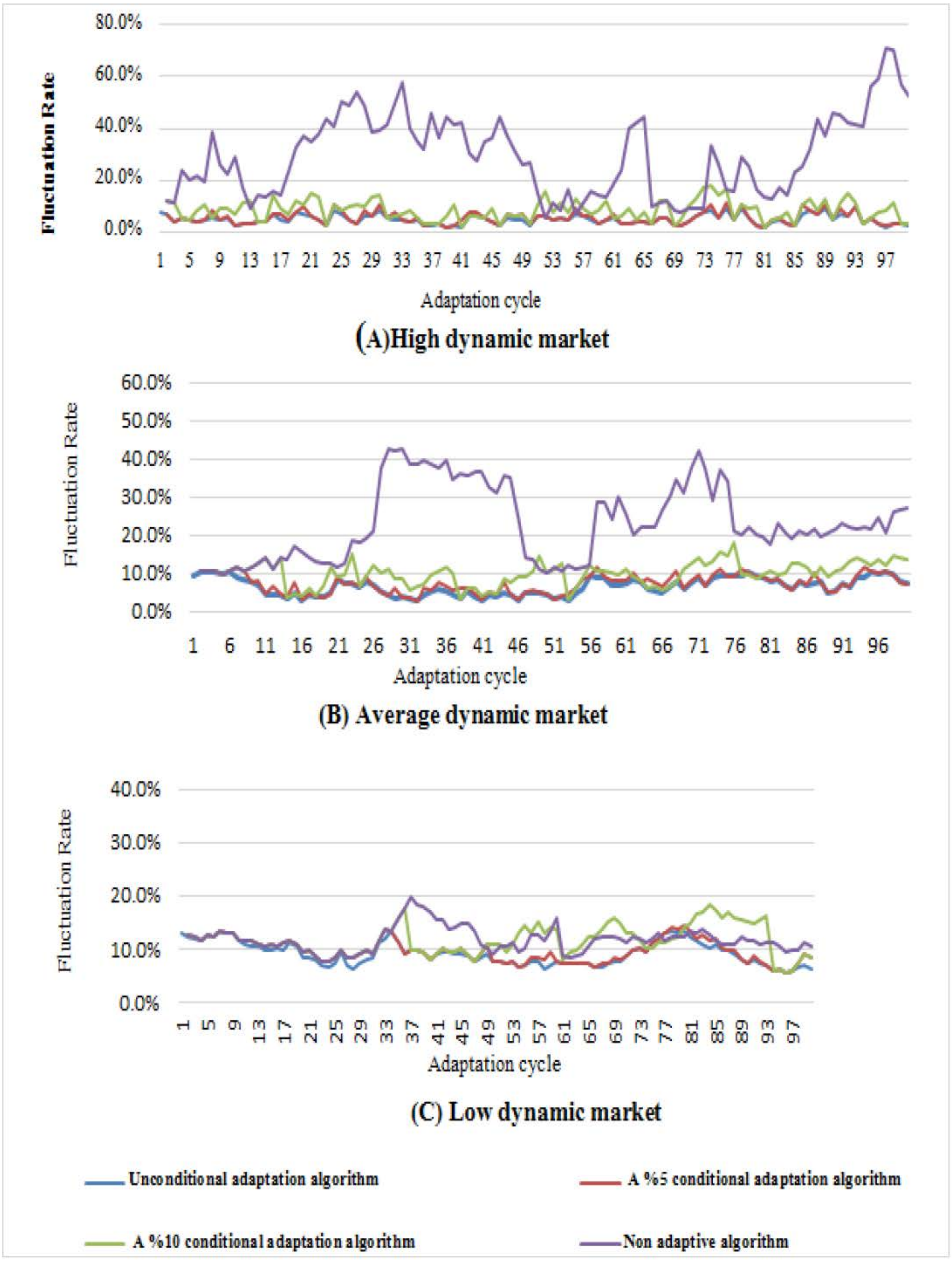


Figure 5.3: Change in Fluctuation Rates of CSC During 100 Adaptation Cycles of Four Different Algorithms in High, Average and Low Dynamic Market Setting.

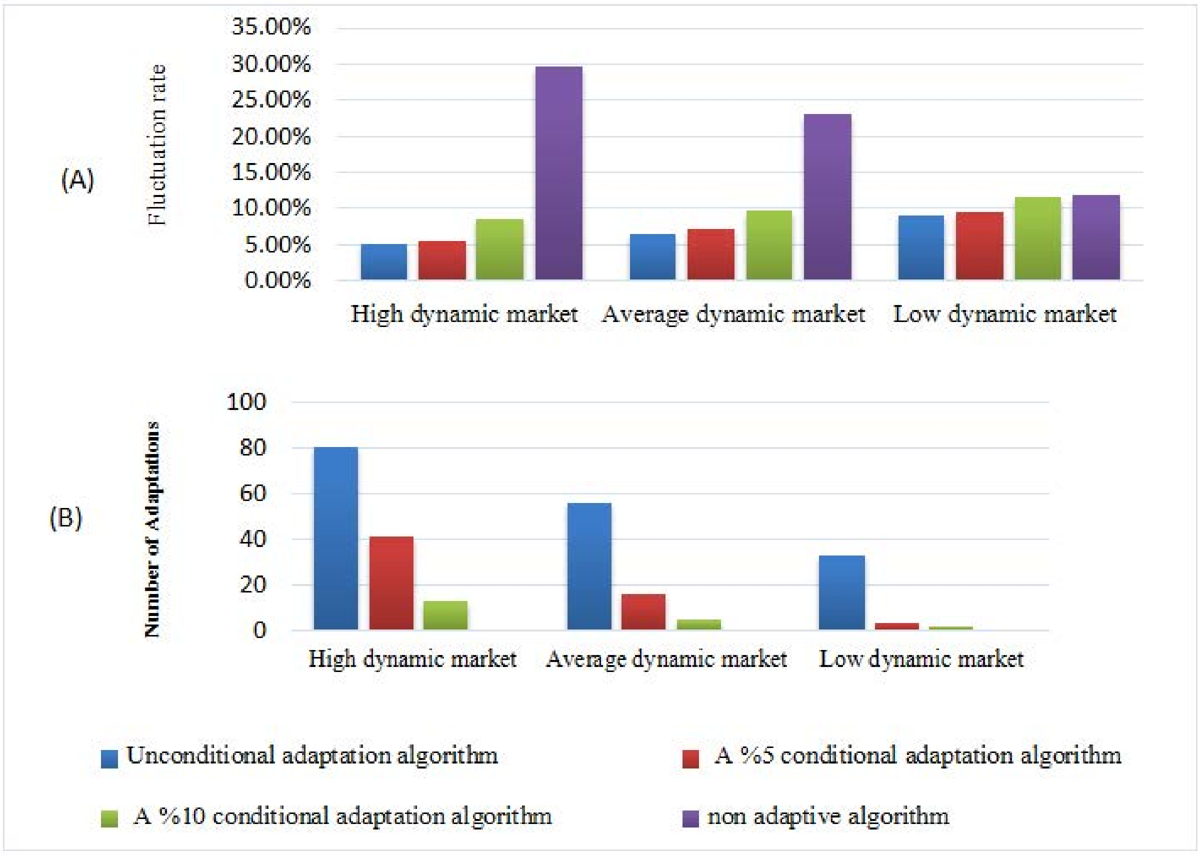


Figure 5.4: Average Fluctuation Rate of CSC And Number of Adaptations of Each of the Algorithms in High, Average and Low Dynamic Market Settings for 100 Adaptation Cycle.

5.6.5 A Prototype Using CloudSim Environment

In the previous experiments, we considered only a simulation of a hypothetical cloud-based market where QoS values are drawn from a normal distribution. To get more realistic settings, we have implemented a prototype where the whole portfolio based CSC is integrated with CloudSim environment [128]. CloudSim is a well known framework that has been used by researchers and practitioners for evaluating their cloud based solution (e.g. [129], [130], [131], [132] and [133]). The details of our experiment are described in the following sections.

CloudSim Environment

CloudSim positions itself as a framework for modelling the behaviour of both cloud applications and infrastructures. It achieves that by providing a fine-grain modelling of both the hardware and software of the cloud environment. Among the modelled hardware entities are: data centres, physical machines and networks, whereas the modelled software entities include: virtual machines, brokers and Cloudlets (cloud web services). When a Cloudlet is submitted by a user, the broker receives the Cloudlet and sends it to a virtual machine, which is hosted on the physical machines at one of the data centres. Upon the completion of each Cloudlet, the broker records the results and the execution time. In this case, the response time of services is not just a random number but is determined by multiple factors such as: the number of concurrent users, the size of the application, the processing power of the virtual machines and the physical machines.

The Effectiveness of The Portfolio-Based Composition Using Cloudsim Prototype.

We consider a scenario where CSC requires the integration of three abstract services (WS_1,WS_2,WS_3). The cloud market offers nine candidate services (three candidates for each abstract service) hosted over nine different data centres.

Each data centre hosts approximately 30 physical machines. Each machine has 10TB of storage and six-core CPU runs 60000 MIPS (Million Instruction per Second). The virtual machines hosting the services are based on an Amazon's small instance *T2.small* (2GB of memory, 1 virtual core CPU). For each candidate service, we simulate a fluctuating number of users that varies from 2000 to 18000 users and run the experiment for 30 times.

Running such a large scale experiment in real cloud environment makes the reproduction of results an extremely difficult and expensive because we need to buy or rent hundreds of physical machines. A suitable alternative is the use of simulation-based approach to run the experiment. For that, we utilize CloudSim which gives us the ability to evaluate our system prior to software development in an environment and gives us the ability to reproduce the results.

The exact number of users for each service and the response time for each service in the 30 runs are presented in appendix B. The summary of the response time of each service is presented as in Fig. 5.5 and the correlation between the services is presented in table 5.7.

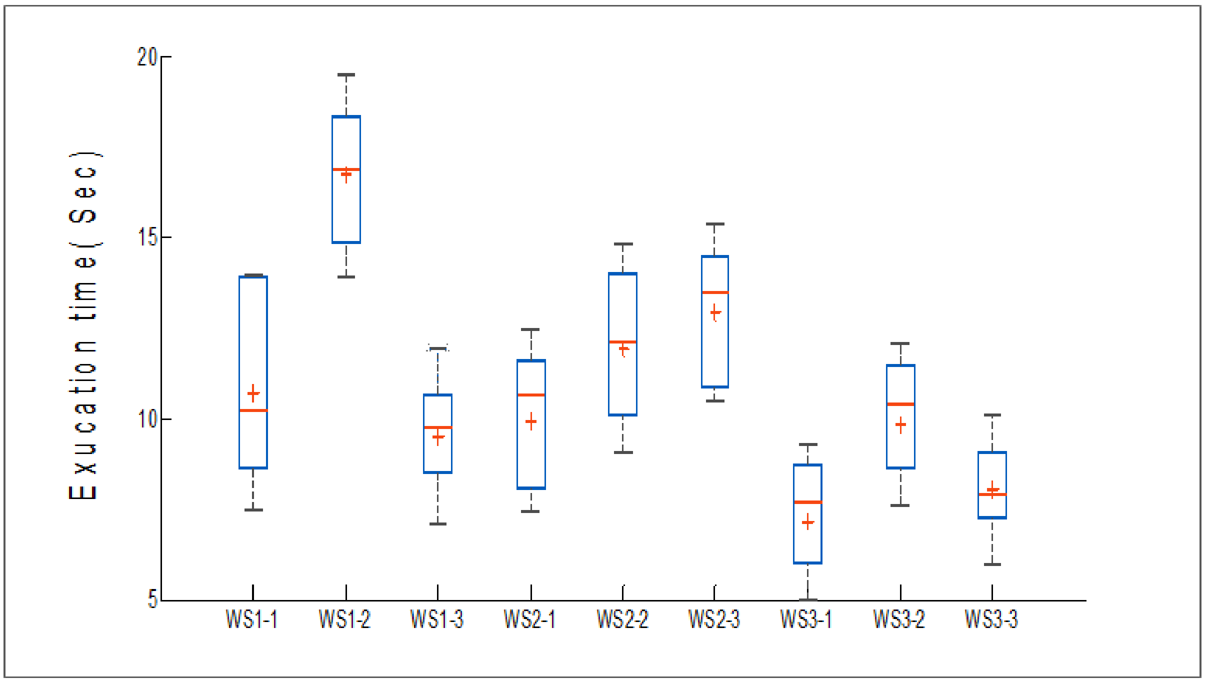
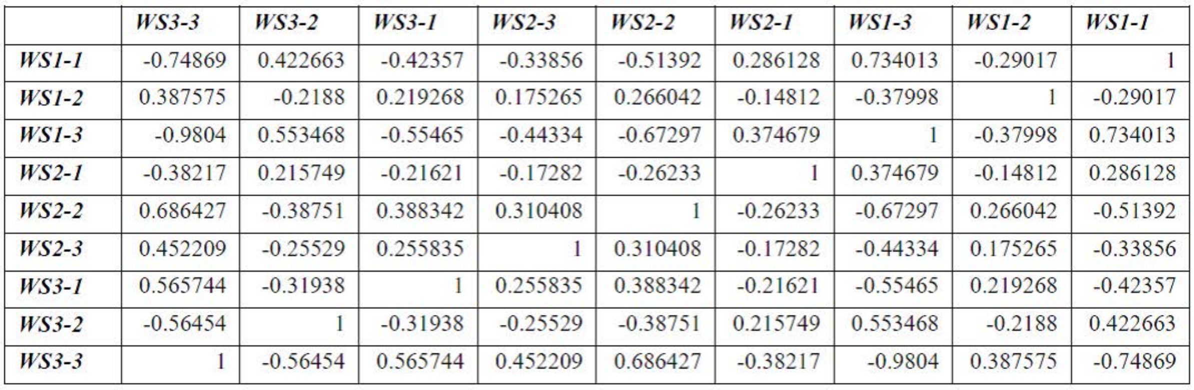


Figure 5.5: Execution time of web services in the market represented by mean and standard deviation (fluctuation rate) over 30 independent runs. The red plus sign represents the mean, and the black edge represents the standard deviation(fluctuation rate).

Table 5.7: The Performance Correlation Matrix of Services in The Cloudsim Market.



	WS3-3	WS3-2	WS3-1	WS2-3	WS2-2	WS2-1	WS1-3	WS1-2	WS1-1
WS1-1	-0.74869	0.422663	-0.42357	-0.33856	-0.51392	0.286128	0.734013	-0.29017	1
WS1-2	0.387575	-0.2188	0.219268	0.175265	0.266042	-0.14812	-0.37998	1	-0.29017
WS1-3	-0.9804	0.553468	-0.55465	-0.44334	-0.67297	0.374679	1	-0.37998	0.734013
WS2-1	-0.38217	0.215749	-0.21621	-0.17282	-0.26233	1	0.374679	-0.14812	0.286128
WS2-2	0.686427	-0.38751	0.388342	0.310408	1	-0.26233	-0.67297	0.266042	-0.51392
WS2-3	0.452209	-0.25529	0.255835	1	0.310408	-0.17282	-0.44334	0.175265	-0.33856
WS3-1	0.565744	-0.31938	1	0.255835	0.388342	-0.21621	-0.55465	0.219268	-0.42357
WS3-2	-0.56454	1	-0.31938	-0.25529	-0.38751	0.215749	0.553468	-0.2188	0.422663
WS3-3	1	-0.56454	0.565744	0.452209	0.686427	-0.38217	-0.9804	0.387575	-0.74869

We compared the portfolio-based composition with Random Geographical diverse composition inspired by [106] and Non-correlated portfolio-based composition proposed in [127] where diversity is linked to the stability (fluctuation rate) of the candidate services from multiple cloud providers. The portfolio-based allocation approach formed a composition using services (WS_{1-3},WS_{2-3},WS_{3-3}). This composition have average response time of 30.5 seconds and a 23% risk of response fluctuation. This is because the services forming the composition share a low correlation between their performances.

The uncorrelated portfolio-based composition formed a composition using services (WS_{1-2},WS_{2-3} ,WS_{3-2}) which have average response time of 39 seconds and 38% risk of response time fluctuation. On the other hand, Random Geographical diverse composition have selected (WS_{1-2} , WS_{2-2} , WS_{3-2}) which have average response time of 35 seconds with 56% risk of response fluctuation. These results are displayed in Fig. 5.6. We can say that the prototype shows consistent results with the findings of the simulated experiments where the portfolio-based CSC outperforms the other composition algorithms in terms of the quality of selection (lowest fluctuation rate).

