TECHNICAL DEBT-AWARE AND EVOLUTIONARY ADAPTATION FOR SERVICE COMPOSITION IN SAAS CLOUDS

by

SATISH KUMAR

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
October 2020

ABSTRACT

The advantages of composing and delivering software applications in the Cloud-Based Software as a Service (SaaS) model are offering cost-effective solutions with minimal resource management. However, several functionally-equivalent web services with diverse Quality of Service (QoS) values have emerged in the SaaS cloud, and the tenant-specific requirements tend to lead the difficulties to select the suitable web services for composing the software application. Moreover, given the changing workload from the tenants, it is not uncommon for a service composition running in the multi-tenant SaaS cloud to encounter under-utilisation and over-utilisation on the component services that affects the service revenue and violates the service level agreement respectively. All those bring challenging decision-making tasks: (i) when to recompose the composite service? (ii) how to select new component services for the composition that maximise the service utility over time? at the same time, low operation cost of the service composition is desirable in the SaaS cloud. In this context, this thesis contributes an economic-driven service composition framework to address the above challenges. The framework takes advantage of the principal of technical debt- a well-known software engineering concept, evolutionary algorithm and time-series forecasting method to predictively handle the service provider constraints and SaaS dynamics for creating added values in the service composition. We emulate the SaaS environment setting for conducting several experiments using an e-commerce system, realistic datasets and workload trace. Further, we evaluate the framework by comparing it with other state-of-the-art approaches based on diverse quality metrics.

Keywords: Economic-Driven, Service Composition, SaaS Cloud, Technical Debt

I would like to dedicate this thesis to my guru ji

Professor Sanjay Jasola

Vice-Chancellor

Graphic Era Hill University

Dehradun, India

ACKNOWLEDGEMENTS

I would like to express my earnest obligation and special appreciation to my supervisor Dr Rami Bahsoon because this complex journey couldn't be envisaged to be completed without his inspiration, persistence and profound knowledge. His indefatigable attitude always fostered me to bring the best out of me. I will always cherish those moments as the golden period of my life because I got the opportunity to learn a lot from him. I feel blessed to have Professor Rajkumar Buyya (Director - CLOUDS Laboratory, University of Melbourne, Australia) and Dr Tao Chen (Loughborough University, UK) as my co-supervisor. Both of them are my consistent source of motivation in the findings of the results from scratch till end. Words won't be able to suffice their contributions.

Secondly, I would like to offer my cordial gratitude to the thesis group members Professor Uday Reddy, and Dr Per Kristian Lehre, their visionary approach, insightful comments, constructive criticism and valuable feedback guided me to the right direction of research work. Along with this, I want to thank Dr Ke Li (University of Exeter, UK), Mr Prashant Kumar Bamania, Cloud Architect, HCL Technologies, India and Mr Virendra Kaushik, Software Architect, Adobe Systems, India they also played a vital role in the development and progress of this project by giving their valuable inputs.

Apart from this, I won't forget to say thanks to the team members of Birmingham Software Engineering Research Group; Vahab Samandi, Francisco Ramirez, Alexandros Evangelidis, Suwichak Fungprasertkul, Paola Yanez, Hargyo Tri Nugroho, Dr Carlos Joseph Mera Gomez, Dr Sara Hassan, Dr Dalia Sobhy and Rajeev Ranjan Singh (Security Group) who were always there for me both technically and emotionally. We spent hours and hours discussing the minute details of topics, and I was always the beneficiary of some extraor-

dinary thought process from the debates and research discussions. I am also pleased to say thanks to my academic mentors and friends, Dr Rajesh Mishra, Dr Mange Ram, Dr Vimal Kumar, Dr Avadhesh Kumar Gupta; Rajkumar, Piyush Dixit, Kuldeep Singh, Laxmi Kant Dhariwal, Anil Kumar, Vinod Sherwal, Pradeep Saini who were always there for me during odd and even times of my journey. You all really mean a lot to me.

Last but not least, I am grateful to the almighty, my beloved parents, and siblings. My family always stood beside me both financially and emotionally. Their prayers, motivation, and moral support helped me to achieve this milestone. While discussing my loved one's I can't forget the vital contribution of my loving and caring life partner Dr Priya. Generally, wives offer sympathy and empathy to their husband, but I found myself lucky as I am also the recipient of scrupulous advice, both academically and technologically.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Overview

The growing popularity of cloud-based service models, mainly Software as a Service (SaaS) has emerged as a standard model for delivering software application [1]. In such a model, service composition is a promising technique for building a software application by composing multiple existing web services [2]. The resulting software application, namely composite service, is often deployed in the cloud, forming the basics of modern SaaS cloud. A pronounced benefit of deploying composite service in the SaaS cloud is the realisation of multi-tenancy, in which a single application instance can serve several tenants[1] simultaneously [3]. On the other hand, the SaaS provider has its own optimisation goals for delivering software application in the SaaS cloud [4]. For example, Salesforce [5] provides a Sales CRM (Customer Relationship Management) service in the cloud market. Further, Salesforce keeps the good reputation of Sales CRM service in the cloud market by delivering better application performance and ensuring the minimum Service Level Agreement (SLA) violations, at the same time, they want to maximise the service revenue at a lower operation cost (e.g., infrastructure execution cost). However, uncertainty is the common property in the dynamic environment like SaaS cloud. In particular, uncertainty in the request workload generated by the tenants may affect the composite service

[1]Tenants denote the end-users in the SaaS cloud.

performance; because of each participating component service[2] in the composition has a different capacity to process n requests per seconds [6]. More importantly, it may cause the problem of under-utilisation or over-utilisation on the component service with respect to their capacities. Consequently, the increasing workload can rise the over-utilisation for the component services within a composite service, which in turns, would negatively affect the Quality of Service (QoS) and violates the SLA [3]. On the other hand, the decreasing workload may lead to under-utilisation of the capacity of component services, reducing the revenue that should have been achieved as the infrastructural resources also impose a monetary cost. All those bring challenging tasks: (i) when to recompose the composite service? (ii) how to select new component services for the composition that maximise the service utility over time? at the same time, the low operation cost of the service composition is desirable in the SaaS cloud.

In this thesis, we address the above challenges by taking advantage of technical debt metaphor [7, 8, 9] that supports an economic-driven decision on the selection and (re)composition of composite service in the SaaS cloud. In particular, we argue that the technical debt could be the consequences of taking imperfect or poorly justified run-time service recomposition decisions. Further, in the SaaS cloud environment, the utilisation of component services participating in composite service execution may be sub-optimal due to significant up and drop in the requests workload generated by tenants. The sub-optimal service composition leads the debt to provide higher capacity service than the tenant's service demand. Consequently, the operating cost may outweigh the service revenue. Furthermore, under-/over utilisation of component service may incur interest over the debt. For example, over-utilisation of service capacity leads the SLA violation and the penalty cost against the response time violation can be count as interest over the technical debt. Additionally, the interest makes the negative impact on the service utility and continuously grow over the technical debt for a given service execution time. Therefore, we view the technical debt as a time-sensitive moving target that needs to be

[2]Component service denotes the individual web service in the service composition.

dynamically monitored to estimate the incurred debt and its nature (good or bad) over service execution. The estimated cost of technical debt facilitates to optimise the present service utility by making a proactive decision on whether to keep the current service composition plan or recompose the new composite service. Previous work by Alzaghoul and Bahsoon [10] has exploited the technical debt metaphor in the design-time service selection, where they used real options and technical debt analysis to justify the added value of a selected service from the cloud marketplace. Though run-time composition and recomposition are critical decisions, which should be judged from added values, technical debt and economics perspectives, this work is fundamentally different: (i) We focus on the problem of composition and recomposition at run-time; (ii) We use run-time and predictive learning techniques to (re)compose the service in the SaaS cloud.

1.2 Problem Statement

The development of modern software delivery systems (e.g., SaaS cloud) for providing composite software application over the Internet has many benefits and service quality risks. The former implies that the SaaS cloud offers cost-effective solutions by supporting the multi-tenancy system architecture, in which a single application instance service the many users simultaneously [4, 11]. The latter indicates that the dynamic nature of the SaaS cloud tends to lead the uncertainty in an operating environment and the composite service running under such operating environment always has some risks, e.g., performance, scalability, delayed latency, service availability and SLA violation [6, 12, 13]. In this environment, composing and deploying a composite application is more complicated than a traditional service composition environment, for example, it is common in the SaaS cloud; several tenants may demand a diverse functional and QoS requirements for the same service [5]. But, most of the current service composition approaches are designed to optimise the service composition plan for a single end-user system [4]. However, these approaches have their own limitations and would not be efficient for emerging comput-

ing paradigms such as SaaS cloud. To deal with the multi-tenants requirements, He et al. [4] presented an approach that optimises different QoS of service execution plans (with similar functionalities) for all the tenants in the SaaS cloud. However, these approaches tend to ignore the fact that multi-tenant execution environment needs to provide variant service execution plans, each offering a customised plan for the given tenants with its functionality, QoS and cost constraints.