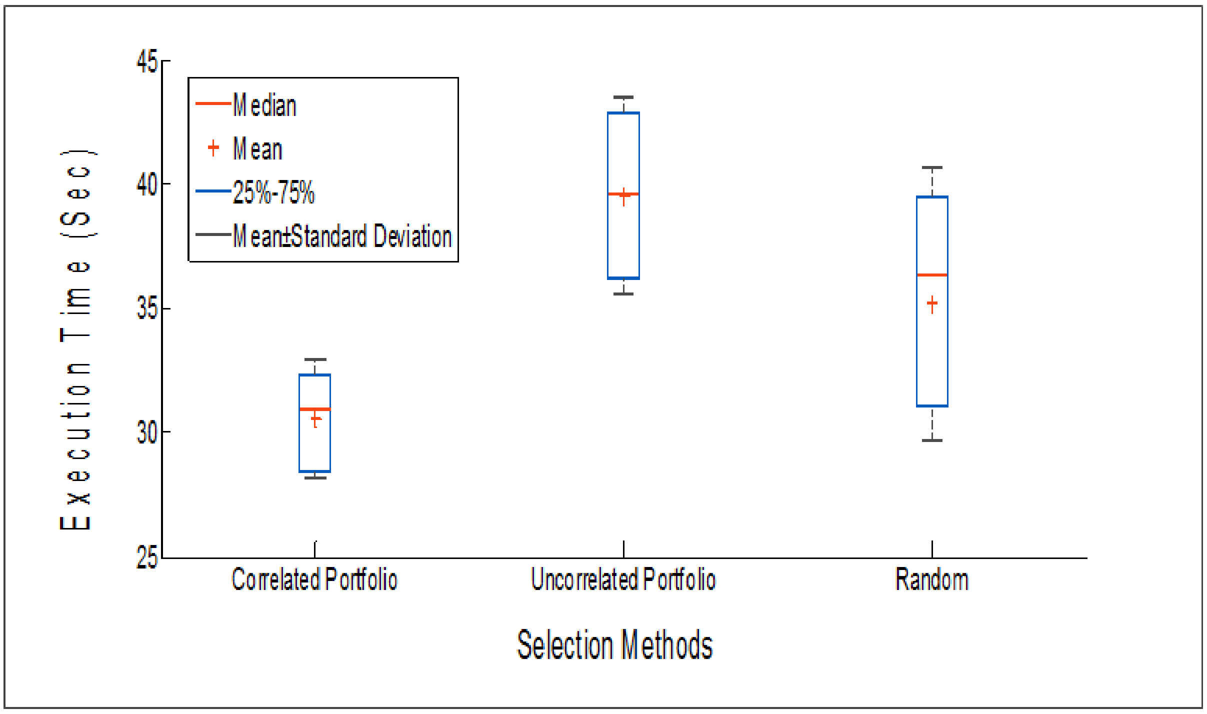


Figure 5.6: Execution time of services composition selected by correlated portfolio, random geographical diversity (random geographical composition) and uncorrelated portfolio diversity. Execution time is represented by mean and standard deviation (fluctuation rate) over 30 independent runs. The red plus sign represents the mean, and the black edge represents the standard deviation (fluctuation rate).

5.7 Conclusion

In this chapter, we have presented a novel self-adaptive approach to assist a CSC in the process of selecting set of well-diversified candidate services that helps in providing more stable performance. In particular, we have used a portfolio-based optimisation technique to improve performance stability by diversifying the selection of candidate services that share low correlation between them. Unlike the reviewed classical design diversity solutions that share the assumption of uncorrelated failures, our portfolio-based design diversity is correlation-sensitive; it explicitly links the distribution of resources to performance fluctuation and accounts for correlation.

The evaluation shows that in comparison to the Non-diverse, Geographical diverse and Non-correlated composition algorithms, our portfolio-based approach achieves minimum performance fluctuation for CSC. Moreover, it shows that using conditional adaptation algorithm is more effective as they make an explicit trade-off between the cost and benefit of performing changes to CSC.

However, in this chapter we did not cover scalability analysis services composition which is known to be a NP-hard problem [9]. An important challenge is how our services composition approaches scale with respect to changes in the market and the environment. Scalability is also concerned with how the mechanism caters for changes that relate to the number of candidate services, workflow size, number of QoS under consideration, constraints among the others.

To address this challenge, in the next chapter we will use an approach to make the scalability dimensions for the CSC problem explicit and transparent to the mechanism: we perform a systematic elaboration of scalability requirements through goal-obstacle analysis [35] to identify the necessary dimensions, which can influence the behaviour of the mechanism. Then, we present scalability evaluation for our portfolio-based composition algorithm.

CHAPTER 6

SYSTEMATIC ELABORATION OF SCALABILITY REQUIREMENTS USING SCALABILITY GOAL-OBSTACLE ANALYSIS: *THE CLOUD SERVICES COMPOSITION*

6.1 Introduction

In the last chapter, we demonstrate the effectiveness and self adaptivity of our portfolio based CSC. However, the scalability of the algorithm was not evaluated. This is important considering the large scale and the dynamic nature of the cloud-based market, which can lead to exponential increase in the problem search space. One would expect a variety of dimensions to be considered in evaluating the scaling of dynamic web services composition, such as the workflow size, number of QoS constraints, number of recomposition requests and so forth.

This will make the identification of the system design goals, and the functional and non-functional requirements that are relevant to the scalability analysis of software a complicated process. To overcome the complexity of scalability analysis, Duboc et al. [35] proposed a systematic approach for elaborating the scalability requirements, which extends KAOS goal-oriented model [134].

In this chapter, we present a background section that gives an overview of the necessary concepts to understand the scalability analysis. These concepts include KAOS goal-oriented modelling and goal obstacle analysis. Then, we show the basic steps of the scalability goal obstacle analysis which includes the identification, assessment and revolving of scalability obstacle. We then present the goal modelling and the scalability goals of the portfolio-based composition. Finally, the evaluation presents a set of experiments that are designed to evaluate the scalability of the portfolio-based composition. It reports on the sensitivity of time needed to find an optimal CSC composition scale in relation to multiple dimensions of the CSC problem.

6.2 Background

6.2.1 A Brief Introduction to KAOS Goal-Oriented Modelling

One of the most influential measurements of software system success is the degree to which it complies with its purpose. For that reason, identifying the purpose of software should be one of the main activities of the software development life cycle. There is a consensus on the fact that incomplete, vague, or inconsistent requirements can have a negative effect on both QoS and user satisfaction level [135].

Traditional requirements engineering approaches, such as structured analysis [136] and object-oriented modelling [137] failed to: (1) cope with the increasing complexity of the new systems and (2) consider the non-functional requirements. To overcome these limitations, KAOS which stands for *Keep All Objectives Satisfied* have been presented as a framework for implementing Goal-Oriented Requirements Engineering [138]. KAOS views software systems as a composition that consists of the software and its hosting environment as a collection of agents aiming to achieve different system goals as objectives [134].

In KAOS, agents are active components that may represent humans, hardware devices or software that are able to monitor and modify the system in order to satisfy the set of goals [35]. Goals may range from high level goals that require the cooperation of multiple agents (such as *Achieve[Services composition is composed of selected concrete services that share minimum correlation between their performance]* to low-level goals that require fewer agents to achieve such as *Achieve[buyer informed if a concrete service for a given QoS and budget constraints does not exist]*. Goals also involve different types of requirements: (1) functional requirements, such as task the system required to do, and (2) non-functional requirements, such as performance, security, scalability, and so forth [134].

The goal model depicts all the system goals, how they contribute to each other and who is responsible for achieving them. It represents the goals in a hierarchical graph with *AND/OR* refinement links. Both of the *AND/OR* refinement links are used to relate a parent goal with a set of sub-goals. In the case of *AND* refinement link, the satisfaction of the parent goal requires the satisfaction of all the offspring goals linked to it. On the other hand, *OR* refinement indicates that an offspring goal represents an alternative way to satisfy the parent goal [139].

Elaborating system's goals can be performed using top-down fashion by asking *HOW* questions to decompose goals into new sub-goals. In addition, a bottom-up fashion can be used for elaborating system's goals by asking *WHY* questions to find new parent goals. The elaboration of the goals stops when each sub-goal in the system is assigned to a single agent which means that this agent has the ability to monitor and control the variables associated with the goal [35]. In KAOS, right leaning parallelograms are used to represent system goals, whereas left leaning parallelograms are obstacles. Hexagon are used to represent agent and arrows connected by a circle are used to represent the *AND* refinements link [140].

Figure 6.1 shows a portion of the elaborated goal modelling for portfolio-based service composition. The goal *Achieve[Dynamic composition of services with minimum QoS performance fluctuation]* is the main goal of portfolio based web service composition. In order to achieve this goal, the system must (a) keep information about open requests for cloud services compositions and buyer preferences which is the responsibility of the buyer agent, (b) compose applications by selecting set of diversified services for the abstract services in the application workflow, and (c) maintain an updated information about concrete services in the market. In Fig. 6.1, this relation is represented in the model by using an *AND*, refining the top goal into three sub goals, (a) *Maintain[Information about services composition request and buyer preference]*, (b) *Achieve [effective diversification of service composition]* and (c) *Maintain[an updated information about concrete services price and QoS in the market]*, respectively. The first goal is a leaf goal, assigned to the Buyer agent.

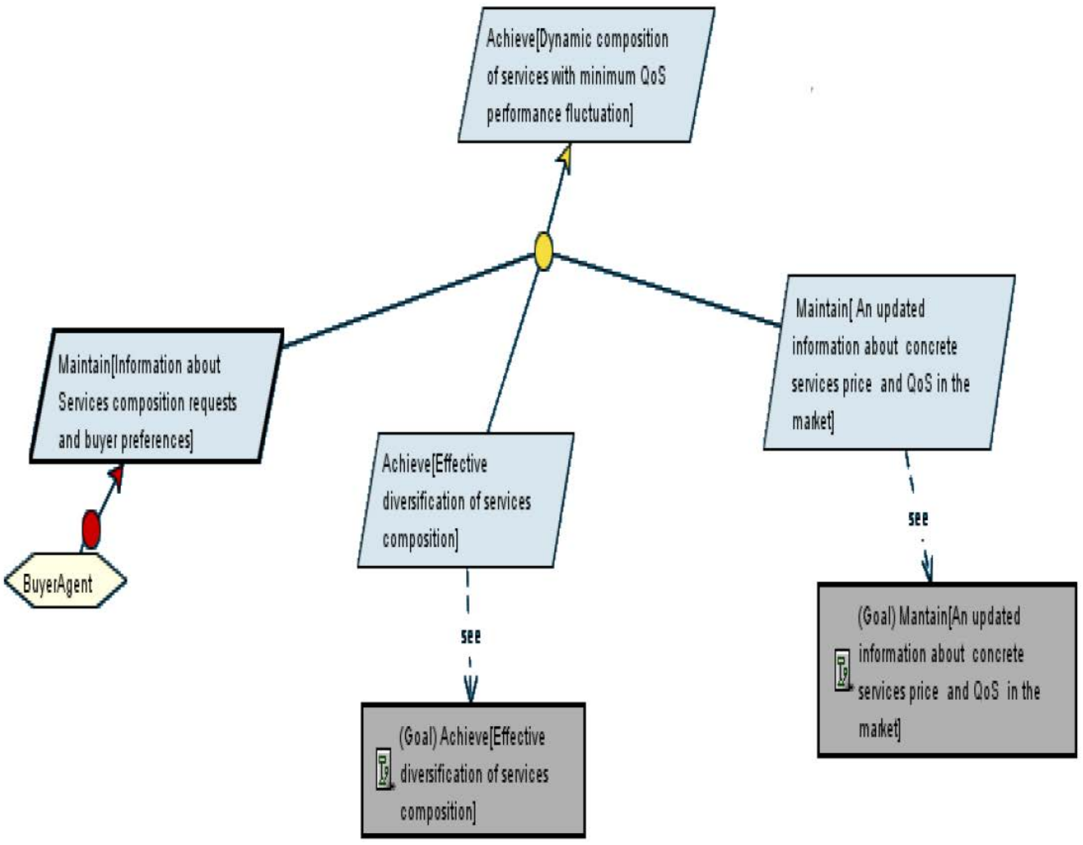

Achieve[Dynamic composition of services with minimum QoS performance fluctuation]
Maintain[Information about Services composition requests and buyer preferences]
BuyerAgent
Achieve[Effective diversification of services composition]
Maintain[An updated information about concrete services price and QoS in the market]
see
see
(Goal) Achieve[Effective diversification of services composition]
(Goal) Mantain[An updated information about concrete services price and QoS in the market]

Figure 6.1: Portion of the Goal Model for Portfolio Based Composition.

6.2.2 Overview Of Goal Obstacle Analysis

Goals, requirements and assumptions of the firstly produced KAOS goal model are often too ideal [139]. The reason for that is failing to consider extreme scenarios and exceptional conditions in the domain (such as having a large number of concurrent requests or a network failure). These exceptional conditions can present a violation to some of the system goals. To avoid such violations, the KAOS framework includes a *goal-obstacle analysis* that takes a pessimistic view of goal model [35].

The objective of the goal-obstacle analysis is checking the model looking for exceptional conditions and scenarios that prevent the goal from being satisfied. In the model, these exceptional conditions and scenarios are referred to as obstacles [134]. Starting from an elaborated KAOS model, goal-obstacle analysis consists of three main activities:

1. Identifying all obstacles that can hinder the realisation of system goals.
2. The assessment of the likelihood and criticality of identified obstacles on top-level goals.
3. Resolving the most important obstacles by modifying goals and assumptions of the model.

Duboc et al [35] defined scalability as the ability of a system to achieve an acceptable level of satisfaction for each of its quality goals when the system variable vary over expected operational ranges. Based on this definition, we need to identify the following elements in order to evaluate the scalability of a system:

1. Quality goals correspond to goals and their objective function.
2. Expected operational variations of variables in the application domain and these are referred to as scaling assumption.
3. Acceptable level of satisfaction for each of the system quality goals and these are referred to as scalability goal.

To define these elements Duboc et al. [35] introduced a methodology which extends the conventional KAOS framework with two new concepts: *scaling assumption* and *scalability goal*.

6.2.3 Scalability Assumption

The scaling assumption is defined as a domain assumption specifying the expected variation of certain variables in the application domain in specific deployment environment. In our scalability analysis, the application domain is web services composition and we have the cloud as the deployment environment. The scaling assumptions should define as follows:

1. One or more variables in the application domain that are expected to vary in ranges of values.
2. The expected range of values for variables in the application domain in a specific deployment environment.

For example, one of the variables that affect the scalability of the portfolio-based composition is the number of candidate services per abstract service. A scaling assumption associated with that variable could be the expected number of candidate services per abstract service, which can be described as:

Assumption Expected number of candidate services per abstract service.

Category Scaling assumption.

Definition The number of candidate services per abstract service is expected to vary between 3 and 50 candidate services.

The ranges of values used in scaling assumptions must be identified and negotiated with system stakeholder. The ranges may be obtained by analysing the existing system [35]. Each identified scalability assumption presents constraints on the ranges of values for each system variables. The absence of a scaling assumption for system variables indicates that there is no assumed constraint on its possible values and it could be infinite.

To illustrate this, if we consider the goal: *Achieve [Services composition is selected quickly]*, the absence of a scalability assumption that limits the range of values of candidate services per abstract service means that no agent can satisfy the goal unless he has unlimited capacity. However, that is not feasible as all the agents have limited capacity.

6.2.4 Scalability Goal

A *scalability goal* is a system goal that requires as part of its definition expected level of satisfaction specified by the stakeholders and makes an explicit reference to one or more scaling assumptions [35]. As an example, if we consider the system goal *Achieve [Services composition is selected quickly]* as well as the scalability assumption, *Expected number of candidate services per abstract service.* We can manage to elaborate a new sub-goal; which is A*chieve [Services composition is constructed quickly under expected number of candidate services per abstract service].* The new sub-goal is a scalability goal because it makes an explicit reference to a scaling assumption, and it can be described as:
Goal *Achieve [Services composition is selected quickly under expected number of candidate services per abstract service]*
Category Performance goal, Scalability goal.
Definition A service composition defined as a workflow of a set of abstract services should be composed of a set of candidate services where the constructed composition should maintain the budget and the QoS constraints specified by the Buyer. The time taken to find appropriate services should not exceed 5 minutes (the stakeholder specifies time limit), as long as the number of candidate services does not exceed the bounds stated in the scaling assumption *Expected number of candidate services per abstract service.* This goal can now feasibly be satisfied by a Buyer agent with sufficient capacities and no longer requires an unlimited capacity as in its parent goal *Achieve [Services composition is selected quickly].*

6.3 Scalability Goal-Obstacle Analysis

The KAOS goal model fails to explicitly consider the scalability of the system defined in terms of goal load and agent capacity. For that reason, the scalability goal-obstacle analysis [35] extends the KAOS model with concepts of scaling assumptions and scalability goals. The basic steps of the scalability goal-obstacle analysis are illustrated in figure 6.2.