Moreover, uncertainties may arise in the operating environment from the dynamic changing workload generated by tenants on the composite application in the SaaS cloud [6]. Consequently, the composite application encounters under-utilisation and over-utilisation on the component services that affect the service revenue and violates the SLA. In fact, both cases are undesirable, and it is, therefore, nature to mitigate them by recomposing the services to a newly optimised composition plan once they have been detected. The current recomposition approaches were triggered in the response of QoS constraints violation or component service failure during the execution [14, 15, 16, 17]. These approaches also partially support the SaaS provider optimisation goals (e.g., maintaining QoS constraints) without considering other essential objectives (e.g., maximise the service revenue and minimise an operating cost). However, they ignored the fact that temporary effects can merely cause under-/over-utilisation, and thus the advantages may be short-term, which hinders the long-term benefits that could have been created by the original composition plan while generating unnecessary overhead and disturbance via recomposition.

In this context, this thesis aims to address the stated limitations and the value-added facts in the composition that are ignored in the existing work. We address the following problems driven by the existing research works.

- **Problem 1 :** The limited support of economic-driven perspective in the service composition in SaaS cloud.

- **Problem 2 :** The limited support to process different service composition plans for the tenants with varied functional, QoS and cost constraints.

- **Problem 3 :** The lack of runtime economic-driven decision approach that can dynamically and predicatively track the value-added benefits in the service composition, while composite service encounters the under/over-utilisation on the component services.

- **Problem 4 :** The lack of strictness of soft and hard constraints on the different levels of service composition and the limited support of long-term based economic-driven service selection in the dynamic service composition.

1.3 Research Questions

In this thesis, we address the following research questions

- **RQ1 :** *Reviewing state-of-the-art service composition approach and identifying the research gaps in the area of economic-driven service composition in the SaaS cloud* − What is the state-of-the-art service composition approaches with a particular interest in economic aspects? What are the pending research challenges in an economic-driven service composition in the SaaS cloud?

- **RQ2 :** *Realising diverse functional and QoS requirements from the tenants in the SaaS cloud* − How can we leverage an evolutionary algorithm to support dynamic optimisation of multi-tenant service compositions in the SaaS Cloud?

- **RQ3 :** *Realising economic-driven service recomposition decisions in the changing workload from the tenants in the SaaS cloud* − How can we leverage the technical debt metaphor to support an economic-driven decision for dynamic service recomposition in SaaS Cloud? How can the use of predictive analytics of workload improve the decision making and evaluates the service debt?

- **RQ4 :** *Realising strictness of soft and hard constraints on the different levels of service composition* − How can a debt-aware two-levels constraints reasoning of a

service selection create the long-term values in the self-adaptive service composition in the SaaS cloud?

1.4 Research Methodology

To address the above research questions, in this thesis, we adopt the classical research methodology presented by Peffers et al. [18]. The following steps were carried out to lead our research in this thesis.

- **Identifying the thesis problem :** The first step is to get better knowledge of the dynamic service composition, particularly an economic-driven perspective in the SaaS cloud environments. For this purpose, we conducted a systematic literature review that helped to gain a good understanding of the field and guided us to identify the room for improvement in the current state-of-the-art service composition approach. Further, based on the understanding of the problem domain and the key findings, the problem of this thesis is identified and formulated in the form of the research questions.

- **Identifying the thesis objective :** Based on the identified problems, the next step is to define the objective of its solution. We formulated these objectives in the form of Research Questions in the Section 1.3.

- **Design and Development :** A review study is conducted for service composition with the particular focus of an economic-driven perspective in the SaaS cloud. Based on the results obtained from the survey, we are in the position to identify the inadequacies of the service composition research in the SaaS cloud, specifically, for evaluating the run-time decision of service composition/recomposition. In this regard, we adopt an evolutionary algorithm and technical debt metaphor for optimising multi-tenant service composition and contributing an economic-driven decision approach respectively in the SaaS cloud. In particular, technical debt metaphor

facilitates to identify, estimate and monitor the good and bad values (e.g., added values or impacts) in the service composition. We developed the debt-aware middleware framework that introduces technical debt as a novel metric for analysing the composite service execution in the SaaS cloud environment.

- **Demonstration :** We implement an economic-driven framework for service composition in the SaaS cloud. In particular, to emulate an environment of SaaS cloud, we deployed 100 web services over 10 Docker containers and each web service exhibits the different QoS which is associated with real-world WSDream dataset [19]. We developed an e-commerce system which is formed as service composition. Further, the e-commerce system is executed under the real-world requests workload collected from the FIFA 98 World Cup trace [20] and a time-series forecasting method named Autoregressive Fractionally Integrated Moving Average model (ARFIMA) [21] is employed for predicting the workload on the service composition.

- **Evaluation :** The proposed approach is quantitatively evaluated by comparing with other state-of-the-art approach based on the diverse quality metrics. We used diverse metrics which are specific to individual contribution in this thesis. Particularly, these metrics include the running overhead, aggregated utility, operating cost, accumulated debt, global utilization and latency for comparing the performance of the approaches.

1.5 Thesis Contributions

The research presented in this thesis shows the significant contributions in the area of service composition, specifically in the multi-tenant SaaS cloud environment. In particular, the main contribution of this thesis is to present an economic-driven framework for service composition in the SaaS cloud. Notably, this thesis presents the following research contributions.

1. **Systematic Literature Review on Service Composition Based on the Service Quality Factors :** We conduct a systematic literature review that provides a better understanding of the state-of-the-art of service composition approach. Further, we present a classification framework of existing service composition approaches based on the service quality factors and also discuss the techniques and algorithms taken by these approaches. We identify the research gap in current service composition approaches that allows us to address the research gap in the form of thesis research questions.

2. **A Diverse Requirements-Driven Multi-Tenant Service Composition in the SaaS Cloud :** We formulate the multi-tenant service composition as an optimisation problem. In particular, we present a new encoding representation and fitness function that explicitly model the service selection and composition as an evolutionary optimisation for the multi-tenant SaaS cloud. For this, we adopted an evolutionary algorithm as the driver for implementing service composition engine that optimises the different service composition plans for the tenants they have varied functional, QoS and cost constraints in the SaaS cloud environment.

3. **Technical Debt-Aware Adaptive Service Recomposition in SaaS Cloud :** We propose a new concept and model, namely service debt that explicitly maps the technical debt metaphor in the context of service composition. The proposed service debt model enhanced by the time-series prediction method that allows us to build a utility model for identifying and estimating the debt in the service composition. Overall, we combined all components and developed a holistic debt-aware framework for recomposing service in the SaaS cloud, namely DebtCom. Particularly, the DebtCom framework is capable of making an economic-driven decision on whether to trigger recomposition or not, considering the long-term benefits.

4. **Self-Adapting Service Composition with Debt-Aware Two Level Constraints Reasoning :** We present a **D**ebt-**A**ware **T**wo l**E**vels con**S**traint reasoning

for **S**elf-adapt-ing service c**O**mposition (DATESSO) framework that leverages debt-aware two level constraints reasoning for self-adapting service composition. Instead of formalising the constraints at both local and global as hard ones, we refine the global constraints as the soft one. This has enabled us to tailor the reasoning process in self -adaptation and mitigate over-optimism. Specifically, we design an efficient two level-constraint reasoning algorithm in DATESSO that is debt-aware and utilises different strictness of the two level constraints to reduce the search space.

1.6 Publications

The following research papers [6, 22, 23, 24, 25] are published/in-submission during PhD research. Moreover, this thesis is an absolute reference of the details discussions, formulation of ideas and core contributions presented in the following research publications.

- **S.Kumar**, R. Bahsoon, T. Chen, K. Li, and R. Buyya. Multi-Tenant Cloud Service Composition using Evolutionary Optimization. In the Proceeding of 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), Singapore, 2018.

- **S.Kumar**, R. Bahsoon, T. Chen, and R. Buyya. Identifying and Estimating Technical Debt for Service Composition in SaaS Cloud. In the Proceeding of 2019 IEEE 24th International Conference on Web Services (ICWS), Italy, 2019.

- **S.Kumar**, T. Chen, R. Bahsoon, and R. Buyya. DATESSO: Self-Adapting Service Composition with Debt-Aware Two Levels Constraints Reasoning. In the Proceeding of 2020 IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), South Korea, 2020. [**SEAMS Best Student Paper Award**]

- **S.Kumar**, R. Bahsoon, T. Chen, and R. Buyya. DebtCom: Technical Debt-Aware Service Recomposition in SaaS Cloud. IEEE Transactions on Service Computing

(TSC), 2020. [Review Cycle]

- **S.Kumar**, R. Bahsoon, T. Chen, and R. Buyya. A Systematic Review and Taxonomy on Service Composition based on the Service Quality Factors. ACM Computing Surveys (CSUR), 2021. [To be submitted]

1.7 Thesis Roadmap

Figure 1.1 illustrates the structure of the thesis roadmap as follows

- **Chapter 2 :** *Reviewing state-of-the-art service composition approach and identifying the research gaps in the area of economic-driven service composition in the SaaS cloud* – In this chapter, we present a systematic literature review on the service composition and the findings derived from the study is to enable us to introduce a taxonomy of services composition approaches in the SaaS cloud. Firstly, we present the general area of service composition including web service technology and then exploring the dynamic aspects of service composition using different strategies such as QoS/SLA/Constraints/Requirements aware approaches. We then indicate the research gaps which we address in this thesis from the finding of this study. This chapter is derived from our research paper (in preparation for submission) [25].

- **Chapter 3 :** *Realising diverse functional and QoS requirements from the tenants in the SaaS cloud* – In this chapter, we introduce a novel multi-tenant service composition approach. First, we model the problem as an evolutionary optimisation problem with a new encoding representation and the fitness function. We incorporate our approach in MOEA/D-STM [26] and develop an evolutionary optimisation based service composition engine. Further, MOEA/D-STM based service composition engine supports different types of users requests and optimise the service composition plan for each category of users in a multi-tenant SaaS cloud. We have also incorporated this encoding representation in NSGA-II [27] for performing the comparative

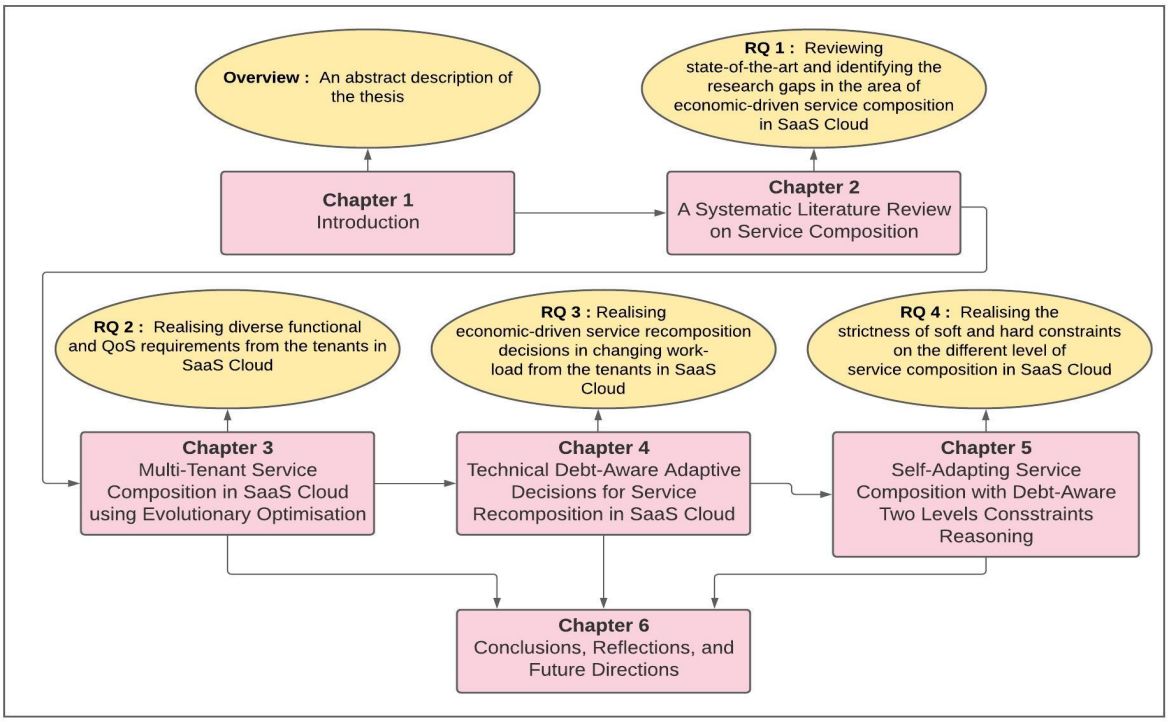


Figure 1.1: Thesis Roadmap

study. This chapter is derived from our research work published in [23]. In particular, the aim of developing this service composition engine is to accelerate our DebtCom framework presented in Chapter 4.

- **Chapter 4 :** *Realising economic-driven service recomposition decisions in the changing workload from the tenants in the SaaS cloud* – In this chapter, we contribute to an economic-driven decision approach for dynamic service recomposition leveraging the principle of technical debt. We provide a systematic connection between technical debt and service composition and discuss some critical situations that incurred technical debt in service composition. Further, as the solution, we present technical debt-aware adaptive decision-making approach which is implemented in our proposed novel DebtCom middleware architecture. This chapter is derived from our research work published/under review in [6] [24].

- **Chapter 5 :** *Realising strictness of soft and hard constraints on the different levels of service composition* – In this chapter, we first motivate the need for considering

the different strictness of two-level constraints using a motivating scenario of an e-commerce system. And then, we model these constraints for monitoring the composite service execution environment. Further, we develop a temporal debt-aware utility model that select the suitable component service for repairing the infeasible component services during the execution. This chapter is derived from our research work published in [22].

- **Chapter 6 :** *Reflections, future directions and conclusion remarks* – In this chapter, we conduct a reflective evaluation on the each contribution of this thesis using different qualitative criteria (e.g., running overhead, performance metrics, datasets and practical deployment). Further, we discuss future research directions and summarise the main contributions of this thesis.

NOTE : The algorithms presented in this thesis are not evaluated from the theoretical computer science perspective. In particular, Software Engineering and Service-Oriented Computing communities follow the experimental approach, in which, the performance of an algorithm is evaluated under different types of datasets. This thesis follows the experimental approach for analysing and testing of our proposed algorithm's performance under real-world datasets such as WSDream [19] and FIFA 98 World Cup Trace [20].

CHAPTER 2

A SYSTEMATIC LITERATURE REVIEW ON SERVICE COMPOSITION

In this chapter, we contribute a systematic literature review and taxonomy on service composition. In particular, this study aims to provide a better understanding of the field and review the state-of-the-art practices of service composition in the dynamic environment, e.g., SaaS cloud. We briefly discuss the background and the concept of technical debt metaphor, and then present this metaphor in the context of service composition. Further, we provide a classical taxonomy of the existing service composition approaches based on the service quality factors. We also discuss the limitations of current service composition approaches from an economic-driven perspective in the SaaS cloud environment. Overall, this study identifies the research gaps in the current service composition approaches and allows us to address these research gaps in the form of research questions.

2.1 Introduction

Service composition is a key technology that allows individual web services to be combined together to create a new value-added service, e.g., composite application [13]. Over the Internet, many functionally equivalent web services are available with different Quality of Service (QoS) values [28, 29]. The selection of these web services for composing a composite application is very challenging due to different QoS requirements. Further, an end-user may impose several conditions (e.g., normal QoS, hard QoS constraints, or

SLA specification) as part of the service demand in the cloud market [30, 31]. Due to such complex requirements, researchers have started to explore the service composition from different perspectives such as QoS-aware service composition, Constraint-aware service composition, SLA-aware service composition, or context-aware service composition etc. [32, 33, 34, 35, 36] . Moreover, to satisfying such complex requirements in the service composition, they transformed the service composition problem into a single-objective optimisation and multi-objective optimisation problems [2, 13, 37, 38, 31, 39, 40].

Further, deploying a composite application in a dynamic environment like SaaS cloud tends to lead many challenges, for example, composite service would inevitably operate under the dynamic changes on the workload that affects the service performance, cause constraints or SLA violations or reduce the service utility. Researchers investigated these issues; in which they proposed an adaptive service composition or composite service reconfiguration approaches [14, 30, 41, 42] in the context of QoS constraints or SLA violations, and service failure etc. They have ignored the service provider optimisation goals (e.g., less operating cost and maximum service revenue). There is a little work done in this direction [4, 10], but their approaches have not supported and taken the full advantages of the dynamic environment, e.g., SaaS cloud.

Further, to get the depth understanding of service composition and finding the current state of art issues in the field. We aim to conduct a Systematic Literature Review (SLR) that (i) provides a state-of-the-art service composition approach and a classification framework of the current service composition approaches based on the service quality factors, (ii) provide a comprehensive discussions on the techniques/methods taken by these approaches for dealing with the service quality factors in the composition, and also discuss an economic-driven perspective for service composition in the SaaS cloud, (iii) identifying the research gaps for conducting future research. Specifically, in this chapter, we address the first research question of our thesis:

RQ1: *Reviewing state-of-the-art service composition approach and identifying the research gaps in the area of economic-driven service composition in the SaaS cloud* − **what**

is the state-of-the-art service composition approaches with a particular interest in economic aspects and what are the pending research challenges in an economic-driven service composition in the SaaS cloud?. For making a concrete focus on each point in the research question, we further split it into following sub-research questions.

RQ 1.1: What are the current state-of-the-art approaches for service composition[1], and what are the quality of service factors addressed in these approaches?. *This question aims to provide a classification framework of existing service composition approaches based on service quality factors (e.g., QoS, SLA, Constraints, or uncertainty ect.).*

RQ 1.2: What are the current techniques/methods taken by these approaches for supporting service quality factors and what is the current status of an economic-driven approach in the SaaS Cloud? *This question aims to provide an extensive discussion on the current techniques/methods that were used for implementing the service quality factors based service composition and an adoption of the SaaS-based economic-driven service composition.*

RQ 1.3: What are the future directions in an economic-driven service composition research, in particular SaaS cloud? *This question provides the state-of-the-art discussions and gives useful insight into how we can benefit from the existing service composition approaches to draw the motivation on the key requirements and discussing the pitfalls when applying these service composition approaches in the SaaS cloud.*

2.1.1 Preliminaries and Basic Concepts

2.1.1.1 Web Service

Web service is a software module that supports interoperable machine-to-machine communication over the Internet through standard protocol [43]. Particularly, in the Service-Oriented Architecture (SOA), web services can be defined as a self-described, self-contained and modular unit of application that can be published, located and dynamically invoked