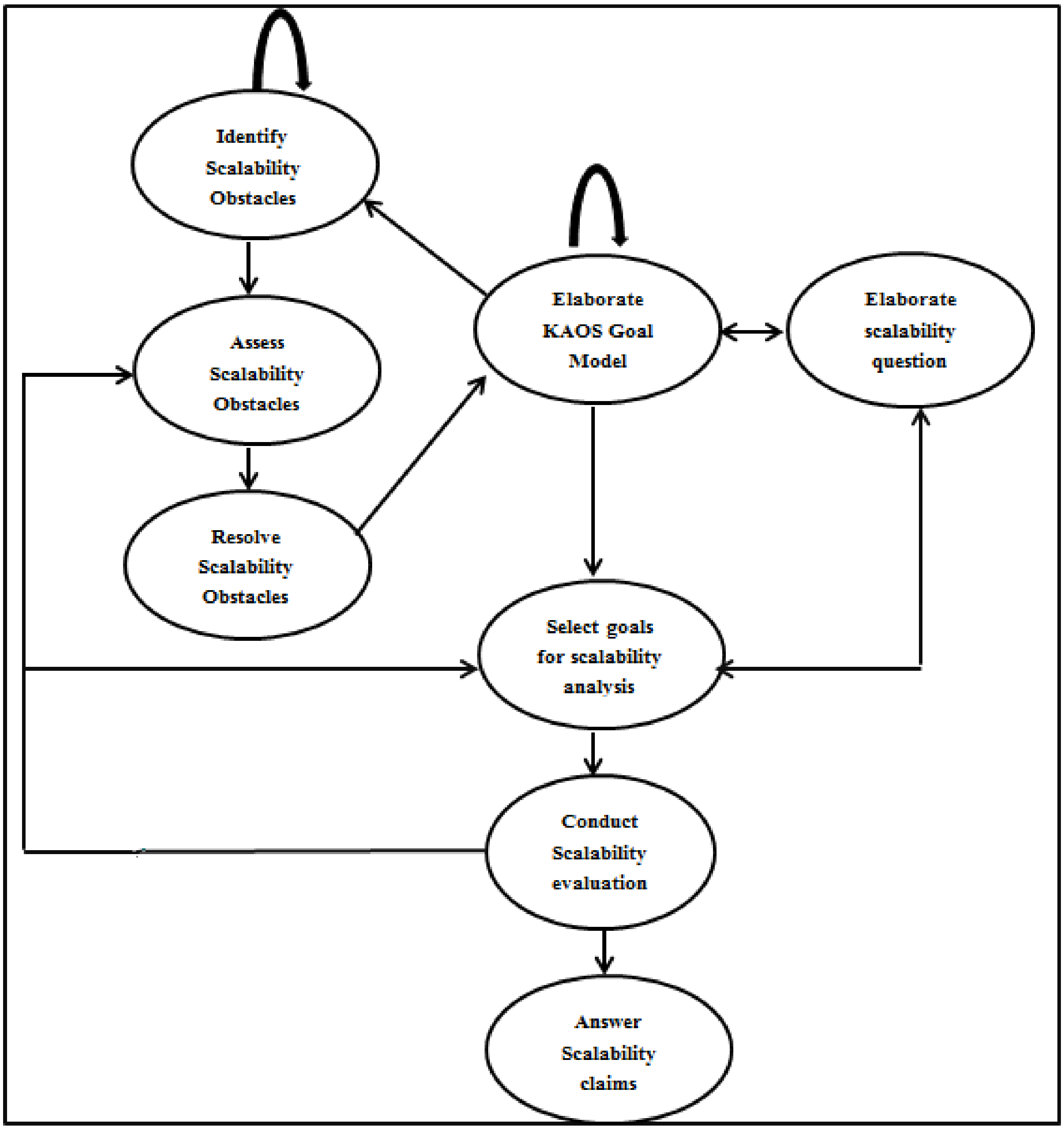

Identify Scalability Obstacles
Assess Scalability Obstacles
Resolve Scalability Obstacles
Elaborate KAOS Goal Model
Elaborate scalability question
Select goals for scalability analysis
Conduct Scalability evaluation
Answer Scalability claims

Figure 6.2: Basic Steps of the Scalability Goal-Obstacle Analysis [35].

The main objective of the approach is to help in identifying the relevant scalability goals for a software system. Such analysis requires a proper understanding of the system goals and the system variables that will influence these goals. To gain such understating, the approach begins with a KAOS goal model in order to identify the system goals. After that, it presents a set of well-defined steps to derive the variables and functions to be used in the scalability analysis from the KAOS goal-oriented model. These steps include iteratively identifying, assessing and resolving scalability obstacles followed by updating the goal model with *scaling assumptions* and *scalability goals*. Thereafter, a set of scalability goals are selected for analysis. The scalability evaluation is conducted at the end, and the answer to the scalability question is stated. In the following sections, we will present more details about the activities related to identifying, assessing and resolving scalability obstacles.

6.3.1 Identifying Scalability Obstacles

Scalability obstacle is a condition that prevents an agent from achieving the required level of satisfaction of a system goal. This is because the load imposed by the goal exceeds the capacity of the agents. To identify all the scalability obstacles related to a goal: we need to identify the goal load and specify the agent capacity. We define scalability obstacle in the form of *Goal Load Exceeds Agent Capacity.*

For example, consider again the goal *Achieve [Services composition is selected quickly]* assigned to the Buyer agent. The obstacle analysis for that goal has identified two types of obstacles that can prevent the Buyer agent from satisfying the goal: 1) functional obstacles and 2) scalability obstacles. Figure 6.3 shows the obstacles related to goal *Achieve [Services composition is selected quickly].*

Functional obstacles related to the goal are *No candidate services meet QoS and budget requirements* and *Market infrastructure not accessible.* To identify the scalability obstacles for a goal, it is necessary to define what the agent capacity is and what are the goal loads. The agent capacity is the Buyer agent's ability to compose services defined on term of seconds. The goal loads in this case are:

1. Workflow size.

2. Number of candidate services per abstract services in the workflow.

3. Number of services available in the market.

4. Number of QoS considered in the selection.

5. Number of concurrent service composition request.

A scalability obstacle in this case will be the condition when one of these goals loads exceeds the buyer agent ability to select services within imposed time limit.

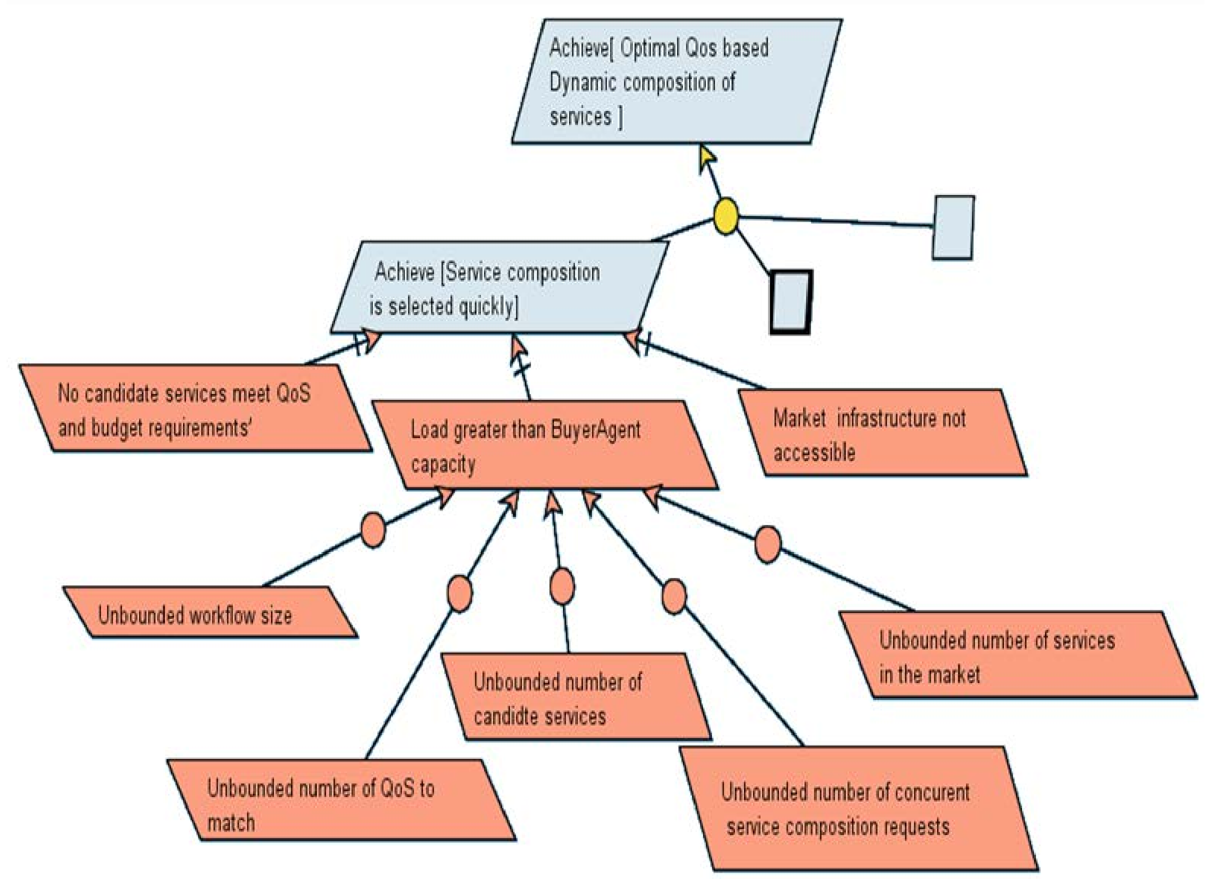

Achieve[Optimal Qos based Dynamic composition of services]
Achieve [Service composition is selected quickly]
No candidate services meet QoS and budget requirements'
Load greater than BuyerAgent capacity
Market infrastructure not accessible
Unbounded workflow size
Unbounded number of services in the market
Unbounded number of candidte services
Unbounded number of QoS to match
Unbounded number of concurent service composition requests

Figure 6.3: The Obstacles Related to Goal *Achieve [Services Composition is Selected Quickly]*

6.3.2 Assessing Scalability Obstacles

After identifying potential scalability obstacles, it is necessary to assess the criticality and likelihood of each one. The criticality of scalability obstacles indicates their effect on satisfaction of top level goals. The likelihood defines the probability of each scalability obstacle to take place in the application domain. A technique to support the assessment of scalability obstacles is inspired by standard qualitative risk analysis matrix. In this matrix, the likelihood of scalability obstacle can be estimated on a scale from low to high and similar scale to estimate criticality [141]. The risk of scalability obstacle is the product of its likelihood and its criticality. The obstacle assessment aims to separate the scalability obstacles that impose a high risk on the system needed from the low risk scalability obstacles, which can be safely be ignored.

If we take for example scalability obstacles *Unbounded number of candidate services* and *Unbounded number of available services* related to goal *Achieve [Services composition is selected quickly]* presented in figure 6.3. While both of the obstacles have the same likelihood of high possibility to take place in a cloud-based market, their criticality on services selection differ. The obstacle *Unbounded number of available services* has a low criticality on the selection, because most of the available services will be excluded from search space as they do not match functional or QoS constraints. On the other hand, the *Unbounded number of candidate services* will have catastrophic effect on the size of search space as all candidate services are considered as an option for composition. Considering both of the likelihood and criticality of the two scalability obstacles, we can say that obstacle *Unbounded number of candidate services* impose high risk on the system and that there is an urgent need to resolve it, while the risk associated with scalability obstacle *Unbounded number of available services* is relatively low and can be safely ignored.

6.3.3 Resolving Scalability Obstacles

As we discuss earlier, only high risk scalability obstacles are considered serious enough and must be resolved. Duboc [139] has presented a number of specialized resolution tactics to overcome scalability obstacles. This section presents a brief description that covers two of these tactics: *goal weakening* and *goal substitution.*

Goal Weakening

Sometimes the goal definition can be strong and its achievement may introduce a scalability obstacle to the system. In this case, we can resolve the obstacle by changing the goal definition to a more liberal one with the aim of eliminating the obstacle. The following goal weakening strategy can be adopted to help in resolving scalability obstacles:

1. *Weaken goal definition by introducing scaling assumption*: this goal weakening strategy can be implemented by limiting goal satisfaction to an assumption that specifies a range of expected load that does not exceed the agent capacity.
2. *Weaken goal objective function*: this goal weakening strategy can be implemented by weakening the goals objective function, in such a way that it does not exceed the agent capacity.

For instance, applying the strategy weaken goal definition introduces scaling assumption to the goal *Achieve [Services composition is selected quickly].* This can be realised by introducing a scalability assumption, *Expected number of candidate services per abstract service.* This will create a weaker new goal *Achieve [Services composition is selected quickly under expected number of candidate services per abstract service]* that is easier to satisfy than the original goal. An alternative way to resolve the obstacle is to apply the weaken goal objective function to the goal *Achieve [Services composition is selected quickly].* This can be achieved by relaxing the time limit imposed by the stakeholder from 5 to 7 minutes.

Goal Substitution

Obstacle prevention can be implemented by replacing an obstructed goal with an alternative goal that aims to eliminate the obstacle completely. For example, if a goal *Achieve [an optimal services composition is selected within 5 minutes]* was obstructed by the time limit because it requires full exploration of the search space. We can overcome this obstacle by introducing alternative goal as *Achieve [a sub-optimal services composition is selected within 5 minutes].* The new goal is more likely to require less time to satisfy as a sub-optimal selection does not require full exploration of the search space.

6.4 Scalability Goal-Obstacle Analysis for Portfolio-Based Service Composition

In this section, we demonstrate how goal modelling can be used to: (1) elaborate the scalability requirements for the portfolio based services composition and; (2) identify the dimensions relevant to the scalability analysis. A Cloud services composition should be specified as a workflow of abstract services where each abstract service describes the functional specification of a certain task. Cloud market is expected to offer multiple candidate services that satisfy the functional requirement but come with different QoS and fluctuation rate. In this context, we view the selection of cloud services composition from a cloud-based market as an optimisation problem, aiming to reduce possible risks of performance fluctuation.

The mechanism modifies the composition in response to changes in the market. For a given adaptation cycle, the adaptation decisions are informed by the extent to which diversification can reduce risk of performance fluctuation subject to QoS and cost constraints. In the following sections, we present the goal modelling and main agents of system. Then, we present the main obstacles that can affect the scalability of the system and the scalability goals of the portfolio-based composition.

6.4.1 Goal Modelling of Portfolio Based Composition

The portfolio-based services composition has 15 goals and the scalability goal-obstacle analysis revealed 8 potential scalability obstacles, which led to the identification of 7 scaling dimensions. In the following subsection, we discuss the agents and their main goals, findings of scalability goal-obstacle analyses, and their resulting metrics and dimensions. The elaborated goal model and a brief explanation of its parts can be seen in Appendix A.

6.4.2 Agents and Goals in the KAOS Goal Model

The KAOS goal model for portfolio-based composition defines the following agents:

Buyer:

This agent is acting on behalf of the application stakeholder. It is responsible for selecting a well-diversified set of candidate service that satisfy the requirements of the stakeholder and achieve optimal performance stability. First, services are screened for compatibility before they are shortlisted as candidates. When candidate services exist in the market, this agent implements portfolio theory to select a set of compatible services that have low correlation between their performance. Otherwise, if no candidate service exists, it notifies the buyer. When there is a change in the QoS of the candidate services or an SLA contract expires, this agent is responsible for recomposing the services composition. This agent is responsible for the goals *Achieve [Services composition is composed from selected concrete services that share minimum correlation between their performance]*, *Maintain[information about services composition request and buyer preference]*, *Achieve[Service composition is recomposed if there is change in the condition of one of the services or SLA contract expires]* and *Achieve[Optimality of the composition by exploring all the possible solutions]*. (see Figures 6.1 A.2 and A.3 in Appendix A).

Seller:

This agent is acting on behalf of the services providers. (s)he is responsible for publishing a new concrete service on the market services registry, informing its functionality and price. (S)he should also inform about any change in the functionality and price of an existing service. This agent is responsible for the goals *Achieve [New services and change in existing price or functionality is reported]*. (See Figures A.1 in Appendix A).