[1]service composition and web service composition are used interchangeably

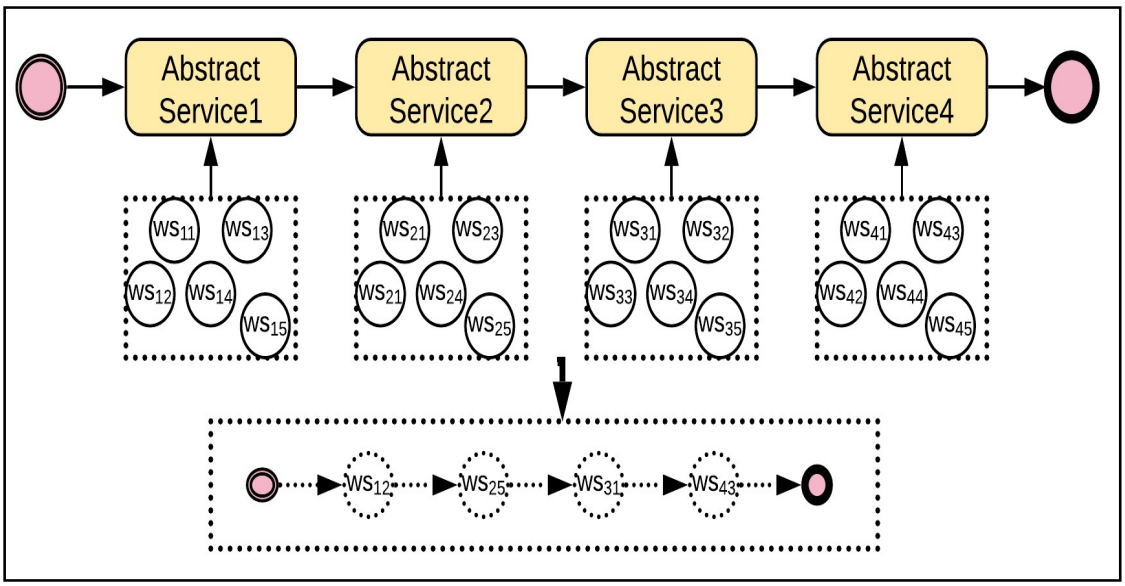


Figure 2.1: Application Workflow

over the Internet for developing a web-based application [44, 45].

2.1.1.2 Service Composition

In the service-based systems, a complex application can be defined as a process of invoking the suitable web service selected at run-time [46]. In this scenario, service composition is a logical combination of multiple abstract services resulting into a single unit (e.g., software application) for performing complex requests submitted by the users in the multi-tenant SaaS cloud as shown in the Figure 2.1. An abstract service can be realised by a set of candidate component services [47, 48], each of which comes with different capacities to process n requests per second. Further, when a user submits the request for a service in the SaaS cloud, the application engineer selects one suitable service from each candidate service set and optimise the service composition plan that satisfies the user's demands.

2.1.1.3 Software as a Service (SaaS) Cloud Model

SaaS is a cloud-based software application delivery model over the Internet [49]. A service consumer can access these software applications through a web browser. In particular, SaaS cloud offers a cost-effective solution through adopting multi-tenancy system ar-

chitecture, in which the single application instance simultaneously serves multiple users based on the shared resources [11, 50]. For example, deploying a composite application in the SaaS cloud takes the full advantages of multi-tenancy architecture in terms of least management of application resources, and delivering an economic-driven service to the end-users.

2.1.1.4 Technical Debt

Technical debt can be attributed to sub-optimal decisions, shortcut on decisions, and/or deferred activities that can incur extra cost/rework, if it would be carried in the future as when compared the current time [6, 51]. Technical debt metaphor was initially coined by Cunningham in 1992 to explain *"Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. The danger occurs when the debt is not repaid. Every minute spent on not quite right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise"* [7]. In the recent years, software engineering community presented this metaphor and discussed its applicability to many software artifacts, covering code, requirements, architecture, testing and documentation, among the other [9, 52, 53]. The common understanding is that technical debt is the result of making technical compromises that are expedient in the short-term but that create a technical context that increases complexity and cost in the long term [54, 55]. If these technical compromises are not paid back than technical debt may be incurred and degrade the system quality or the development team productivity in the long term. By incurring technical debt is not always bad, if organization makes informed decisions or strategic reasons about to incur the debt [56]. McConnell [57] classified the term "technical debt" into *intentional technical debt* and *unintentional technical debt*. An *intentional technical debt* is the debt which is taken by an organization to optimise the present value in the software project rather than the future value or to make informed decisions for gaining short-term benefits. On the other hand, *unintentional*

technical debt can be incurred unknowingly when an organization makes non-strategic or inappropriate decisions in the software project.

2.1.1.5 Service Debt

One form of Service Debt can be attributed to sub-optimal utilisation of composite service capacity due to an imperfect decision of web services selection in the composition. We provide a systematic formulation of service debt in Chapter 4.

2.2 Systematic Literature Review Process

In this section, we describe the Systematic Literature Review (SLR) protocol that will help to develop the SLR process. Then, we conduct the SLR to compile the primary studies related to service composition based on the review criteria and objectives.

2.2.1 Review Protocol

We developed a review protocol based on the guidelines and procedure proposed by Kitchenham [58, 59]. In particular, our review protocol comprises review objective and background, research questions , search process (e.g., search terms and bibliographical resources), selection criteria (e.g., inclusion and exclusion), search execution, quality assessment and data extraction and synthesis method. We have reported the review objective, background, and research questions in the Section 2.1 and the remaining protocol procedures are explained in the subsequent sections.

2.2.2 Search Process

In the search process, we selected the primary studies that were published from 2002 to 2019. The search process could be performed either an automatic search using bibliographical resources (e.g., IEEE Xplore) or the manual search that retrieves the research articles from the specified conferences and journals in the field of study. In particular,

we used well-known research venues for conducting manual search such as IEEE ICWS, IEEE TSC, and ICSOC etc. Moreover, our search process used the following points to identify the relevant studies.

- Applying the preliminary search that aims to identify the existing systematic literature review.
- Performing trail searches with a different combinations of search terms obtained from the research questions.
- New search terms were obtained by swapping singular to plural forms or plural to singular forms.
- Conducting further trail searches that examine the related references and ensures that we have not missed any significant study.
- Applying "Quasi-Gold Standard (GGS)" [60] in the manual scan process of well-known venues (e.g., conferences and journal) recognised by the research community in the field of study.

Further, ensuring the QGS standards in our search process, we included the paper's title, abstract and keywords in the manual search process and then conducting the manual and an automatic searches. The results obtained from both search strategies were examined based on the coverage differences. However, we observed that the manual search results were covered in the automatic results, that indicated the coverage of QGS standards.

2.2.2.1 Search Terms Selection

We identified the search terms as guided by [61] and supported the research questions. Further, the collected set of search terms were the same for all the research questions because it minimises the duplicate search terms. The search strings were constructed by considering the abbreviations, synonyms, and alternative spelling. Moreover, advanced

search strings were created by boolean AND or OR operators on the identified search terms. We have picked the most appropriate and related search terms that assist our SLR study. Therefore, five search terms were chosen; "Economic", "Driven", "Web Service", "Composition", and "Approach". The search query was created using these five major search terms: Economic **AND** Driven **AND** Web Service **AND** Composition **AND** Approach.

Towards developing more sophisticated search queries, we identified several challenges in automatic search due to miss leading of the service quality attributes in the other field of studies (e.g. network management service or mobile service quality). To mitigate these problems, we further classified the service quality requirements either end-user based requirements (e.g., QoS, SLA, and constraints) or environment-driven service quality (e.g., location, context and uncertainty etc.). Besides, the terms service selection, web service selection, and service composition were used interchangeably in the existing research.

Finally, generating the main search query, we also used these keywords by connecting boolean OR as follows.

(QoS **OR** SLA **OR** Constraint **OR** Constraints **OR** Context **OR** Uncertainty **OR** Uncertainties **OR** Service **OR** Selection) **AND** Economic **AND** Driven **AND** (Web Service **OR** Webservices) **AND** Composition **AND** Approach **OR** SaaS.

We conducted both the manual search and an automatic search. From these searches, we observed that the manual search is not an efficient method than automatic search because it encounters the difficulties to manually filtering thousands of the results which were retrieved in the search. But, we still accepted the manual search process to meet the quasi-gold standards that ensure the good practice of search queries in this review.

2.2.2.2 Selection of Bibliographical Sources

Table 2.1 provides an overview of the selected electronic databases that give confidence to covering most of the high impact journals and conference proceedings. However, these electronic databases enable us to conduct advanced search queries with a variety of search

Table 2.1: Bibliographical Sources

Bibliographical Source	**URL**
IEEE Xplore	https://ieeexplore.ieee.org/Xplore/home.jsp
ACM Digital Library	https://dl.acm.org/
ScienceDirect	https://www.sciencedirect.com/
SpringerLink	https://link.springer.com/
Google Scholar	https://scholar.google.com/

options and constraints such as time frame constraints, boolean operations, title, keywords and author etc. We have used Google Scholar as a data source because some significant research studies were not found in the first four electronic databases. We have performed a pilot test that indicated that the first 300 results were significantly relevant to this review domain. For maintaining generality and reliability of the results, we extended the Google scholar resulting coverage up to 500 results.

2.2.3 Selection of Inclusion and Exclusion Criteria

We performed the inclusion and exclusion criteria on the peer-reviewed journals, conferences, and workshops papers published between 2002 and 2019. The reason for choosing this time frame was that the earliest notable research work in the field of service composition was published in 2004 [2]. In the following subsection, we provide a detailed discussion of applying an inclusion and exclusion criteria based on the review's scope.

2.2.3.1 Inclusion Criteria

In this criteria, we only included the research papers published in the English languages in the peer-reviewed journals, conferences and workshops. Following points were applied for developing inclusion criteria.

- Research papers published between 2002 and 2019.
- Research papers that explicitly demonstrate the service composition or service selection approaches.

Table 2.2: Number of search results from each digital databases

Bibliographic Source	Search Results
IEEE Xplore	2368
SpringerLink	1245
ACM Digital Library	1702
ScienceDirect	652
Google Scholar	500

- Research papers that explicitly provide techniques, methods, solutions and evaluations for supporting service composition.

2.2.3.2 Exclusion Criteria

In this criteria, we excluded the research papers that were either published or appeared in the books, white papers, poster session, technical report, panel report or discussion, and tutorial summary etc. Following points were applied for developing exclusion criteria.

- Research papers that did not explicitly provide techniques or approaches to facilitate the service composition.
- Non-peer-reviewed research papers.
- Research papers' language was other than English.
- Research papers that were not available in full text.

2.2.4 Search Execution

We performed the search execution process, as shown in the Figure 2.2, follow the procedure discussed in the Section 2.2.2. In our manual search, we determined the set of 17 research articles for comparing with automated search results (e.g., quasi-gold standard). Further, we conducted an automatic search on the selected databases (Table 2.1) using search query constructed in the Section 2.2.2.1. In particular, 6467 research articles were collected through search query applied on the title, keywords, and abstract in the automated search process, as shown in the Table 2.2. In the next step, we filtered the set

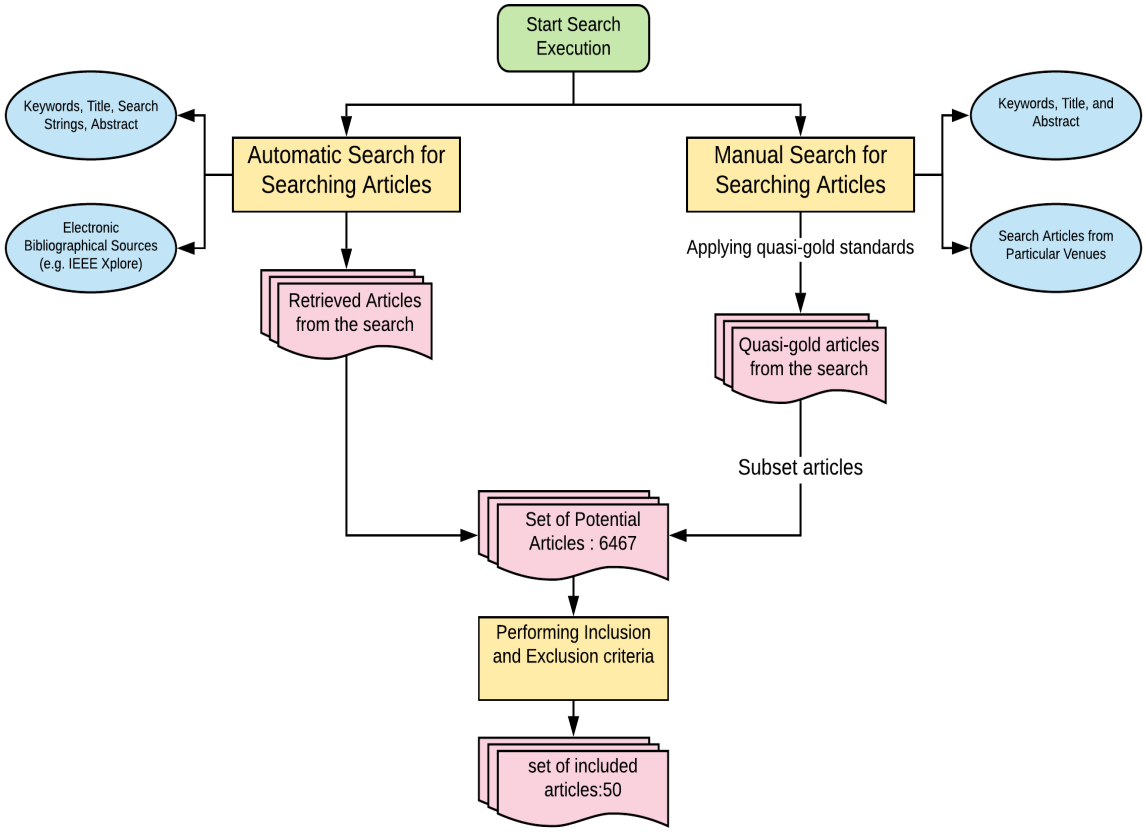


Figure 2.2: Search execution procedure

of 50 most relevant research articles in this SLR by applying an inclusion and exclusion criteria defined in the Section 2.2.3.

2.2.5 Quality Assessment and Data Extraction

Apart from the inclusion and exclusion criteria, we adopted the following quality assessment criteria provided [62]. We guarantee that each research article included in this review has met the following four criteria [62].

1. Research article has rigorous data analysis based on evidence or theoretical reasoning rather than non-justified or ad hoc statements.

2. Research article explains the context in which the research was carried out.

3. The design and execution of the research support the aims of the research article.

4. Research article contains a description of data collection methods.

The data extraction process was carried out by scanning each of the 50 research articles thoroughly, and we used the Excel spreadsheet for managing extracted data. Further, we conducted data analysis process, in which we examined the extracted data with respect to their similarities. The results discussion of this process is provided in the next section.

2.2.6 Overview of the Included Studies

2.2.6.1 An overview of the distribution of an included articles along with data sources

We included 50 research articles that were published in the leading conferences, journals and workshops in the Service-Oriented Computing (SOC) or Service Computing community and all these articles meet the quality assessment criteria that we discussed in the previous section. In the Table 2.3, we give an overview of the distribution of an included research articles with respect to their publication channels and the number of research articles included from each publication channel. Table 2.5 provides the an overview of the included research articles in this SLR. We have also plotted the publication channels along with included research articles in the Figure 2.3. The results are fairly distributed over conference, journal and workshop. Moreover, we observed that most of the articles were published in the conference (48%), followed a journal (48%) and limited articles published in the workshop (4%). However, these results reflect the maturity and active research in the field of service composition.

2.2.6.2 Citation status of included articles

In Table 2.4, we give a summary of the citation rate of the included articles in this SLR. These citation numbers were collected from the Google Scholar. The purpose of collecting this citation data is to provide a rough indication to know the research article quality, and not for making comparison study among them. In particular, 5 research articles were

Table 2.3: An overview of the distribution of research articles with publication details (e.g., Conference, Journal, and Workshop)

Sources (Conference/Journal/Workshop)	**No. of Articles**
IEEE International Conference on Web Services	6
IEEE Transactions on Service Computing	6
IEEE International Conference on Service Computing	5
International Conference on Service-Oriented Computing	3
International World Wide Web Conference	3
Expert System with Applications	1
IEEE Transactions on Software Engineering	1
Service-Oriented Computing & Application	2
IEEE Congress on Services	1
Journal of Systems and Software	2
ACM Transactions on Web	1
IEEE Transactions on Systems, Man, and Cybernetics: Systems	2
IEEE Access	1
Information and Software Technology	2
International Semantic Web Conference	1
International Journal of Web and Grid Services	1
International Workshop on Managing Technical Debt	1
Computer & Industrial Engineering	1
IEEE International Conference on Cloud Computing	1
IEEE Transactions on Knowledge and Data Engineering	1
IEEE Congress on Evolutionary Computation	1
IEEE Transactions on Cloud Computing	1
Information Sciences	1
2012 Ninth International Conference on Information Technology-New Generations	1
International Workshop on Quality of Service	1
Information Systems and E-Business Management	1
2009 IEEE Conference on Commerce and Enterprise Computing	1
Australian Software Engineering Conference	1

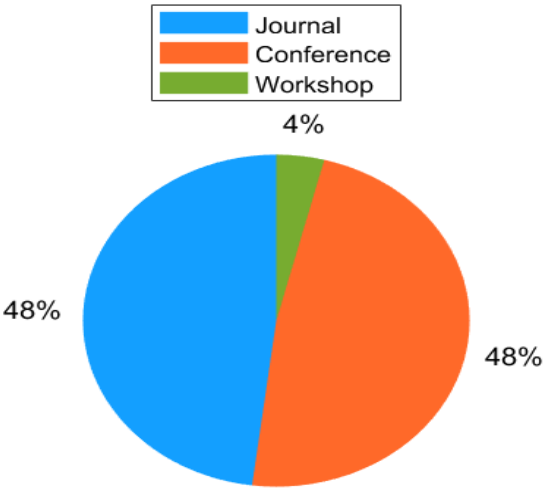


Figure 2.3: Article distribution over publication channel

Table 2.4: An overview of included research articles with citation rate

Cited By	**<10**	**10-50**	**50-100**	**>100**
No. of articles (50 articles)	5	15	12	18

cited by less than 10 other sources, and among them, most of the articles were published between 2017 and 2019. Therefore, it is expected that these articles can not get a high number of citations in such a short time span. Further, 10-50 other sources cited almost 30% of the research articles (15 articles) and 12 articles followed by 50-100 other sources. Notably, 36% of the research articles (18 articles) were cited by more than 100 other sources.

2.2.6.3 Distribution of included articles over year

Figure 2.4 provides an overview of the distribution of included articles over the year. We observed an increasing trend of the articles published in the web service composition (or web service selection) area from 2008 to 2016. In general, it shows an active research interest of the SOC community for exploring service composition in the large scale systems (e.g., cloud computing).