MarketRegulator:
This agent is an independent software agent that has a number of responsibilities. It stores services information in the market registry. Moreover, it returns a set of candidate services given a buyer QoS and budget requirement. In addition, it is responsible for maintaining the historical records of the QoS and the fluctuation rate of each service in the market. Finally, it is responsible for informing the buyer about the correlation between different services in the market, the buyer use correlation to form a well-diversified composition of services. This agent is responsible for the goals *Maintain [Updated concrete services risk of fluctuation and correlation between performance]*, *Achieve[Historical record of QoS is recorded]*, *Achieve[an updated information regarding concrete services available]* and *Achieve [New services and change in existing price or functionality is recorded]* (see Figures A.1, A.2, and A.3 in Appendix A).

6.4.3 Scalability Goal-Obstacle Analysis of Portfolio-Based Services Composition

In this section, we present the first stage of the scalability obstacle analysis, which is identifying application domain characteristic related to system goals. Then, we will define a variety of scalability dimensions and metrics that need to be taken into account when evaluating the scalability of the system.

Goal oriented analysis of the portfolio-based services composition has identified 15 goals of portfolio-based service composition. Two of these goals and their scalability obstacles are illustrated below; The same rationale can be applied to the rest of the 15 goals and are presented in table 6.1.

Maintain[Updated evaluation of concrete services risk of performance fluctuation and correlation between services]
This goal states that the MarketRegulator is responsible for evaluating the risk of performance fluctuation for each service and the performance correlation for each service in the market. This goal is satisfied if all the required information is evaluated and stored within a reasonable time and without exhausting the Market Regulator processing capacity or the available storage space. The goal load is determined by the number of services in the market, the number of considered QoS, the number of changes in QoS and the length of the historical record of each QoS. The Market Regulator agent has a finite capacity and cannot handle an unlimited number of services, QoS or unlimited length of the historical records. The scalability goal load exceeds Market Regulator agent capacity, which can be further refined into *unlimited number of services, unlimited number of QoS, and unlimited length of the historical records.* These obstacles are resolved by introducing the scaling assumptions, *Expected number of services, Expected number of QoS , and limited length of the historical record.* These scaling assumptions limit range of value replacing the original goal by the scalability goal *Maintain[Updated evaluation of concrete services risk of performance fluctuation and correlation for expected number of services requests ,QoS and limited length of the historical record].*

Achieve[Effective diversification of services composition]
This goal states that once the group of the compatible services exist in the market, the Buyer agent is responsible for effectively diversifying the selection of a services composition that has a low risk of performance fluctuation. This goal is satisfied when the buyer agent finds an optimal set of candidate services within acceptable time. The load this goal imposes on the Buyer agent is determined by the number of concurrent application requests and number of QoS constraints. As the buyer agent has a limited capacity, (s)he cannot handle an unlimited number of concurrent application requests or unlimited number of constraints for QoS. The scenario where the scalability obstacle goal load exceeds

Buyer agent capacity can be resolved by introducing the scaling assumptions of *Expected number of concurrent application requests and Expected number of QoS constraints*, and replacing the original goal by the scalability goal *Achieve[Effective diversification of services composition under expected number of concurrent application requests and num. of constraints for QoS].*

As a result of the goal-obstacle analysis, four metrics and nine application domain characteristics may be considered relevant to the scalability analysis of portfolio-based services composition. These are:

Metrics: *performance fluctuation, execution time, communication bandwidth usage and disk storage space.*

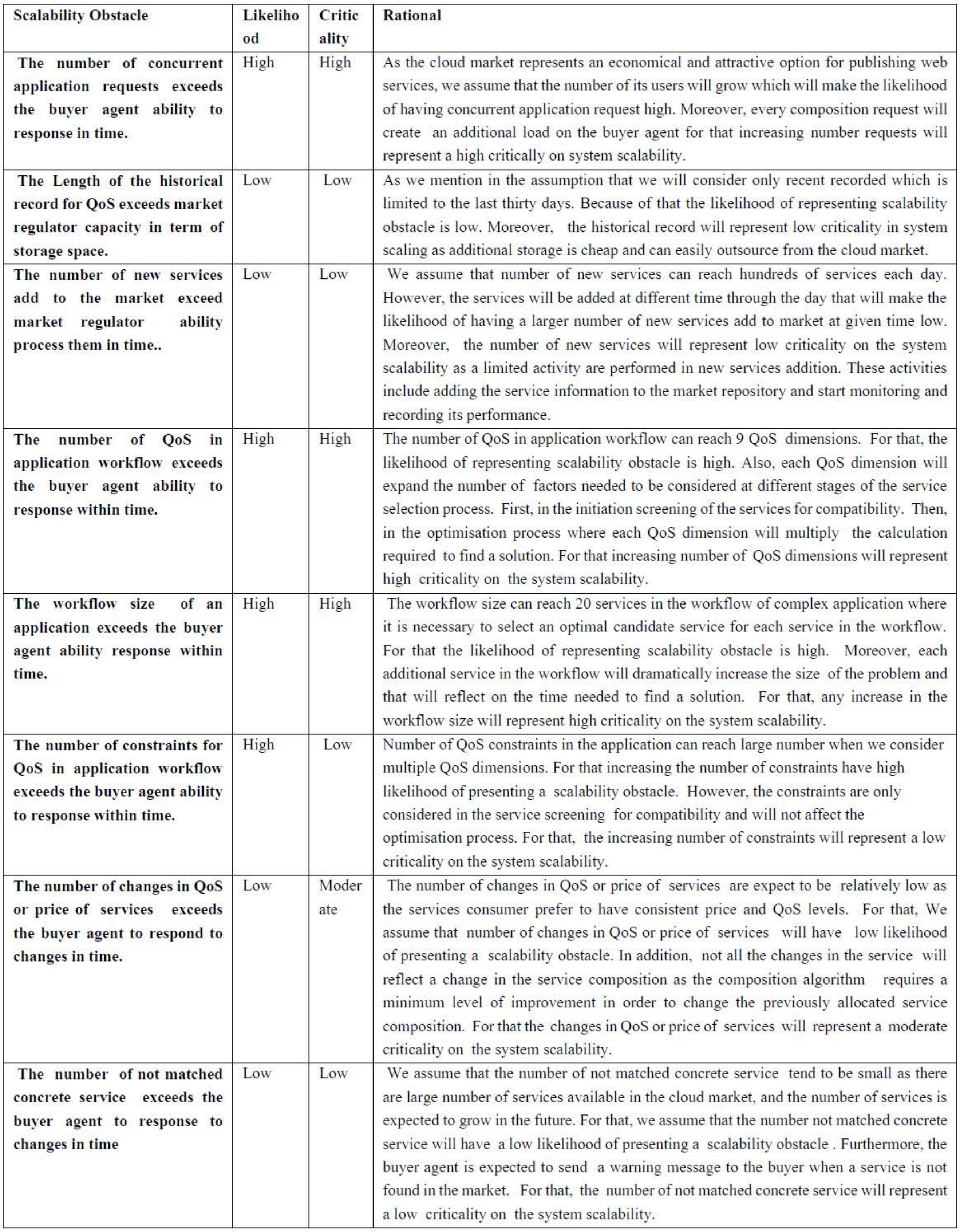
Application domain characteristics: n*umber of the concrete services in the market, number of concurrent application requests, number of expired SLA contract at a given time, workflow size, the length of the historical records for QoS, number of QoS in application, number of changes in QoS or price of services, number of candidate services per abstract service and the number of not matched concrete service.*

After identifying all goals of portfolio-based composition, we define the scalability obstacles of the system that may prevent some of the goals from being satisfied. A list of these scalability obstacles and an assessment of their likelihood and criticality on the system scalability are highlighted in Table 6.2.

Table 6.1: Metrics and Unbounded Variables for each of the Portfolio Based Composition Goals

Goal	metric	Unbounded variables	Influenced by
Maintain[An updated information about concrete services price and QoS]	execution time, communication bandwidth	num. of concrete services, num. of changes in QoS or price of existing concrete services, num. of new concrete services add to the market.	
Achieve[Services composition is created from selected concrete services that share minimum correlation between their QoS performance]	execution time, communication bandwidth, QoS fluctuation.	workflow size, num. of application, num. of constraints for QoS in the application, workflow size, number of candidates concrete service per abstract service.	number of candidates concrete service per abstract service <depend on >[num. of QoS per abstract service, num. concrete service per abstract service]
Maintain[An updated evaluation of concrete services risk of fluctuation and correlation between QoS]	execution time, communication bandwidth, storage space	Workflow size, number of candidates concrete service, num. of changes in QoS or price of existing concrete services, num. of QoS per abstract service.	num. of changes in QoS or price of existing concrete services <depend on >[num. of application requests, num. of virtual machines available in the data center, num. of QoS per abstract service]
Maintain[Information about services composition request and buyer preference]	communication bandwidth, storage space	workflow size, num. of application requests, num. of constraints for QoS in the application	
Achieve[Historical record of Qos is stored]	execution time, communication bandwidth, storage space	Number of candidates concrete service in the market, Length of the historical QoS record, num. of changes in QoS or price of existing concrete services , num. of constraints for QoS in the application,	
Achieve[Change in price or functionality of concrete services is recorded]	communication bandwidth, ,storage space	number of concrete services in the market, num. of changes in functionality or price of existing concrete services	
Achieve[Optimality of the composition by exploring all the possible solutions]	execution time, QoS fluctuation. , storage space	Number of candidates concrete service in the market, workflow size, num. of Qos dimension,	
Achieve[Service composition is recomposed if there is a change in the condition of one of the services or SLA contract expires]	execution time, communication bandwidth, QoS fluctuation.	number of candidates concrete service per abstract, num. of changes in QoS or price of existing concrete services, num. of expired SLAs at a given time, num. of QoS per abstract service.	num. of changes in QoS or price of existing concrete services <depend on >[num. of application requests, num. of virtual machines available in the data center, num. of QoS per abstract service, buyer preference]
Achieve[New services and change in existing services price or functionality is reported]	communication, bandwidth.	Number of new service in the market, num. of changes in functionality or price of existing concrete services.	
Achieve[New services and change in existing services price or functionality is recorded]	storage space	Number of new service in the market, num. of changes in functionality or price of existing concrete services.	
Achieve[Services selected if concrete services that meet QoS and budget exist]	execution time, communication bandwidth	workflow size, num. of application, num. of constraints for QoS in the application, num. of Qos dimension , number of candidates concrete service per abstract service.	number of candidates concrete service per abstract service <depend on >[num. of QoS per abstract service, num. concrete service per abstract service]
Achieve[Services composition is selected based on desired workflow and constraints]	execution time, communication bandwidth	number of concrete services in the market ,workflow size, num. of application, num. of Qos dimension, num. of constraints for QoS in the application, number of candidates concrete service per abstract service.	number of candidates concrete service per abstract service <depend on >[num. of QoS per abstract service, num. concrete service per abstract service]
Achieve[Workflow and Qos/ budget desired for a services composition informed]	Communication, bandwidth	workflow size, num. of application, num. of Qos dimension, num. of constraints for QoS in the application.	
Achieve[Services composition is created from a set of matched services]	execution time, communication, bandwidth	Number of candidates concrete service in the market, workflow size , num. of constraints for QoS in the application .	
Achieve[Set of candidate services found]	execution time, communication, bandwidth	Number of candidates concrete service in the market, workflow size.	

Table 6.2: Assessment of Likelihood and Criticality of Scalability Obstacles



Scalability Obstacle	Likelihood	Criticality	Rational
The number of concurrent application requests exceeds the buyer agent ability to response in time.	High	High	As the cloud market represents an economical and attractive option for publishing web services, we assume that the number of its users will grow which will make the likelihood of having concurrent application request high. Moreover, every composition request will create an additional load on the buyer agent for that increasing number requests will represent a high critically on system scalability.
The Length of the historical record for QoS exceeds market regulator capacity in term of storage space.	Low	Low	As we mention in the assumption that we will consider only recent recorded which is limited to the last thirty days. Because of that the likelihood of representing scalability obstacle is low. Moreover, the historical record will represent low criticality in system scaling as additional storage is cheap and can easily outsource from the cloud market.
The number of new services add to the market exceed market regulator ability process them in time..	Low	Low	We assume that number of new services can reach hundreds of services each day. However, the services will be added at different time through the day that will make the likelihood of having a larger number of new services add to market at given time low. Moreover, the number of new services will represent low criticality on the system scalability as a limited activity are performed in new services addition. These activities include adding the service information to the market repository and start monitoring and recording its performance.
The number of QoS in application workflow exceeds the buyer agent ability to response within time.	High	High	The number of QoS in application workflow can reach 9 QoS dimensions. For that, the likelihood of representing scalability obstacle is high. Also, each QoS dimension will expand the number of factors needed to be considered at different stages of the service selection process. First, in the initiation screening of the services for compatibility. Then, in the optimisation process where each QoS dimension will multiply the calculation required to find a solution. For that increasing number of QoS dimensions will represent high criticality on the system scalability.
The workflow size of an application exceeds the buyer agent ability response within time.	High	High	The workflow size can reach 20 services in the workflow of complex application where it is necessary to select an optimal candidate service for each service in the workflow. For that the likelihood of representing scalability obstacle is high. Moreover, each additional service in the workflow will dramatically increase the size of the problem and that will reflect on the time needed to find a solution. For that, any increase in the workflow size will represent high criticality on the system scalability.
The number of constraints for QoS in application workflow exceeds the buyer agent ability to response within time.	High	Low	Number of QoS constraints in the application can reach large number when we consider multiple QoS dimensions. For that increasing the number of constraints have high likelihood of presenting a scalability obstacle. However, the constraints are only considered in the service screening for compatibility and will not affect the optimisation process. For that, the increasing number of constraints will represent a low criticality on the system scalability.
The number of changes in QoS or price of services exceeds the buyer agent to respond to changes in time.	Low	Moderate	The number of changes in QoS or price of services are expect to be relatively low as the services consumer prefer to have consistent price and QoS levels. For that, We assume that number of changes in QoS or price of services will have low likelihood of presenting a scalability obstacle. In addition, not all the changes in the service will reflect a change in the service composition as the composition algorithm requires a minimum level of improvement in order to change the previously allocated service composition. For that the changes in QoS or price of services will represent a moderate criticality on the system scalability.
The number of not matched concrete service exceeds the buyer agent to response to changes in time	Low	Low	We assume that the number of not matched concrete service tend to be small as there are large number of services available in the cloud market, and the number of services is expected to grow in the future. For that, we assume that the number not matched concrete service will have a low likelihood of presenting a scalability obstacle . Furthermore, the buyer agent is expected to send a warning message to the buyer when a service is not found in the market. For that, the number of not matched concrete service will represent a low criticality on the system scalability.

6.5 Scalability Goals

From these obstacles presented in table 6.2, we consider only the obstacles that have high likelihood and high criticality on the system as scalability goals. As a result, the system has four scalability goals:

1. As the number of candidate services per abstract service increases, the algorithm should exhibit a linear growth in execution time.
2. As the number of abstract services in the workflow increases, the algorithm should exhibit a linear growth in execution time.
3. As the number of QoS attributes increases, the algorithm should exhibit a linear growth in execution time.
4. As the number of concurrent application requests increases, the algorithm should exhibit a linear growth in execution time.

6.6 Scalability Evaluation

We now perform a scalability evaluation to the portfolio-based composition where we evaluate the sensitivity of the time needed to find a solution, which increased in the previously identified four scalability goals.

These scalability goals do not prescribe any specific measurable quantity. Thus, it is difficult to know whether portfolio-based composition scales well, or not. In order to assign specific range of numbers for each variable, we present scaling ranges. These ranges go beyond the findings of the systematic literature review of [126] for dynamic services composition (see Table 6.3). To analyze the scalability of the system, we ran three experiments. The experimental setting follows that of Table 5.4 except that in each experiment we gradually increased the load in two dimensions as follows:

Table 6.3: Scalability Range For Each Variable

Variable Affect scaling	Scalability range
Number of candidate services per abstract	1-50
Number of abstract services in the workflow	1-20
Number of QoS	1-10
Number of concurrent application request	1-50

1. The first experiment evaluates the effect of increasing both the number of candidate services and the number of QoS on the execution time. The results are presented in Fig. 6.4.

2. The second experiment evaluates the effect of increasing both the number of candidate services and the number of concrete application requests on the execution time and the results are depicted in Fig. 6.5.

3. The third experiment evaluates the effect of increasing both the number of candidate services and the number of abstract services in the workflow on the execution time and the results are shown in Fig. 6.6.

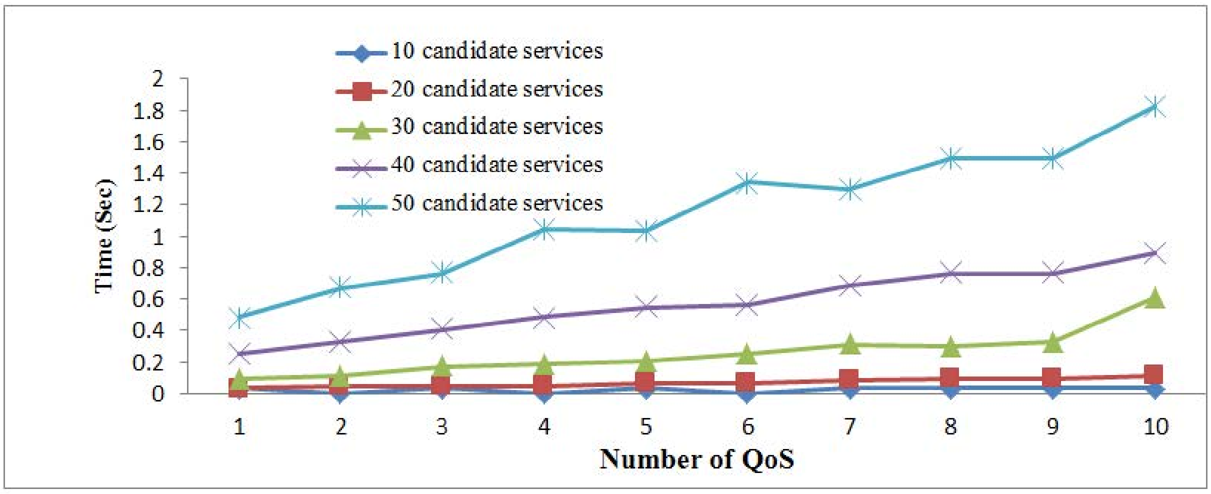


Figure 6.4: The Execution Time of The Portfolio Based Composition as Both of The Number of Candidate Services and Number of QoS Increases.

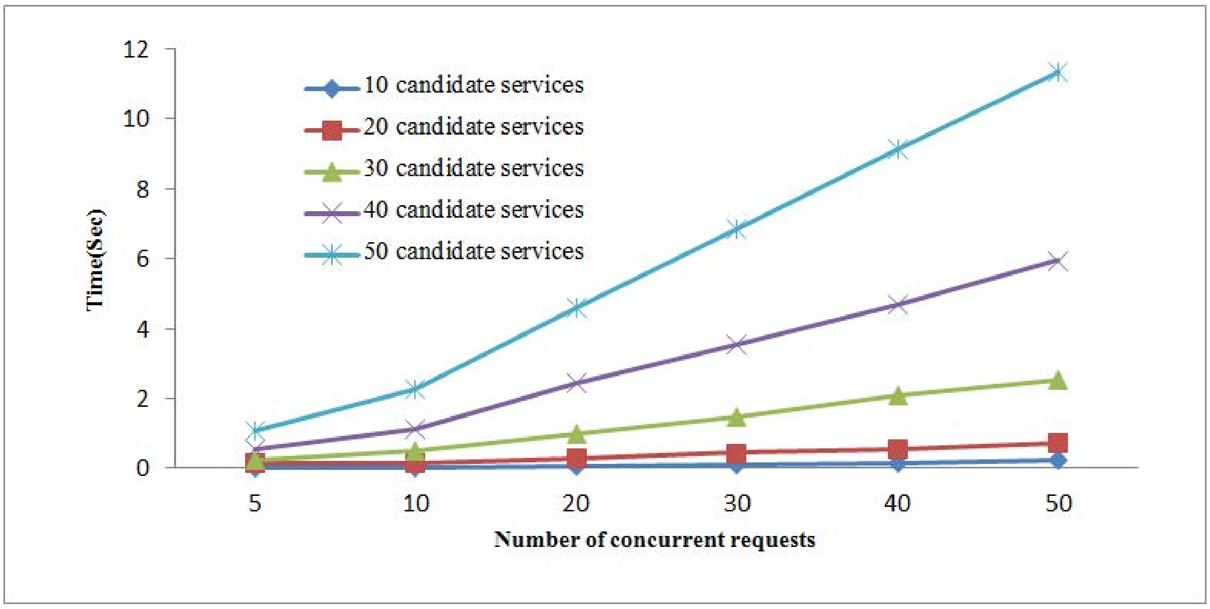


Figure 6.5: The Execution Time of the Portfolio Based Composition as Both of the Number of Candidate Services and Number of concurrent application requests Increases

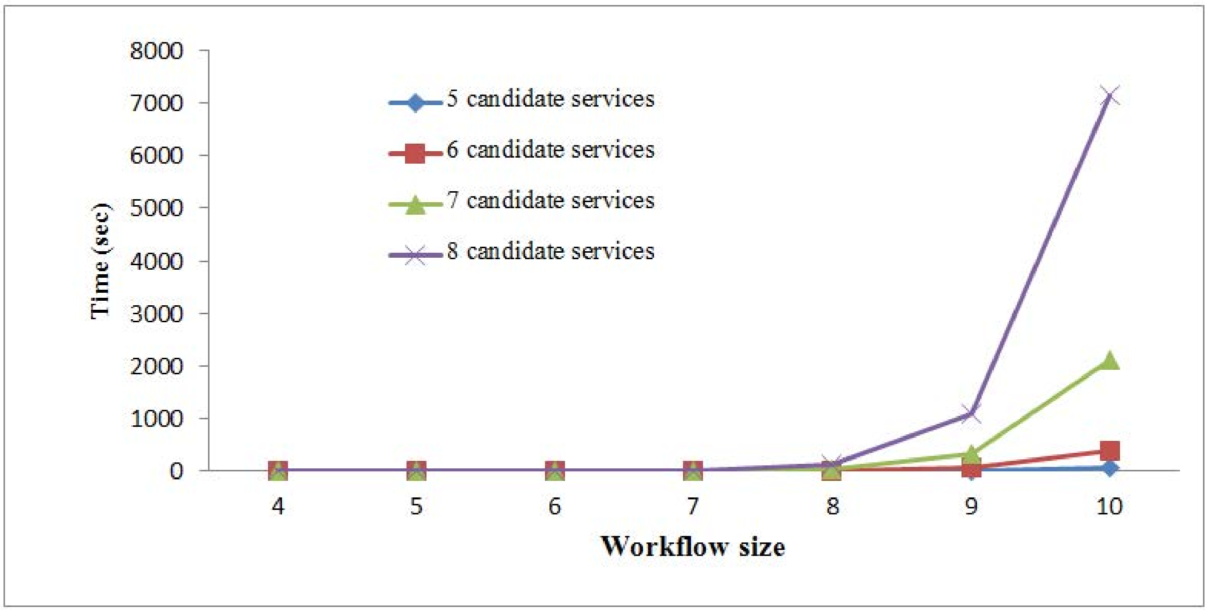


Figure 6.6: The Execution Time of the Portfolio-Based Composition as Both of The Number Of Candidate Services And Workflow Size (Number Of Abstract Services In The Composition) Increases.

From the results displayed in Fig.6.4 and Fig.6.5, it is essential to note that the portfolio-based composition algorithm exhibits a linear growth of execution time when we increase the number of candidate services, the number of QoS or the number of concurrent applications request. However, Fig.6.6 shows that the algorithm exhibits an exponential growth in execution time when we increase both the number of candidate services and the size of the workflow of the application. This indicates that the performance of the portfolio-based composition algorithm is highly sensitive to the increase in the workflow size. As a result, it takes longer time to find the solutions for applications with large workflow size.

6.7 Conclusion

Prior to applying goal-oriented analysis, we reported a scalability analysis of portfolio-based composition which considered execution time against one scalability goal which the number of candidate service [142]. As an immediate result of the goal-oriented analysis of portfolio-based composition, we identified three new scalability goals that are relevant to the scalability of portfolio-based composition.

The systematic analysis has provided more objective means for identifying scalability goals of interest. One advantage of using such disciplined analysis is that these dimensions could be easily missed in case ad hoc practice is used. The scalability evaluation revealed that the our approach could cope with the increasing numbers of candidate services, QoS attributes and concurrent application requests. However, it could not handle scaling of the abstract services in the workflow. More precisely, the scalability obstacle happened when both the abstract services in the workflow and the number of candidate services scale concomitantly. This issue had not been identified in the previous scalability analysis of the portfolio-based composition discussed in [142].

CHAPTER 7

CONCLUSION AND FUTURE WORK

This thesis is a result of our quest to find a stability-aware service selection and composition approach, which would enable a cloud-based application to improve it's performance stability. The thesis has introduced the following research questions:

- *RQ1: Can the concept of design diversity be applicable to the case of cloud services selection and composition to reduce risk of performance fluctuation? How well can it perform compared to well-established services selection methods?*
- *RQ2: How can the approach be extended to self-adaptive mechanism, which can dynamically respond to changes in the market?*
- *RQ3: In the case of CSC, what are the scaling dimensions (e.g. number of web services, number of objectives, candidate solutions, frequency and volatility of change etc.) that we need to render a pragmatic solution ?*

We started our quest for stability-aware services selection approach by reviewing the state of the art of QoS-aware service composition solution. We concluded that there were no solutions that address the problem of reducing performance fluctuation in cloud-based applications. We looked at various approaches that adopt the concept of design diversity to better stabilise cloud-based applications. We had also observed that these approaches have failed to address issues related to correlated failure.

To bridge the gap, we proposed a novel service selection approach, which implements design diversity principles to minimise performance instability in the cloud-based applications. The novelty of our approach emanates from combining the principles of design diversity with Modern Portfolio Theory [30] to select an optimal set of candidate services that share a minimum correlation between their performances to achieve more stable performance. Unlike the reviewed classical design diversity, our portfolio-based approach links the diversification of candidate services to performance fluctuations and correlation between different candidate services in the cloud-based application. To the best of our knowledge, we are not aware of any service selection approach that explicates diversity when architing applications through the selection of candidate cloud services with the aim of reducing the performance fluctuation.

We tackled the problem of self-adaptation by introducing a self-adaptive mechanism that utilised the MAPE control loop [34] to react to changes in the market and maintain the optimality of the cloud-based application in a runtime environment. The proposed self-adaptive approach makes an explicit trade-off between the cost, risks and benefits of performing changes to the cloud-based application. Moreover, we tailored the method of [35] to our case and performed a systematic elaboration of scalability requirements through goal-obstacle analysis to identify the four scalability dimensions, which can influence the behaviour of the portfolio-based algorithm in the case of CSC. Finally, we demonstrated the superiority of the portfolio-based approach by comparing it's performance to that of a number of existing approaches using different scenario of application and correlation settings.

7.1 Contributions of This Thesis

In proposing our stability-aware service selection approach, we borrowed ideas from economics, design diversity, goal oriented requirement engineering, and self-adaptive systems. We took these ideas and integrated them into a self-adaptive stability-aware service selection approach. That journey of reviewing ideas, filtering them for suitability and integrating them into one solution has yielded a few contributions to the field of software engineering. We list them below:

- **A literature review that cover the state of the art of QoS-aware service composition**: We review the existing work on QoS-aware services composition. The objective of the review is to draw from the state of the art solutions, new insights that can assist the problem of stability-aware dynamic selection for cloud-based applications. The review have helped to identify the main activities that support QoS-aware service composition in a dynamic environment. Moreover, it helped to identify three main challenges imposed by cloud environment on QoS-aware service composition. These challenges are performance fluctuation, scalability of the approach and its support for self-adaptivity.

- **Review existing design diversity solution**: The review was an important step towards understanding the research landscape and identifies the gaps of the current design diversity solutions. From the review, we discovered that ignoring the possibility of correlated failure can lead to an ill section of the diversified system. We also recommended that the diversification decision should be linked to the correlation between the candidate services of the applications.

- **A novel portfolio-based service selection algorithm**: We presented a self-adaptive multi-agent system that utilized the Modern Portfolio Theory to enable the diversification of the services selection in order to improve the performance stability of the cloud-based application. We illustrated the applicability and effectiveness of our proposed portfolio inspired method using two scenarios of application: a scaling up scenario and a cloud services composition that represents a more complex scenario of application.

- **Systematic elaboration of scalability requirements for Portfolio-based composition:** Building upon the work of Duboc et al. [35], we systematically analysed the scalability requirements of our portfolio-based composition. The scalability analysis helped us to identify four scalability dimensions on which the portfolio-based compositions should evaluated to verify it's scalability.

- **Conducting a systematic scalability analysis**: We tested the scalability of the portfolio-based services composition on all the dimensions that were identified through the scalability goal obstacle analysis. We showed how to evaluate a service composition algorithm in a systematic way. Systematic analysis has provided more objective means for identifying relevant scalability goals that can be easily missed if an ad hoc method had been used.

7.2 Concluding Remarks

We now summarise the results of the research carried out during the different stages of our Ph.D. The major conclusions that we can draw from this research are the following:

1. There is a pressing need imposed on service selection methods that targets cloud environment to systematically evaluate and improve the performance stability of the applications.

2. The design diversity concept can be used to improve the performance stability in the cloud environment.

3. The portfolio inspired method proposed in this thesis to implement diversity performs better than the classical design diversity methods and traditional services selection.

4. When designing a self-adaptive mechanism in the cloud environment, a trade-off needs to be explicitly considered between the benefits gained by the change and the architectural stability of the cloud-based application.

5. Among the four scalability dimensions covered in our evaluation, the mechanism is most sensitive to size workflow. In other words, a rise in the number of abstract services of the workflow would potentially raise the time required to encounter a set of optimal solutions.

6. The sensitivity of the algorithm on the evaluated scalability dimension (in increasing order) is as follows:

   (a) Number of QoS dimension.

   (b) Number of concurrent application request.

   (c) Number of candidate service.

   (d) The size of the workflow.

7.3 Future work

This thesis is a description of a path in the direction of self-adaptive and stability-aware services selection approach. This path does not terminate at this thesis, as we can see several directions that this research can take in the future:

7.3.1 A Fast Heuristic Portfolio-based Services Composition Algorithm that Seeks Near Optimal Solution

This thesis had investigated performance stability of the cloud-based application and the use portfolio theory to find an optimal solution, while maintaining a set of QoS constraints. However, as discussed in chapter 2, ensuring optimality comes with a high computational cost, since finding the optimal composition represents an NP-hard problem [9]. This is because ensuring optimality requires exploring all the possible compositions of the cloud market. This was confirmed in our scalability evaluation where we found that performance of the portfolio-based algorithm is highly sensitive to increase in workflow size. As a result, it takes longer time to find the solutions for applications with large workflow size.

A future direction of research can be directed towards designing a lightweight heuristic algorithm that seeks near-optimal composition (e.g. [143], [144], [145] and [146]) in order to avoid the high computational cost required for finding an optimal solution. These heuristic algorithms do not perform an exhaustive search that explores all the possible compositions, but they seek near optimal solution. The main goal of using these heuristic algorithms is to provide a lightweight mechanism to explore a subset of the search space that is more likely to lead to finding a satisfying solution. The mechanism will reduce the computational time needed for running these algorithms but sacrificing the optimality of the solution as a trade off.

7.3.2 Multi-Objective Cloud Services Composition Algorithm

The core focus of this thesis was directed towards improving the performance stability. As a result, the proposed approach focused on one objective for the web service composition, which is achieving minimum risk of performance fluctuation. Other QoS goals, such as security and cost were considered as constraints.

However, some service buyers are willing to take small risks of performance fluctuation in order to optimise other QoS such as the security and cost of the cloud-based application. A future direction of research can go toward undertaking an investigation of extending the current portfolio-based composition to consider a scenario of multi-objective Cloud Services Composition. The challenge in this case is to find an optimisation method that dynamically makes trade-off between the multiple conflicting QoS objectives in order to achieve high utility for cloud-based application.

7.3.3 Realistic Implementation on The Cloud

Many vendors have entered the cloud market offering a range os SAAS services. However, deploying application to a cloud and managing them needs to be done using unique API for each vendor. This '*lock in*' is seen as a major hurdle when building cloud based application that integrate services from multiple vendors.

This thesis has focused on the fundamentals, future work will look industrial application. Further extensions to the model/approach to reflect domain-specific requirements and tradeoffs. How domainspecific requirements of providers and cloud markets can inform the design of middleware than extend on our work. We foresee the creation of a middleware that enable interoperability across multiple cloud vendors,and the evaluate it's performance using real word applications.