Table 2.5: List of included research articles for the SLR

Source	Year	Title
Zeng et al. [2]	2004	QoS-aware middleware for web services composition
Keidl et al. [63]	2004	Towards context-aware adaptable web services
Yu et al. [64]	2005	Service selection algorithms for Web services with end-to-end QoS constraints
Maamar et al. [65]	2005	Toward an agent-based and context-oriented approach for web services composition
Berbner et al. [48]	2006	Heuristics for qos-aware web service composition
Hassine et al. [?]	2006	A constraint-based approach to horizontal web service composition
Yu et al. [66]	2007	Efficient algorithms for Web services selection with end-to-end QoS constraints
Canfora et al. [15]	2008	A framework for QoS-aware binding and re-binding of composite web services
Wada et al. [67]	2008	Multiobjective optimisation of sla-aware service composition
Hwang et al. [17]	2008	Dynamic web service selection for reliable web service composition
Halima et al. [68]	2008	A qos-oriented reconfigurable middleware for self-healing web services
Xiong et al. [33]	2008	Sla-based service composition in enterprise computing
Alrifai et al. [69]	2009	Combining global optimisation with local selection for efficient QoS-aware service composition
Huang et al. [47]	2009	An optimal QoS-based Web service selection scheme
Leitner et al. [70]	2009	Runtime prediction of service level agreement violations for composite services
Zhai et al. [41]	2009	SOA middleware support for service process reconfiguration with end-to-end QoS constraints
Kapitsaki et al. [71]	2009	Model-driven development of composite context-aware web applications
Lin et al. [72]	2009	An efficient approach for service process reconfiguration in SOA with end-to-end QoS constraints
Alrifai et al. [28]	2010	Selecting skyline services for QoS-based web service composition
Leitner et al. [73]	2010	Monitoring, prediction and prevention of sla violations in composite services
Lin et al. [14]	2010	The design and implementation of service process reconfiguration with end-to-end QoS constraints in SOA
Tang et al. [74]	2010	A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition
Lin et al. [75]	2011	A relaxable service selection algorithm for QoS-based web service composition
Li(li2011adaptive)	2011	Adaptive QoS-aware service process reconfiguration
aschoff et al. [42]	2011	QoS-driven proactive adaptation of service composition
Wu et al. [76]	2012	Tree-based search algorithm for web service composition in SaaS
Lin et al. [36]	2012	Dynamic service selection based on context-aware QoS
Benouaret et al. [77]	2012	Selecting skyline web services from uncertain qos
Ye et al. [78]	2012	QoS-aware cloud service composition based on economic models
He et al. [4]	2012	QoS-driven service selection for multi-tenant SaaS
Leitner et al. [79]	2013	Data-driven and automated prediction of service level agreement violations in service compositions
Wang et al. [35]	2013	Constraint-aware approach to web service composition
Feng et al. [80]	2013	Dynamic service composition with service-dependent QoS attributes
Alzaghoul et al. [10]	2013	CloudMTD Using real options to manage technical debt in cloud-based service selection
Deng et al. [81]	2014	Service selection for composition with QoS correlations
Wu et al. [82]	2014	Broker-based SLA-aware composite service provisioning
Wen et al. [83]	2014	Probabilistic top-K dominating services composition with uncertain QoS
Alzaghoul et al. [84]	2014	Evaluating technical debt in cloud-based architectures using real options
Ding et al. [85]	2015	A transaction and QoS-aware service selection approach based on genetic algorithm
Wang et al. [86]	2015	Automatic web service composition based on uncertainty execution effects
Mostafa et al. [13]	2015	Multi-objective service composition in uncertain environments
Chen et al. [87]	2016	A flexible QoS-aware Web service composition method by multi-objective optimisation in cloud manufacturing
Chen et al. [38]	2016	Multi-objective service composition with QoS dependencies
Xu et al. [88]	2016	Context-aware QoS prediction for web service recommendation and selection
Laleh et al. [30]	2017	Constraint adaptation in Web service composition
Wang et al. [11]	2017	Efficient QoS-aware service recommendation for multi-tenant service-based systems in cloud
Sun et al. [89]	2018	A fluctuation-aware approach for predictive web service composition
Niu et al. [90]	2019	Towards the optimality of QoS-aware web service composition with uncertainty
Liang et al. [32]	2019	QoS-aware web service composition with internal complementarity
Zhang et al. [91]	2019	Composition Context-Based Web Services Similarity Measure

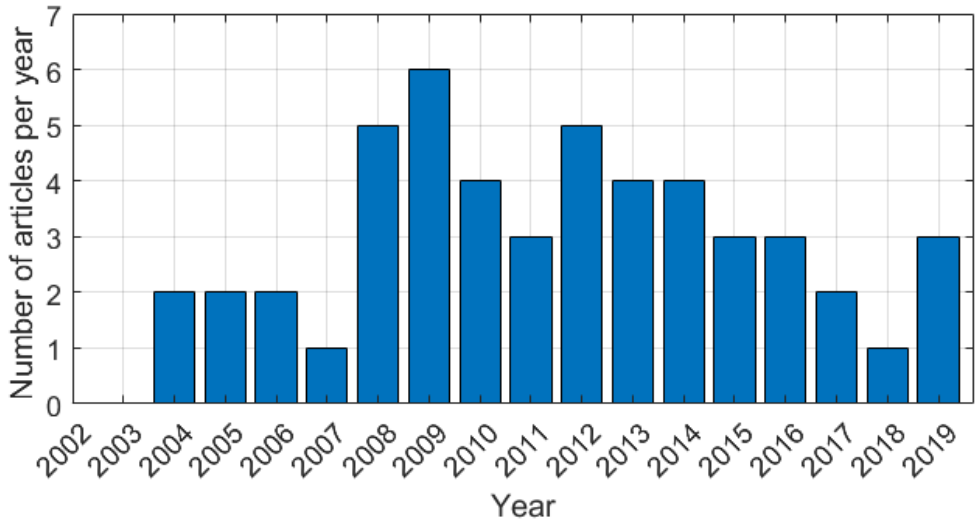


Figure 2.4: Articles distribution over the years 2002−2019

2.3 Data Extraction Results Discussion

In this section, we answer the sub-research questions defined in Section 2.1:

RQ 1.1: *What are the current state-of-the-art approaches for service composition?, and what are the service quality factors addressed in these approaches?* ; **RQ 1.2:** *What are the current techniques/methods taken by these approaches for supporting service quality factors and what is the current status of an economic-driven approach in the SaaS Cloud?* However, the existing systematic literature reviews and surveys [92, 93, 94, 95, 96, 97] supported us to design and develop the following classification framework.

2.3.1 RQ 1.1: A Classification Framework for Service Composition

In this section, we provide a classification framework for existing service composition approaches as shown in the Figure 2.5. The aim of developing this classification framework is to answer the RQ 1.1.

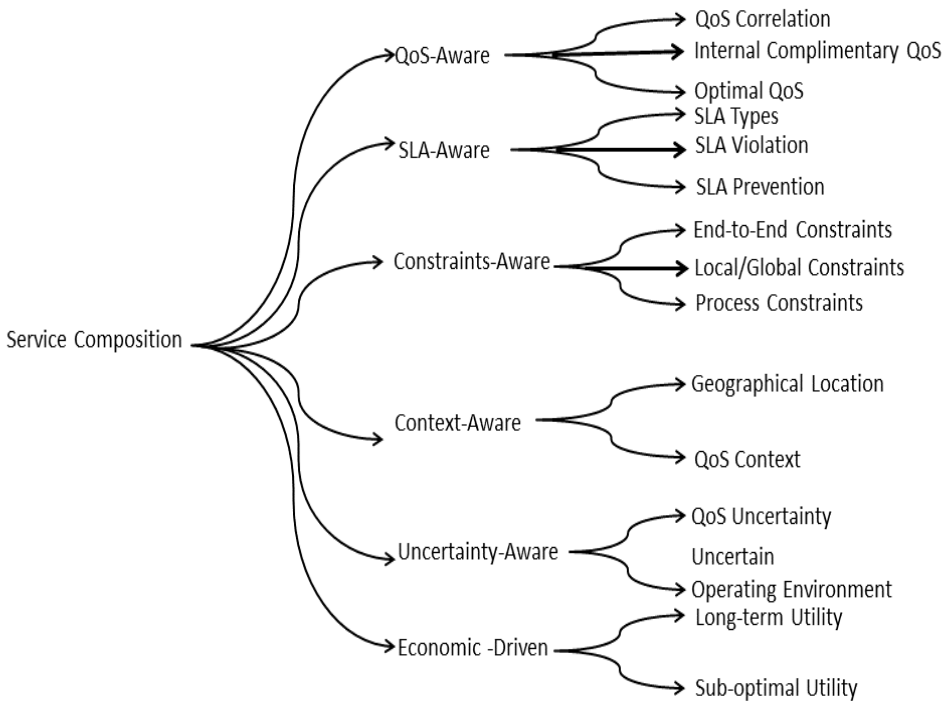


Figure 2.5: Taxonomy of service composition approaches

2.3.1.1 QoS-Aware Service Composition

QoS-aware service composition approaches are explicitly considered the end-user QoS requirements in the composition process [48, 98]. In general, QoS is used to describes the non-functional properties of web services and also play a key role for differentiating similar functional equivalent web services over the Internet [38]. The aim of web service selection in service composition is not only to meet an end-user functional requirements but also guarantee the service QoS throughout composite application execution. However, several quality of service factors are involved in the QoS-aware composition process, which may be imposed by service provider or an end-user [47].