LIST OF REFERENCES

[1] P. Mell and T. Grance, "The nist definition of cloud computing," National Institute of Standards and Technology, 2011.

[2] M. R. Meybodi, "Decreasing Impact of SLA Violations:A Proactive Resource Allocation Approachfor Cloud Computing Environments," IEEE Transactions on Cloud Computing, vol. 2, no. 2, pp. 156-167, April-June 2014.

[3] CloudSleuth, "GLOBAL PROVIDER VIEW," Compuware, https://cloudsleuth.net/global-provider-view, accessed on 17/6/2014.

[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg and I. Brandic, "Cloud Computing and Emerging IT Platforms:Vision, Hype, and Reality for Delivering Computing as the 5th Utility," Future Generation Computer Systems, pp. Volume 25, Issue 6, June 2009, Pages 599–616, 2009.

[5] A. Jula, E. Sundararajan and Z. Othman, "Cloud Computing Service Composition: A Systematic Literature Review," Expert Systems with Applications, vol. 41, pp. 3809-3824, june 2014.

[6] K. Jackson, L. Ramakrishnan, K. Murik, S. Canon, S. Cholia, J. Shalf, H. Wasserman and N. Wright, "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud," in the 2nd International Conference on Cloud Computing Technology and Science., Indianapolis, USA, 2010.

[7] R. Richards, " Universal Description, Discovery, and Integration (UDDI)," in Pro PHP XML and Web Services, Apress, 2006, pp. 751-780.

[8] J. Dejun, G. Pierre and C. Chi, "EC2 Performance Analysis for Resource Provisioning of Service-Oriented Applications," in Proceedings of the 7th International Conference on Service Oriented Computing, , Stockholm, Sweden, 2009.

[9] M. Alrifai, D. Skoutas and T. Risse, "Selecting skyline services for QoS-based web service composition," in The 19th international conference on World Wide Web, New York, NY, USA, 2010.

[10] L. Qia, W. Doua, X. Zhangc and J. Chenc, "A QoS-aware composition method supporting cross-platform service invocation in cloud environment," Journal of Computer and System Sciences, vol. 78, no. 5, p. 1316–1329, September 2012.

[11] J. Gutierrez-Garcia and S. Kwang-Mong, "Self-Organizing Agents for Service Composition in Cloud Computing," in IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom),, Indianapolis, USA, 2010.

[12] S. Chattopadhyay, A. Banerjee and N. Banerjee., "A Scalable and Approximate Mechanism for Web Service Composition," in Web Services (ICWS), IEEE International Conference on. IEEE, New York,USA, 2015.

[13] Y. Feng, L. D. Ngan and R. Kanagasabai, "Dynamic service composition with service-dependent QoS attributes," in Web Services (ICWS), 2013 IEEE 20th International Conference on. IEEE, Santa Clara,USA, 2013.

[14] E. Constante, F. Paci and N. Zannone, "Privacy-Aware Web Service Composition and Ranking," in IEEE 20th International Conference Web Services (ICWS), Santa Clara, USA, 2013.

[15] I. Guidara, N. Guermouche, T. Chaari, S. Tazi and M. Jmaiel, "Heuristic Based Time-Aware Service Selection Approach," in In Web Services (ICWS), IEEE International Conference on, New York, USA, 2015.

[16] E. Maximilien and M. Singh, "Conceptual Model of Web Services Reputation," ACM SIGMOD Record, vol. 31, no. 4, pp. 36-41 , December 2002.

[17] R. Jurca, W. Binder and B. Faltings, "Reliable QoS Monitoring Based on Client Feedback." in the 16th International World Wide Web Conference, Banff, Canada, 2007.

[18] S. Kalepu, S. Krishnaswamy and S. Loke, "Reputation = f(user ranking, compliance, verity)," in Proceedings of IEEE International Conference on Web Services , San Diego, USA,2004.

[19] Z. Malik and A. Bouguettaya, "RATEWeb: Reputation Assessmentment for Trust Establishment among Web Services," The VLDB Journal, vol. 18, p. 885–911, February 2009.

[20] D. Anderson and K. Reed, "Celebrating Diversity in Volunteer Computing," in The 42nd International Conference on System Science, Hawaii, USA, 2009.

[21] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann and M. Rinard, "Managing perormance vs. accuracy trade-offs with loop perforation," in Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering., New York, NY, USA, 2011.

[22] N. Bonvin, T. Papaioannou and K. Aberer, "An economic approach for scalable and highly-available distributed applicationsIn Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, pp. 498-505. IEEE, 2010.," in the IEEE 3rd International Conference on In Cloud Computing (CLOUD), Miami, Florida, USA, 2010.

[23] F. Wagner, F. Ishikawa and S. Honiden., "Robust Service Compositions with Functional and Location Diversity," IEEE Transactions on Services Computing, no. 99, p. 1, 2013.

[24] V. Nallur and R. Bahsoon, "Design of a market-based mechanism for quality attribute tradeoff of services in the cloud," in In Proceedings of the 2010 ACM Symposium on Applied Computing, 2010.

[25] S. Zaman and D. Grosu, "A Combinatorial Auction-Based Mechanism for Dynamic VM Provisioning and Allocation in Clouds," Cloud Computing, IEEE Transactions on, vol. 1, no. 2, pp. 29-141, July-December 2013.

[26] 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, New York, NY, USA, 2010.

[27] L. Qia, W. Doua, X. Zhangc and J. Chenc, "A QoS-aware composition method supporting cross-platform service invocation in cloud environment," 1316–1329, vol. 78, no. 5, p. 1316–1329, September 2012.

[28] E. Constante, F. Paci and N. Zannone, "Privacy-Aware Web Service Composition and Ranking," in IEEE 20th International Conference Web Services (ICWS), Santa Clara ,USA 2013.

[29] J. Gutierrez-Garcia and S. Kwang-Mong, "Self-Organizing Agents for Service Composition in Cloud Computing," in 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), Indianapolis, USA, 2010.

[30] H. Markowitz, Portfolio Selection: Efficient Diversification of Investments, New York: John Wiley & Sons, 1959.

[31] J. Wang and J. Zhu, "Portfolio theory of information retrieval," in the 32nd international ACM SIGIR conference on Research and development in information retrieval, New York, NY, USA, 2009.

[32] W. F. Sharpe, Portfolio Theory and Capital Markets, New York: McGraw-Hill, 2000.

[33] H. Markowitz, "Portfolio selection.," Journal of Finance, vol. 7, no. 1, pp. 77-91, 1952.

[34] de Lemos; H. Giese; H. Müller; M. Shaw; J. Andersson;M. Litoiu; B. Schmerl; G. Tamura; N. Villegas; T. Vogel;D. Weyns; L. Baresi; B. Becker , "Software Engineering for Self-Adaptive Systems:A Second Research Roadmap," in Software Engineering for Self-Adaptive Systems, Dagstuhl, Germany, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[35] L. Duboc, E. Letier and D. S. Rosenblum, "Systematic elaboration of scalability requirements through goal-obstacle analysis," IEEE Transactions on Software Engineering, vol. 39, no. 1, pp. 119-140, 2013.

[36] R. Calheiros, R. Ranjan, C. DeRose and R. Buyya, "CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services" Grid Computing and Distributed Systems (GRIDS) Laboratory,The University of Melbourne, Melbourne,Australia, 2009.

[37] J. Gadrey, "The characterization of goods and services: an alternative approach.," Review of Income and Wealth, vol. 46, p. 369–387, 2000.

[38] W3C, "Web Services Glossary," World Wide Web Consortium, http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/, 2004.

[39] *T. Berners-Lee, R. Fielding, U. Irvine and L. Masinter, "RFC2396: Uniform form Resource Identifiers (URI): Generic Syntax," The Internet net Society, 1998.*

[40] *A. Salgatidou and T. Pilioura, "An overview of standards and related technology in web services," Distributed and Parallel Databases, vol. 12, no. 2, pp. 135-162, 2002.*

[41] *R. Richards, "Universal Description, Discovery, and Integration (UDDI)," in Pro PHP XML and Web Services, Apress, 2006, pp. 751-780.*

[42] *D. Jordan, J. Evdemon, A. Alves, A. Arkin, S. Askary, C. Barreto and B. Bloch, "Web services business process execution language version 2.0," OASIS standard, https://www.oasis-open.org/committees/download.php/21575/wsbpel-specification_public_review_draft_2_diff.pdf, 2007.*

[43] *S. Watt, "On Demand integration with web services," IBM Corporation , http://www.ibm.com/developerworks/library/ws-appint/ws-appint-pdf.pdf, 2004.*

[44] *M. Papazoglou, Web services: principles and technology, Pearson Education, 2008.*

[45] *N. Milanovic and M. Malek, "Current solutions for web service composition," IEEE Internet Computing, vol. 8, no. 6, pp. 51-59, 2004.*

[46] *P. Traverso and M. Pistore, "Automated composition of semantic web services into executable processes," in The Semantic Web – ISWC 2004, Berlin Heidelberg, Springer, 2004, pp. 380-394.*

[47] *J. Yang and M. Papazoglou, "Web component: A substrate for web service reuse and composition," in Advanced Information Systems Engineering, Berlin Heidelberg, Springer, 2002, pp. 21-36.*

[48] *P. Wong and J. Gibbons, "A process-algebraic approach to workflow specification and refinement," in Software Composition, Berlin Heidelberg, Springer , 2007, pp. 51-65.*

[49] *R. Hamadi and B. Benatallah, "A Petri net-based model for web service composition," in Proceedings of the 14th Australasian database conference, 2003.*

[50] *M. Jaeger, G. Rojec-Goldmann and G. Mühl, "Qos aggregation for web service composition using workflow patterns," in Enterprise distributed object computing conference, California, USA, 2004.*

[51] *S. Weerawarana, F. Curbera, F. Leymann, T. Storey and D. Ferguson, Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more., Prentice Hall, 2005.*

[52] *T. Oinn, M. Addis, J. Ferris, D. Marvin, A. Wipat, P. Li, T. Carver, M. Greenwood and C. Goble, "Delivering Web Service Coordination Capability to Users," in In Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, 2004.*

[54] *K. Christos, C. Vassilakis, E. Rouvas and P. Georgiadis, "Exception resolution for BPEL processes: a middleware-based framework and performance evaluation," in Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, 2008.*

[55] *W. Dou, X. Zhang, J. Liu and a. J. Chen, "HireSome-II: Towards Privacy-Aware Cross-Cloud Service Composition for Big Data Applications," Parallel and Distributed Systems, IEEE Transactions on, vol. 26, no. 2, pp. 455 - 466, 2015.*

[56] *A. Kazema, A. Pourhaji, H. Pedramb and H. Abolhassanic, "BNQM: A Bayesian Network based QoS Model for Grid service composition," Expert Systems with Applications, vol. 42, no. 20, p. 6828–6843, 2015.*

[57] *H. Jin, X. Yao and Y. Chen, "Correlation-aware QoS modeling and manufacturing cloud service composition," Journal of Intelligent Manufacturing, pp. 1-14, 2015.*

[58] *S. Mokhtar, J. Liu, N. Georgantas and V. Issarny, "QoS-aware dynamic service composition in ambient intelligence environments," in In Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering,, 2005.*

[59] *F. Rosenberg, "QoS-Aware Composition of Adaptive Service-Oriented Systems.," Ph.D. thesis, Technischen Universität Wien, Fakultät für Informatik, 2009.*

[60] *K. Christos, Costas, E. Rouvas and P. Georgiadis, "Exception resolution for BPEL processes: a middleware-based framework and performance evaluation," in In Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, Bali, Indonesia, 2008.*

[61] *K. Yang, A. Galis and H.-H. Chen, "QoS-Aware Service Selection Algorithms for Pervasive Service Composition in Mobile Wireless Environments," Mobile Networks and Applications , vol. 15, no. 4, pp. 488-501, 2010.*

[62] *H. Chang and K. Lee, "Quality-Driven Web Service Composition for Ubiquitous," in In Proceedings of the 2009 International Conference on New Trends, Washington, DC, USA,, 2009.*

[63] *S. Mokhtar, "Semantic Middleware for Service-Oriented Pervasive Computing.," PhD these, Université Pierre et Marie Curie-Paris VI, Paris, france, 2007.*

[64] *M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient QoS-aware service composition," in In Proceedings of the 18th international conference on World wide web, 2009.*

[65] *A. Klein, F. Ishikawa and S. Honiden, "SanGA: A Self-Adaptive Network-Aware Approach to Service Composition," Services Computing, IEEE Tran, vol. 7, no. 3, pp. 452 - 464, 2014.*

[53] J. Ping, Q. Mair and J. Newman, "Using UML to design distributed collaborative workflows: from UML to XPDL," in Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.

[54] K. Christos, C. Vassilakis, E. Rouvas and P. Georgiadis, "Exception resolution for BPEL processes: a middleware-based framework and performance evaluation," in Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, 2008.

[55] W. Dou, X. Zhang, J. Liu and a. J. Chen, "HireSome-II: Towards Privacy-Aware Cross-Cloud Service Composition for Big Data Applications," Parallel and Distributed Systems, IEEE Transactions on, vol. 26, no. 2, pp. 455 - 466, 2015.

[56] A. Kazema, A. Pourhaji, H. Pedramb and H. Abolhassanic, "BNQM: A Bayesian Network based QoS Model for Grid service composition," Expert Systems with Applications, vol. 42, no. 20, p. 6828–6843, 2015.

[57] H. Jin, X. Yao and Y. Chen, "Correlation-aware QoS modeling and manufacturing cloud service composition," Journal of Intelligent Manufacturing, pp. 1-14, 2015.

[58] S. Mokhtar, J. Liu, N. Georgantas and V. Issarny, "QoS-aware dynamic service composition in ambient intelligence environments," in In Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering,, 2005.

[59] F. Rosenberg, "QoS-Aware Composition of Adaptive Service-Oriented Systems.," Ph.D. thesis, Technischen Universität Wien, Fakultät für Informatik, 2009.

[60] K. Christos, Costas, E. Rouvas and P. Georgiadis, "Exception resolution for BPEL processes: a middleware-based framework and performance evaluation," in In Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, Bali, Indonesia, 2008.

[61] K. Yang, A. Galis and H.-H. Chen, "QoS-Aware Service Selection Algorithms for Pervasive Service Composition in Mobile Wireless Environments," Mobile Networks and Applications , vol. 15, no. 4, pp. 488-501, 2010.

[62] H. Chang and K. Lee, "Quality-Driven Web Service Composition for Ubiquitous," in In Proceedings of the 2009 International Conference on New Trends, Washington, DC, USA,, 2009.

[63] S. Mokhtar, "Semantic Middleware for Service-Oriented Pervasive Computing.," PhD Thesis, Université Pierre et Marie Curie-Paris VI, Paris, france, 2007.

[64] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient QoS-aware service composition," in In Proceedings of the 18th international conference on World wide web, 2009.

[65] A. Klein, F. Ishikawa and S. Honiden, "SanGA: A Self-Adaptive Network-Aware Approach to Service Composition," IEEE Transactions Services Computing, vol. 7, no. 3, pp. 452 - 464, 2014.

[66] 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.

[67] D. Ardagna and B. Pernici, "Global and local QoS constraints guarantee in web service selection," in .Proceedings of IEEE International Conference on Web Services (ICWS), Orlando , Florida , USA , 2005.

[68] M. Alrifai, T. Risse, P. Dolog and W. Nejdl, "A scalable approach for qos-based web service selection," in In Service-Oriented Computing–ICSOC 2008 Workshops, Springer Berlin Heidelberg, 2009, pp. 90-199.

[69] M. C. Jaeger, G. Mühl and S. Golze, "QoS-aware composition of Web services: a look at selection algorithms," in In IEEE International Conference on Web Services (ICWS), Orlando , Florida , USA, 2005.

[70] T. Yu and K.-J. Lin, "Service selection algorithms for composing complex services with multiple qos constraints," in In Service-Oriented Computing-ICSOC 2005, Berlin, Springer , 2005, pp. 130-143.

[71] G. . Nemhauser and L. Wolsey, Integer and Combinatorial Optimization, new york, usa: John Wiley & Sons, 1988.

[72] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam and H. Chang, "QoS-aware middleware for Web services composition," IEEE Software Engineering, vol. 30, no. 5, pp. 311 - 327, may 2004.

[73] Q. Wu and Q. Zhu, "Transactional and QoS-aware dynamic service composition based on ant colony optimization," Future Generation Computer Systems, vol. 29, no. 5, p. 1112–1119, 2013.