- **QoS Correlation:** QoS correlation denotes the correlation of QoS values in the service composition [38, 80, 81], and it is common in the real-life service compo-

sition. Suppose, there are three service providers, namely A, B and C they offer similar functional web services with different quality of services (e.g., response time, throughput, or cost) in the cloud market. Let's consider a service provider A provides a discount if the candidate service Ax and Ay are selected together in the composition. On the other side, service provider B gives a discount if candidate service Bx and Cy are selected together due to some business collaboration between service providers B and C. In this scenario, the selection of candidate services exhibit the QoS correlation in the service composition.

- **Internal Complimentary QoS:** Liang et al. [32] introduced the concept of internal complimentary in the QoS-aware web service selection. Internal complimentary allows multiple candidate services selection within the same service class to form a new composite *candidate service* in the service composition. However, such candidate service selection improves the overall QoS of the service composition plan.

- **Optimal QoS:** It refers to a QoS-aware selection of suitable web services from the service repository and optimises a service composition plan that maximises an end-user satisfaction in terms of QoS requirements [47].

2.3.1.2 SLA-Aware Service Composition

The SLA-aware service composition approach is intended to build the composite application based on the type of SLA and its requirements [67]. This approach is more sophisticated than the QoS-aware service composition because the QoS-aware approach aims to meet an end-user's QoS requirements or to maximise an end-user satisfaction by providing higher capacity service instances [82]. The QoS-aware approach blocks the service resources and fails to take into account that other existing users are requesting similar services and their decisions for service composition can impact the QoS [82]. On the other hand, the SLA-aware approach provisions the service instance in the composite service demanded by multiple end-users based on their SLA types.

- **SLA Type :** In the SaaS cloud environment, end-users can be differentiated based on their SLA type. Moreover, end-users may have different SLA[2] (e.g., silver, gold or platinum) [67] that exhibits the different level of functional and QoS requirements for the same service.

- **SLA Violation :** SLA violation is the consequences of not satisfying the service constraints, which are legally bound in the SLA [34, 82]. The penalty cost would be paid against each request violation in the composition, and it would negatively affect the service utility and reputation in the SaaS cloud.

- **SLA Prevention :** SLA prevention denotes as a process of proactive prevention of SLA violation using some predictive techniques. It helps to improve the composite service utility.

2.3.1.3 Constraints-Aware Service Composition

This approach implements the constraints usually imposed by either service provider or service consumer (e.g., end-user) [46]. In general, constraints denote the minimum expectations (or strict conditions), for example, a service consumer may specify the constraints, such as service response time should be below 10 ms. Further, the service provider has its own optimisation goals [4], such as the utilization of service capacity should be above 85% throughout execution. However, all web services in the real world could not be universally applicable, and there might be some process restrictions (or constraints) imposed by the service provider [35]. For example, the process constraints could be related to a service location. The following service quality factor are involved in composing constraints-driven service composition.

- **End-to-End QoS Constraints :** End-to-End constraints guarantee that the composite service or newly recomposed service should meet an end-user's specified constraints throughout execution time [14].

[2]SLA is a legal agreement binding between the service provider and the user.

- **Local and Global Constraints :** Local constraints are applied at the individual component service selection, whereas global constraints are specified for the entire service composition [99].

- **Process Constraints :** Process constraints denote web service restrictions, e.g., let us take an item delivery service in an e-commerce system, there are many functionally equivalent web services they can process item delivery. But these web services have restrictions to deliver the item in the specified zone (or location) in the country or around the world [35].

2.3.1.4 Context-Aware Service Composition

This approach employs context-awareness, which refers to information about the geographical service environment and end-users relevant information (e.g., end-user's preferences) [100]. Context-aware information is essential for web service selection, e.g., the physical distance between an end-user and web service could be viewed as a context related factor for the service response time [36]. Moreover, a context could be derived from the parameters correlation that exists between the input parameter and out parameter of the invoked web services in the service composition [91]. The parameter correlation can be extracted from the service composition history that was used for selecting reliable web service in the service composition.The following quality factors are associated with the context-aware service composition.

- **Geographical location :** Due to different geographical location of web services and end-users, the same web service provides the different response time for end-users they have varied physical distance from the web service location [36].

- **QoS Context :** User's preferences and the service operating environment are combined together to model end-user QoS context for composing specific context-driven service composition [63].

2.3.1.5 Uncertainty-Aware Service Composition

Previous approaches are intended to form the composite service based on the prior knowledge of QoS attributes. Besides, they ignore the fact that the QoS values are uncertain in the dynamic environment (e.g., cloud environment) where no prior knowledge of the QoS attributes are available [13]. Uncertainty-aware service composition is designed to deal with QoS uncertainty in the service operating environment. In the SaaS cloud environment, QoS uncertainty could be the consequences of changing workload, service failure or error-prone operating environment etc. [17, 86, 90]. Moreover, this approach uses probabilistic and skyline techniques to deal with an uncertainty in a dynamic service composition environment [83]. The following factors cause uncertainty issues in the service composition.

- **QoS Uncertainty :** The QoS uncertainty is a common problem in a dynamic environment [6, 101] due to unpredictable and changing request workload from the users, e.g., users may frequently join or leave the service pool in the cloud environment.
- **Uncertain Operating Environment :** The uncertainty of physical or technical failure in an operating environment may lead the severe issues related to service composition, e.g., service down or degrade the service quality (e.g., performance) [17]. These issues need to mitigate promptly, otherwise, they can lower the service reputation in the cloud market.

2.3.1.6 Economic-Driven Service Composition

An economic-driven approach applies economic aspects for building a composite application. An economic aspects may be related to either individual component service selection or the entire service composition [10]. Moreover, the aim of an economic-driven service selection is to maximise the utility of the entire service composition (e.g., maximise service revenue) [10]. The following service quality factors are involved in the selection of an economic-driven service composition.

Table 2.6: Classification of Service Composition Approaches

Approach	Service Quality Factors	Representative Examples
QoS-Aware	QoS Dependency or Correlation, Internal Complimentry, Optimal QoS	[2, 4, 11, 15, 28, 32, 38, 42, 47], [48, 68, 69, 76, 80, 81], [85, 87, 98]
SLA-Aware	SLA Type, SLA Violation, SLA Prevention	[33, 67, 70, 73, 79, 82]
Constraints-Aware	End-to-End QoS constraints, Local/Global, Constraints, Process Constraints	[14, 30, 35, 41, 64, 66], [72, 74, 99]
Context-Aware	Geographical Location, QoS-Context	[36, 63, 65, 71, 88, 91]
Uncertainty-Aware	QoS Uncertainty, Uncertain Operating Environment	[13, 17, 77, 83, 86, 89, 90, 102]
Econmic-Driven	Long-Term Utility, Sub-Optimal Utility	[10, 78, 84]

- **Long-term utility :** The selection of individual web service is considered not only the QoS constraints but also long-term economic-driven perspective [78]. For example, predictive utility could be estimated from the utility history of web services. Also, this prior knowledge of predictive utility helps to select an economic-driven service that maximises the long-term utility of entire service composition.

- **Sub-optimal utility :** Sub-optimal utility is the consequences of the running composite service that may be under-utilised or over-utilised due to environmental conditions. Both cases negatively affect the service utility (e.g., reducing service revenue).

Table 2.6 summarises the service composition approaches and the relevant service quality factors that we studied above. However, most of the above approaches are further extending to deal with the dynamic adaptation of composition and reconfiguration/re-composition of composite service in the error-prone operating environment or constraints violations [17, 103]. As we discussed, web service failure is a common problem in the error-prone operating environment. Therefore, the current composite service needs to reconfigure by replacing the faulty services with the newly selected web services using these approaches. In this direction, the End-to-End QoS constraint-aware composite service reconfiguration and adaption approaches were presented by [14, 41, 72, 104]. Further, QoS-aware approach has gained more attention in this direction, the researcher viewed the

service reconfiguration in a different way such as QoS-aware service reconfiguration [16, 68] and QoS-aware-adaptation of service composition [42, 98, 105] and also applied it in the multi-tenant SaaS cloud environment [3, 4]. We will provide more discussions on these approaches in the RQ 1.2.

Answering RQ1.1: From the above discussions, we are in the position to state that the Service composition is an active research area in the Service-Oriented Computing (SOC) community. The results obtained from the classification framework suggests that the SOC community has a keen interest in exploring QoS and constraints-aware service compositions.

2.3.2 RQ 1.2: Discussions on the techniques/Methods taken by service composition approaches

This section describes the techniques/methods or actions taken by service composition approaches introduced in the Section 2.3.1.