[74] I. Paik, W. Chen and M. N. Huhns, "A Scalable Architecture for Automatic Service Composition," Services Computing, IEEE Tran, vol. 7, no. 1, pp. 82-95, 2014.

[75] A. Elhabbash, R. Bahsoon and P. Tino, "Towards self-aware service composition," in High Performance Computing and Communications, Paris,France, 2014.

[76] 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, 2010.

[77] N. B. Mabrouk, N. Georgantas and V. Issarny, "Set-Based Bi-level Optimisation for QoS-Aware Service Composition in Ubiquitous Environments," in Web Services (ICWS), 2015 IEEE International Conference on, New York, USA, 2015.

[78] J. Lartigaua, X. Xua, L. Niea and D. Zhana, " Cloud manufacturing service composition based on QoS with geo-perspective transportation using an improved Artificial Bee Colony optimisation algorithm," International Journal of Production Research, vol. 53, no. 14, pp. 4380-4404, 2015.

[79] S. Wang, Q. Sun, H. Zou and F. Yang, "Particle Swarm Optimization with Skyline Operator for Fast Cloud-based Web Service Composition," Mobile Networks and Applications, vol. 18, no. 1, pp. 116-121, 2013.

[80] A. Bucchiarone, C. Cappiello, E. D. Nitto, R. Kazhamiakin, V. Mazza and M. Pistore, "Design for adaptation of service-based applications: main issues and requirements," in Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops, Berlin,Germany, Springer Berlin Heidelberg, 2010, pp. 467-476.

[81] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 4, no. 2, p. 14, 2009.

[82] R. Kazhamiakin, S. Benbernou, L. Baresi, P. Plebani, M. Uhlig and O. Barais, "Adaptation of service-based systems," in In Service research challenges and solutions for the future internet, Berlin ,Germeny, Springer Berlin Heidelberg, 2010, pp. 117-156.

[83] Y. Yan, P. Poizat and L. Zhao, "Self-adaptive service composition through graphplan repair," in In Web Services (ICWS), 2010 IEEE International Conference on, Miami, Florida, USA, 2010.

[84] A. Erradi and V. T. Piyush Maheshwari, "Policy-Driven Middleware for Self-adaptation of Web," in Middleware 2006, Berlin,germany, Springer Berlin Heidelberg, 2006, pp. 62-80.

[85] K. S. McCann, "The diversity–stability debate," nature, vol. 405, no. 6783, pp. 228-233, 2000.

[86] B. Baudry and M. Monperrus, "The Multiple Facets of Software Diversity: Recent Developments," INRIA and University of Lille, Lille, France, 2014.

[87] A. Avizienis and L. Chen, "On the Implementation of N-version Programming for Software Fault Tolerance During Execution," vol. se 11, no. 2, pp. 1491 - 1501 , November 1977.

[88] B. Littlewood, P. Popov and a. L. Strigini, "Modeling software design diversity: a review," Computing Surveys , vol. 33, no. 2, pp. 177-208, June 2001.

[89] I. Gashi and P. T. Popov, "Rephrasing rules for off-the-shelf SQL database servers," in Sixth European Dependable Computing Conference (EDCC), Coimbra, Portugal, 2006.

[90] H. B. Diab, A. Y. Zomaya and L. Strigini, "Fault Tolerance Against Design Faults," in Dependable Computing Systems: Paradigms, Performance Issues, and Applications, Hoboken, NJ-USA, Wiley , 2005.

[104] H. Abu-Libdeh, L. Princehouse and H. Weatherspoon, "RACS: a case for cloud storage diversity," in In Proceedings of the 1st ACM symposium on Cloud computing , New York, NY, USA, 2010.

[105] A. Bessan, i. M. Correia, B. Quaresma, F. Andre and P. Sousa, "DEPSKY: Dependable and Secure Storage in a Cloud-of-Clouds," in Proceedings of the sixth conference on Computer systems, 2011.

[106] Bonvin, Nicolas, T. Papaioannou and K. Aberer, "An economic approach for scalable and highly-available distributed applications" in IEEE 3rd International Conference on In Cloud Computing (CLOUD), Miami, Florida, USA, 2010.

[107] D. P. Anderson and K. Reed, " Celebrating diversity in volunteer computing " in Proceedings of the 42nd International Conference on System Sciences, Hawaii, 2009.

[108] A. Gorbenko, V. Kharchenko, P. Popov and A. Romanovsky, " Dependable Composite Web Services with Components Upgraded Online," in Lecture Notes in Computer Science, Springer Berlin Heidelberg, Berlin, Germany, 2005, pp. 92-121.

[109] H. E. Mansour and T. Dillon, "Dependability and rollback recovery for composite web services," IEEE Transactions on Services Computing, vol. 4, no. 4, pp. 328-339, 2011.

[110] H. Costa, M. Barros and A. Rocha, "Software Project Portfolio Selection A Modern Portfolio Theory Based Technique," in the 22nd International Conference on Software Engineering & Knowledge Engineering (SEKE'2010), Redwood City, San Francisco Bay, CA, USA, 2010.

[111] S. Awerbuch, "Portfolio-Based Electricity Generation Planning: Policy Implications For Renewables And Energy Security," Mitigation and Adaptation Strategies for Global Change, vol. 11, no. 3, pp. 693-710, 2006.

[112] I. Hwang and M. Pedram, "Portfolio Theory-Based Resource Assignment in a Cloud Computing System," in IEEE 5th International Cloud Computing (CLOUD), 2012.

[113] Z. Bodie, A. Kane and A. Marcus, Essentials of Investments, 5th edation ed., McGraw-Hill., 2004.

[114] B. B. Mandelbrot and R. L. Hudson, Misbehavior of Markets, Basic Books, 2004.

[115] G. 0. Bierwag and M. A. Grove, "Slutsky Equations for Assets," Journal of Political Econom, vol. 76 , pp. 114-126, 1968.

[116] K. Hiroshi and S. Ken-ichi, "A mean-variance-skewness optimization model," Journal of the Operations Research Society of Japan , vol. 38, no. 2, pp. 173-187, 1995.

[117] S. Royama and K. Hamada, "Substitution and Complementarity in the Choice of Risky Assets," in Risk Aversion And Portfolio Choice, new yourk, John Wiley & Sons, 1967.

[118] Mathworks, "Mean-Variance Efficient Frontier," Retrieved February 15, 2014, from mathworks.co.uk, : http://www.mathworks.co.uk/help/toolbox/finance/f10-19463.html#f10-3659.

[119] American Society for Quality, "The scatter plot provides a visual representation of the relationship between two variables," ASQ Service Quality Division, http://asq.org/service/body-of-knowledge/tools-scatter-plot, accessed on 20/1/2015.

[120] F. Peng, K. Tang, G. Chen and X. Yao, "Population-Based Algorithm Portfolios for Numerica Optimization," IEEE Transactions on Evolutionary Computation, vol. 14, no. 5, pp. 782-800, 2010.

[121] I. Hwang and M. Pedram, "Portfolio Theory-Based Resource Assignment in a Cloud Computing System," in IEEE 5th International Cloud Computing (CLOUD), Honolulu, USA, 2012.

[122] J. L. Rodgers and W. A. Nicewander, "Thirteen ways to look at the correlation coefficient," The American Statistician, vol. 42, no. 1, p. 59–66, February 1998.

[123] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," Journal of Network and Systems Management, vol. 11, no. 1, pp. 57-81, March 2003.

[124] A. Michlmayr, F. Rosenberg, P. Leitner and a. SchahramDustdar, "Comprehensive Comprehensive QoS monitoring of Web services and event-based SLA violation detection" in Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing, New York,USA, 2009.

[125] P. Zhang, Y. Zhuang, H. Leung, W. Song and Y. Zhou, "A Novel QoS Monitoring Approach Sensitive to Environmental Factors," in In Web Services (ICWS), 2015 IEEE International Conference on, New York,USA, 2015.

[126] V. Nallur, "A Decentralized Self-Adaptation Mechanism For Service-Based Applications in The Cloud," Phd Thesis, birmingham,UK, 2012.

[127] F. Alrebeish and R. Bahsoon, "Risk-Aware Web Service Allocation in the Cloud Using Portfolio Theory," in 10th IEEE International Conference on Services Computing, San Francisco, CA, USA, 2013.

[128] R. Calheiros, R. Ranjan, A. Beloglazov, C. DeRose and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and Experience, vol. 41, no. 1, pp. 23-50, 2011.

[129] S. Long and Y. Zhao, "A Toolkit for Modeling and Simulating Cloud Data Storage: An Extension to CloudSim," in IEEE international Conference on Control Engineering and Communication Technology (ICCECT), Liaoning, China, 2012.

[130] B. Wickremasinghe, R. Calheiros and R. Buyya, "Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications," in the 24th IEEE

International Conference on Advanced Information Networking and Applications (AINA) , 2010.

[131] Y. Shi, X. Jiang and K. Ye, "An energy-efficient scheme for cloud resource provisioning based on cloudsim," in IEEE International Conference on Cluster Computing (CLUSTER), 2011.

[132] R. Jeyarani, R. V. Ram and N. Nagaveni, "Design and implementation of an efficient two-level scheduler for cloud computing environment," in the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid) , 2011.

[133] G. Anastasi, E. Carlini and P. Dazzi, "Smart cloud federation simulations with CloudSim," in InProceedings of the first ACM workshop on Optimization techniques for resources management in clouds, New York, NY, USA, 2013.

[134] A. v. Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour," in Proceedings of the Fifth IEEE International Symposium on Requirements Engineering,, 2001.

[135] A. Lapouchnian, "Goal-oriented requirements engineering: An overview of the current research," University of Toronto, Toronto,Canada, 2005.

[136] D. ROSS, "Structured Analysis: A Language for Communicating Ideas," IEEE Transactions on Software Engineering, vol. 3, no. 2, pp. 16 - 34, 1977.

[137] I. Jacobsen, M. Christerson, P. Jonsson and G. Overgaard, Object Oriented Software Engineering, Boston,USA: Addison-Wesley ACM Press, 1992.

[138] A. VanLamsweerde and E. Letier, "From object orientation to goal orientation: A paradigm shift for requirements engineering.," in Radical Innovations of Software and Systems Engineering in the Future, Berlin,Germeny, Springer, 2004, pp. 325-340.

[139] L. Duboc, A framework for the characterization and analysis of software systems scalabilit., PhD Thesis, London ,UK: UCL (University College London), 2010.

[140] G. Brown, B. Cheng, H. Goldsby and J. Zhang, "Goal-oriented specification of adaptation requirements engineering in adaptive systems," in In Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems, 2006.

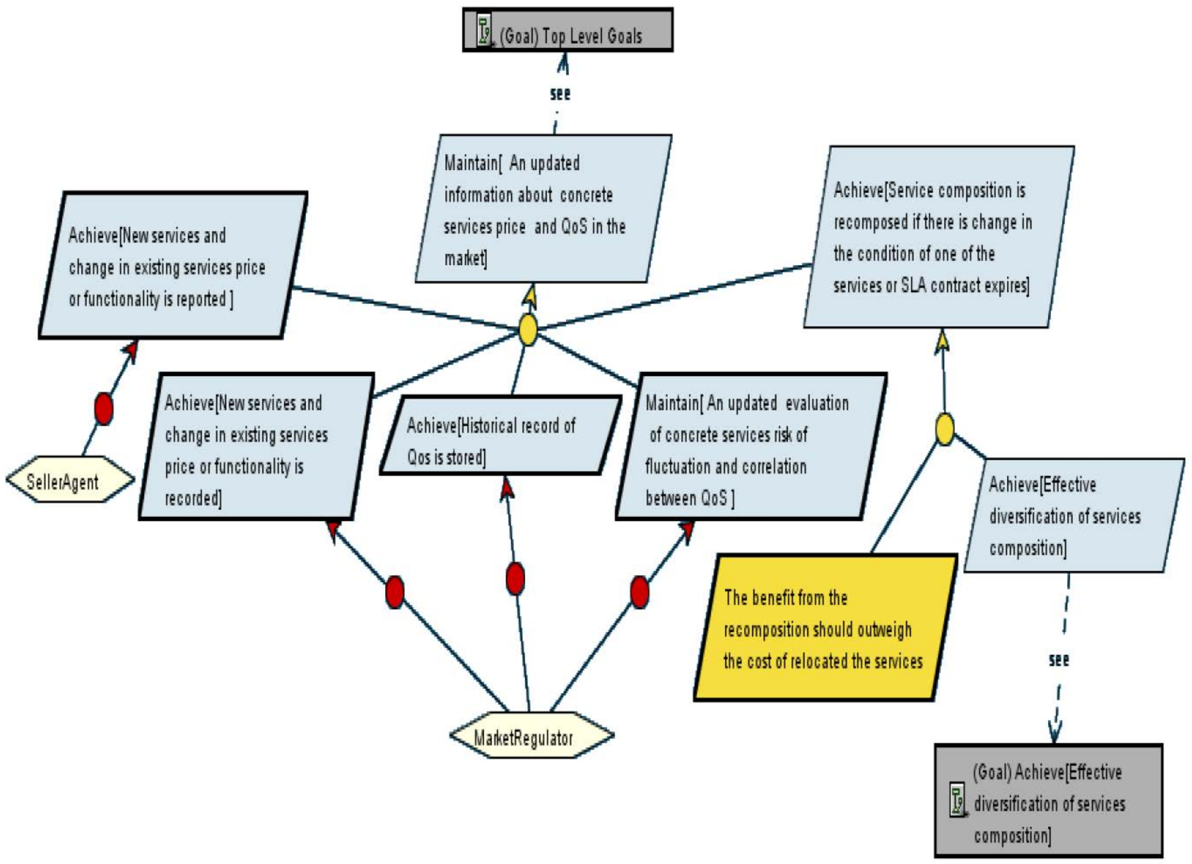
[141] United-States-Department-of-Defence, Risk Management Guide for DoD Acquisition, United States Department of Defense, 2006.

[142] F. Alrebeish and R. Bahsoon, "Stabilising Performance in Cloud Services Composition Using Portfolio Theory," in IEEE International Conference on Web Services (ICWS), New York, USA, 2015.

[143] D. Schuller, M. Siebenhaar, R. Hans, O. Wenge, R. Steinmetz and S. Schulte, "Towards Heuristic Optimization of Complex Service-based Workflows for Stochastic QoS Attributes," in IEEE International Conference on Web Services (ICWS), Alaska, USA, 2014.

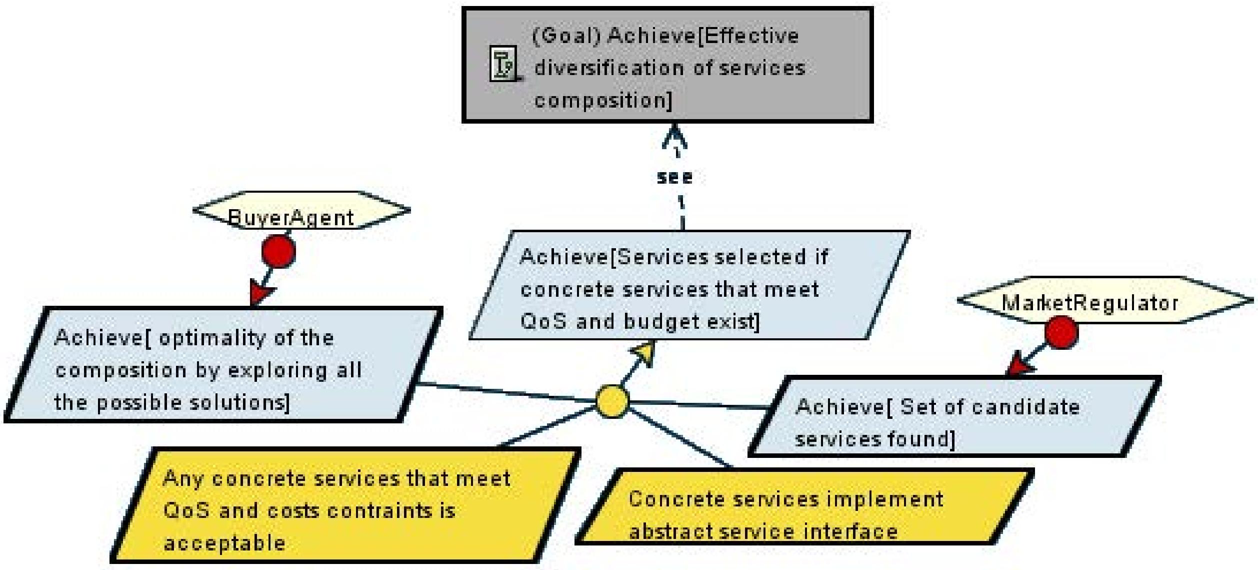
APPENDIX

A- *KAOS GOAL ORIENTED MODEL*



(Goal) Top Level Goals
see
Maintain[An updated information about concrete services price and QoS in the market]
Achieve[New services and change in existing services price or functionality is reported]
Achieve[Service composition is recomposed if there is change in the condition of one of the services or SLA contract expires]
Achieve[New services and change in existing services price or functionality is recorded]
Achieve[Historical record of Qos is stored]
Maintain[An updated evaluation of concrete services risk of fluctuation and correlation between QoS]
Achieve[Effective diversification of services composition]
The benefit from the recomposition should outweigh the cost of relocated the services
SellerAgent
MarketRegulator
see
(Goal) Achieve[Effective diversification of services composition]

Figure A1: Refinement of goal Maintain[updated information about concrete services price and QoS in the market]. As the cloud is dynamic environment, the framework must track changes of all services registered in the market. This includes attempting to maintain an updated price and QoS of each service, recording historical performance of QoS and maintaining an updated evaluation risk of fluctuation and correlation between QoS for the services. In case of change in one of the services used in a service composition or an SLA contract expires, a recomposition of services will be triggered.



(Goal) Achieve[Effective diversification of services composition]
see
BuyerAgent
Achieve[Services selected if concrete services that meet QoS and budget exist]
MarketRegulator
Achieve[optimality of the composition by exploring all the possible solutions]
Achieve[Set of candidate services found]
Any concrete services that meet QoS and costs contraints is acceptable
Concrete services implement abstract service interface

Figure A2: Refinement of goal Achieve[Services selected if concrete services that meet QoS and budget exist]: Satisfying this goal requires identifying a group of compatible candidate services, assuming that any services that satisfy the constraints is selected as candidate and that these services do indeed implement the interface they advertise. When all services are found, the are reported to the Buyer agent by the MarketRegulatoer , before they are considered in the diversification process. the diversification process, the Buyer agent explore all the possible services composition to ensure the optimality of the selected CSC.

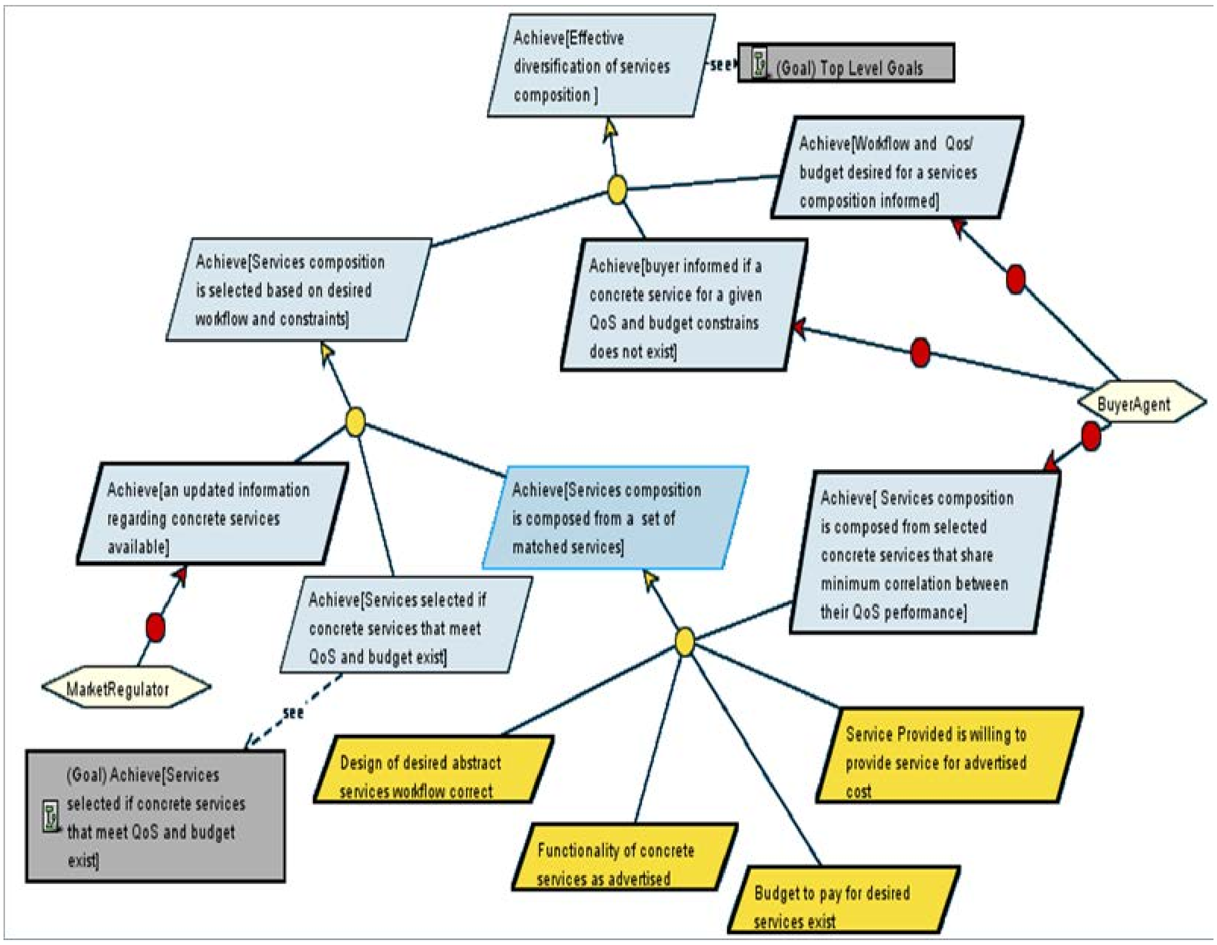
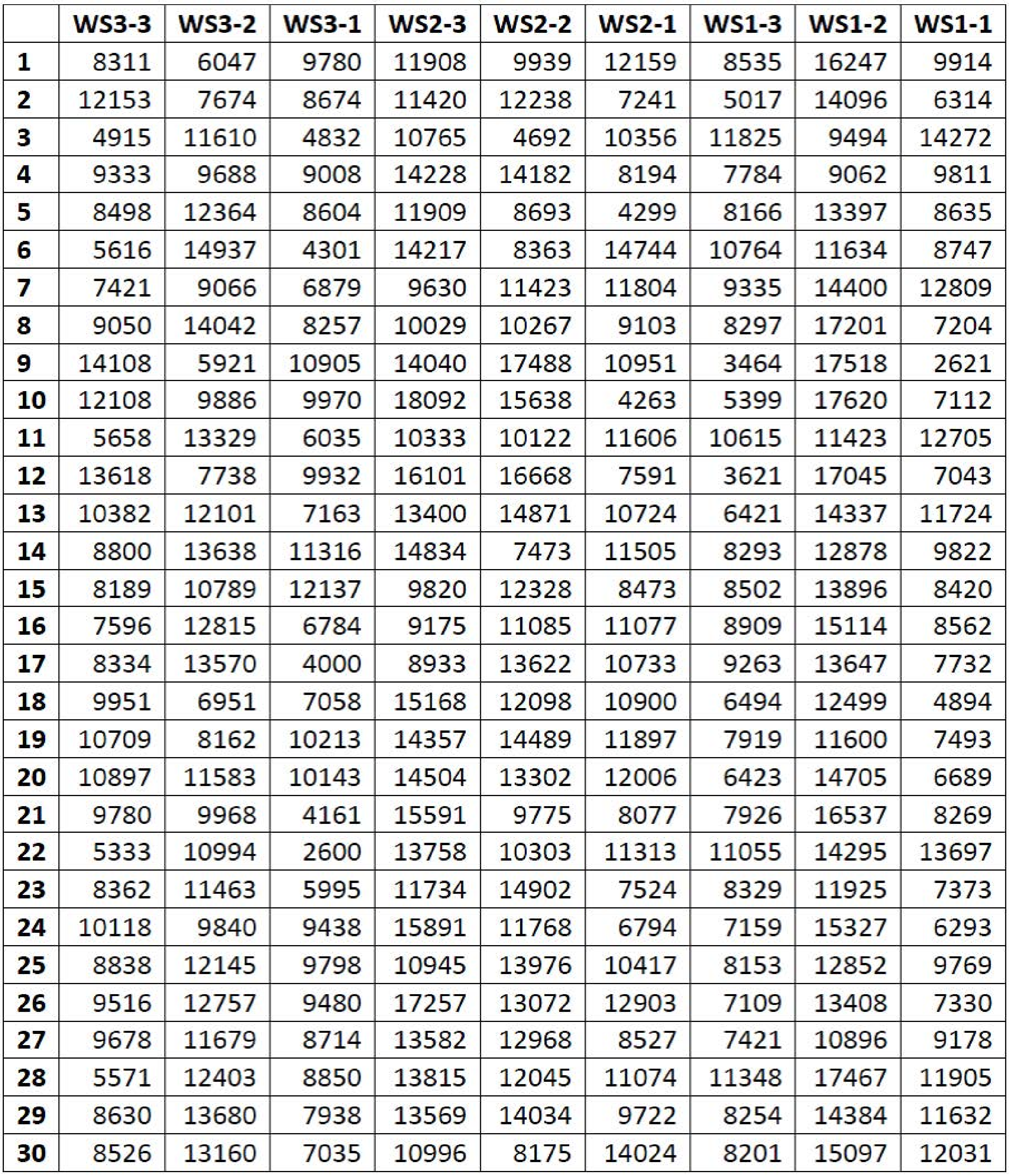

Achieve[Effective diversification of services composition]
see
(Goal) Top Level Goals
Achieve[Workflow and Qos/ budget desired for a services composition informed]
Achieve[Services composition is selected based on desired workflow and constraints]
Achieve[buyer informed if a concrete service for a given QoS and budget constrains does not exist]
BuyerAgent
Achieve[an updated information regarding concrete services available]
Achieve[Services composition is composed from a set of matched services]
Achieve[Services composition is composed from selected concrete services that share minimum correlation between their QoS performance]
Achieve[Services selected if concrete services that meet QoS and budget exist]
MarketRegulator
see
(Goal) Achieve[Services selected if concrete services that meet QoS and budget exist]
Design of desired abstract services workflow correct
Service Provided is willing to provide service for advertised cost
Functionality of concrete services as advertised
Budget to pay for desired services exist

Figure A3: Refinement of goal Achieve[Effective diversification of services composition]: In order to diversify the selection of services composition, the buyer agent need to compliment this process with information regarding the buyer preference and QoS constraint. Then, buyer preference and constraint are used to search market registry to find a compatible candidate services. When compatible candidate services cannot be found, a warning is issued to notify the buyer. However, if a group of compatible candidate services exist in the market, The buyer agent will use portfolio theory to effectively selected a set of diversified set of services.

APPENDIX

B- CLOUDSIM PROTOTYPE DATA

Table B1: Number of Users of Each Service in the CloudSim Market in Each of the 30 Runs.



	WS3-3	WS3-2	WS3-1	WS2-3	WS2-2	WS2-1	WS1-3	WS1-2	WS1-1
1	8311	6047	9780	11908	9939	12159	8535	16247	9914
2	12153	7674	8674	11420	12238	7241	5017	14096	6314
3	4915	11610	4832	10765	4692	10356	11825	9494	14272
4	9333	9688	9008	14228	14182	8194	7784	9062	9811
5	8498	12364	8604	11909	8693	4299	8166	13397	8635
6	5616	14937	4301	14217	8363	14744	10764	11634	8747
7	7421	9066	6879	9630	11423	11804	9335	14400	12809
8	9050	14042	8257	10029	10267	9103	8297	17201	7204
9	14108	5921	10905	14040	17488	10951	3464	17518	2621
10	12108	9886	9970	18092	15638	4263	5399	17620	7112
11	5658	13329	6035	10333	10122	11606	10615	11423	12705
12	13618	7738	9932	16101	16668	7591	3621	17045	7043
13	10382	12101	7163	13400	14871	10724	6421	14337	11724
14	8800	13638	11316	14834	7473	11505	8293	12878	9822
15	8189	10789	12137	9820	12328	8473	8502	13896	8420
16	7596	12815	6784	9175	11085	11077	8909	15114	8562
17	8334	13570	4000	8933	13622	10733	9263	13647	7732
18	9951	6951	7058	15168	12098	10900	6494	12499	4894
19	10709	8162	10213	14357	14489	11897	7919	11600	7493
20	10897	11583	10143	14504	13302	12006	6423	14705	6689
21	9780	9968	4161	15591	9775	8077	7926	16537	8269
22	5333	10994	2600	13758	10303	11313	11055	14295	13697
23	8362	11463	5995	11734	14902	7524	8329	11925	7373
24	10118	9840	9438	15891	11768	6794	7159	15327	6293
25	8838	12145	9798	10945	13976	10417	8153	12852	9769
26	9516	12757	9480	17257	13072	12903	7109	13408	7330
27	9678	11679	8714	13582	12968	8527	7421	10896	9178
28	5571	12403	8850	13815	12045	11074	11348	17467	11905
29	8630	13680	7938	13569	14034	9722	8254	14384	11632
30	8526	13160	7035	10996	8175	14024	8201	15097	12031

Table B2: Response Time In Seconds For Each Service In The Cloudsim Market in Each of the 30 Runs.

	WS3-3	WS3-2	WS3-1	WS2-3	WS2-2	WS2-1	WS1-3	WS1-2	WS1-1
1	7.4618	5.4	8.7204	11.8882	9.8902	12.088	10.192	19.4244	11.8704
2	10.878	6.8328	7.7316	11.3888	12.188	7.1936	6	16.788	7.554
3	4.4052	10.4288	4.3164	10.6894	4.5956	10.2898	14.1484	11.2708	17.026
4	8.361	8.6316	8.1	14.186	14.086	8.092	9.2324	10.794	11.7504
5	7.5528	11.058	7.7316	11.8882	8.5916	4.196	9.7122	15.9468	10.3116
6	5.0348	13.3952	3.8658	14.186	8.2918	14.6854	12.8294	13.9088	10.4316
7	6.6528	8.1	6.1136	9.5912	11.3888	11.7884	11.151	17.2704	15.3488
8	8.1	12.588	7.372	10	10.19	9.091	9.832	20.6232	8.6352
9	12.6762	5.3042	9.7992	13.988	17.3828	10.8892	4.0768	20.985	3.1176
10	10.878	8.8104	8.901	17.984	15.5848	4.196	6.3548	21.104	8.513
11	5.0348	11.9568	5.4	10.2898	10.09	11.5888	12.7096	13.6692	15.2274
12	12.2268	6.9224	8.901	16.084	16.5836	7.495	4.3176	20.384	8.394
13	9.2598	10.878	6.383	13.3868	14.7856	10.6894	7.6752	17.1458	14.0286
14	7.9116	12.2268	10.1588	14.7856	7.3928	11.489	9.832	15.3488	11.7504
15	7.2828	9.6194	10.878	9.7904	12.2878	8.392	10.192	16.5468	10.0728
16	6.75	11.5076	6.0234	9.091	10.99	10.99	10.6712	18.105	10.192
17	7.4618	12.141	3.598	8.8912	13.5872	10.6894	11.0312	16.3072	9.2324
18	8.901	6.2034	6.294	15.085	11.992	10.8892	7.6752	14.868	5.76
19	9.6194	7.2828	9.1704	14.2858	14.3864	11.7884	9.4722	13.9088	8.8728
20	9.7128	10.339	9.08	14.486	13.2868	11.992	7.6752	17.6256	7.914
21	8.7204	8.901	3.686	15.485	9.6904	7.996	9.4722	19.785	9.832
22	4.7648	9.7992	2.3376	13.6864	10.2898	11.2888	13.19	17.026	16.3072
23	7.4618	10.2492	5.3042	11.6884	14.8852	7.495	9.9518	14.2682	8.7528
24	9.08	8.8104	8.4508	15.7844	11.6884	6.6934	8.513	18.345	7.434
25	7.9116	10.878	8.7204	10.8892	13.8862	10.3904	9.7122	15.3488	11.6304
26	8.541	11.4174	8.4508	17.1832	12.988	12.8872	8.513	16.0668	8.7528
27	8.6316	10.4288	7.8216	13.487	12.8872	8.492	8.8728	12.9504	10.911
28	4.945	11.148	7.9116	13.7864	11.992	10.99	13.5488	20.8632	14.2682
29	7.7316	12.2268	7.1022	13.487	13.988	9.6904	9.832	17.1458	13.9088
30	7.642	11.777	6.294	10.8892	8.092	13.988	9.832	18	14.4