– **QoS-aware service composition :** In the literature, researchers addressed the QoS-aware service composition from different viewpoints, e.g., QoS correlation, service-dependent QoS, internal complimentary and optimal QoS along with the varying strategies of optimisation. Our discussion include the most influenced research works in the QoS-aware approach. Zeng et al. [2] introduced AgFlow middleware platform for QoS-aware web service composition. They presented two different strategies for QoS-aware web service selection (i) local optimisation of web service selection using Simple Additive Weighting (SAW) technique and the global planning of web service selection using Integer Programming (IP) approach. In the local optimisation approach, a Multiple Criteria Decision Making (MCDM) is applied to select a web service for each given task in the composite web service; (ii) whereas, the global optimisation approach chooses an optimal execution plan from all possible paths based on the IP. Further, AgFlow middleware pro-

vided the service quality model that helps to evaluate the overall quality of composite web service. Alrifai et al. [69] proposed a heuristic methodology that combines the global optimisation with local selection technique. They used Mixed Integer Programming (MIP) to get the best decomposition of the global QoS constraints into native constraints and then the local selection is applied to find an optimal web service that satisfies these local constraints. Further, Alrifai et al. [28] presented a skyline technique for selecting QoS-aware web services in the composition. In the skyline process, they first used MIP to prune all non-skyline service from each service classes that improve the skyline search. Further, the skyline services are selected by comparing all local skyline services in the skyline service set that excludes those services that are dominated by other services. Berbner et al. [48] presented a heuristic H1_RELAX_IP that applies a backtracking algorithm on the result obtained from the Integer Programming. H1_RELAX_IP used the Integer Programming to find the web services that satisfy the local constraints and then a backtracking algorithm is applied to the collected web services and repeatedly search the set of web services that meet the global constraints. Chen et al. [87] modelled QoS-aware service composition as a multi-objective optimisation problem and proposed E-Dominance Multi Objective Evolutionary Algorithm (EDMOEA) for optimising the service composition. The algorithm is used to find the pareto optimal services, and it also facilitates the users to select the best service with the tradeoff of QoS risk and performance. Chen et al. [38] modelled QoS-dependencies in multi-objective service composition. They adopted a pruning algorithm for eliminating infeasible candidate services from the search space. A Vector Ordinal optimisation techniques based service composition algorithm is developed for finding pareto optimal set of the candidate service composition plan. Recently, Liang et al. [32] introduced the concept of internal complimentary in the QoS-aware web service selection. Internal complimentary allows multiple candidate services selection within the same service class to form a new composite candidate service in the service composition. They first transformed the problem into a Multi-choice Multi-dimensional Knapsack Problem (MMKP) and then performed an optimisation process to find the optimal service compo-

Table 2.7: Representative examples for QoS-aware approach

Source	Method	QoS Parameter	Optimisation
Zeng et al. [2]	Simple Additive Weigh, Integer Programming	Execution Price, Reputation Execution Duration, Availability , Sucessful Execution Rate	Local and Global
Alrifai et al. [69]	Mixed Integer Programming	Reponse Time, Availability, Price, Reputation	Global
Alrifai et al. [28]	Skyline Algorithm	Response Time, Availability, Price, Reputation	Global
Berbner et al. [48]	Integer Programming	Response Time, Availability, Throughput, Scalability, Reputation	Global
Chen et al. [87]	Multi-Objective Evolutionary Algorithm	Cost, Execution Time, Latency, Reliability, Availability	Global
Chen et al. [38]	Vector Ordinal optimisation Technique	Response Time, Availability, Cost, Reliability, Throughput, Reputation	Global
Liang et al. [32]	multi-choice multi-dimensional knapsack	Response Time,Throughput , Cost, Reliability,	Global
Huang et al. [47]	multi-criteria decision making, Integer Programming	Availability, Response Time, Reliability, cost	Global
Ding et al. [85]	Genetic Algorithms	Price, Execution Time, compensating time, success possibilit	Global
Li et al. [98]	Integer Programming, Iterative Algorithm	Response Time,Throughput , Cost,	Local
Canfora et al. [15]	Genetic Algorithms	Response Time, Availability, Cost, Reliability	Global
Aschoff et al. [42]	EWMA	Response Time, Cost	None
Wu et al. [76]	Tree-Based Search	Execution Time, Reliability	None
He et al. [4]	Integer Programming, Skyline, Greedy Algo	Cost, Response Time , Availability, Throughput	Global

sition. Huang et al. [47] presented an optimal QoS based web service composition. They adopted multi-criteria decision making with the weighted sum model that enables the service consumer to assess service quality numerically. Further, they modelled the QoS-driven optimisation using Integer Programming for selecting an optimal service. Ding et al. [85] considered transactional property in the QoS-aware service selection. They argued that the service execution time might be affected by the transactional properties, and the existing method deals with either local optimal or global optimal composite service under fixed transactional workflow. To deal with these issues, they employed a genetic algorithm for optimising the service composition based on the defined transaction rules.

Li et al. [98] presented an expand region algorithm that reduces the reconfiguration

cost by identifies the limited reconfiguration region of faulty service and then the service reconfiguration algorithm replaces all subprocess services which are involved in the reconfiguration region. Canfora et al. [15] presented a framework for binding and re-binding the composite service. They proposed an algorithm that re-estimates the new QoS and triggers the service re-binding whenever any QoS deviations are detected. Further, they adopted Genetic algorithms for optimising the service composition. Aschoff et al. [42] presented a ProAdapt framework for the proactive adaptation of service composition due to changes in composite service. They used the Exponential Weighted Moving Average (EWMA) that models the response time of service operation, which would then trigger recomposition when likely degradation of response time is detected. He et al. [4] proposed MSSOtimizser (Multi-tenant SaaS Optimizer) that provides effective and efficient service selection for a multi-tenant SaaS cloud. They considered the SaaS provider's optimisation goals, e.g., least resource cost, better performance and maximised revenues etc. Further, they adopted a hybrid methodology that combines the skyline, integer programming, and greedy algorithm together to optimise an optimal service composition plans for all the users. Wu et al. [76] introduced a tree-based heuristic approach for web service composition in the SaaS cloud. They presented a tree-based algorithm that builds the tree of all possible combinations of composition solution and then performed the filtering to remove all unlikely and illegal composition paths. After that, they applied a best-first search algorithm to evaluate and rank the optimal service composition. Table 2.7 summarises the algorithms and their metrics for the QoS-aware service composition approach.

– **Constraints-aware service composition :** Web services in the real world may not be universally applicable, and there might be some process restrictions (or constraints) imposed by the service provider [35]. For example, order delivery web service may have constraints to deliver items in the specified region in the country or world. Such type of process constraints presented by Wang et al. [35] in their constraint-aware service composition approach. They adopted preprocessing methods for filtering all the web service that can do the same task in the SIDE service set and then used a graph search-based

Table 2.8: Representative examples for constraint-aware approach

Source	Methods	Constraints	Optimisation
Wang et al. [35]	Graph Search-Based Algorithm	Service uses restriction	None
Lin et al. [14]	Iterative Algorithm	End-to-End QoS	Local
Laleh et al. [30]	Graph Plan-Based Approach	External Constraints (e.g., new restictions)	None
Yu et al. [66]	MMKP, Graph-Based Search Approach	End-to-End QoS	Local and Global
Tang et al. [74]	Hybrid Genetic Algorithm	Optimal Constraints	Global
Yu et al.[64]	MMKP, Multi-Constraint Optimal Path	End-to-End QoS	Local and Global
Zeng et al.[106]	Linear Programming	Global Constraints	Global

algorithm that selects an appropriate web service from the SIDE service set and optimise the constrains-aware service composition. Lin et al. [14] described a multisteps algorithmic approach; in which an expand region algorithm is proposed that identifies the reconfiguration region of faulty services including some neighbouring services for maintaining end-to-end QoS constraints and then the service process reconfiguration algorithm replaces all subprocess services which are involved in the reconfiguration region. Laleh et al. [30] presented a constraint adaptation-aware service composition that supports runtime-adaptation of new constraints in the composite service execution. They adopted a graph-plan-based approach for optimising the constraint-aware service composition. Yu et al. [66] introduced a broker-based architecture that facilitates the service selection intended to maximise an application-specific utility function to meets the end-to-end QoS constraints. They defined the problem into two steps. Firstly, they transformed the problem into a combinatorial optimisation problem as Multi-dimension Multichoice 0-1 Knapsack Problem (MMKP). Secondly, the graph model is adopted to define the problem as a Multi Constraint Optimal Path (MCOP) problem. Based on these methods, an efficient heuristic algorithm is proposed for optimising the end-to-end constraint-aware service selection. Tang et al. [74] proposed a hybrid genetic algorithm for optimal constraints -aware service selection. In the hybrid strategy, they used a local optimiser with

an aim to improves QoS values and eliminates the constraints violation for optimising the service composition plan. However, many approaches were discussed in the literature they imposed QoS-constraints or end-to-end constraint either at the design-time service selection or run-time composite service reconfiguration [64, 106]. Table 2.8 summarises the algorithms and constraints types used in the constraints-aware approaches.

– **SLA-aware service composition :** Wu et al. [82] introduced a broker-based strategy for SLA-aware service composition. In this approach, they implemented a service broker agent that works as an intermediator between the service providers and users. The service broker agent facilitates on-demand service strategies and provides a guarantee with the reliability of QoS data which is specified in the SLA. Further, they presented two algorithms for implementing a service broker agent. Depth-First Search (DFS) algorithm traverses the entire search space and find the service instance which satisfies the QoS requirements. A greedy algorithm is proposed that combines the Genetic Algorithm for optimising the service composition. In the SaaS cloud, a composite application would inevitably operate under dynamic changes on the workload that affects the composite application performance or leads to SLA violation [6]. For handling such SLA violation proactively, Leitner et al. [70] used machine learning regression model for predicting the SLA violation in the composite service execution. Further, Leitner et al. [79] extended data-driven approach for automated prediction of SLA violation in the service composition. Also, they presented a PREvent framework that dynamically monitors and prevents the SLA violation in the composite service execution [73]. Wada et al. [67] presented an optimisation framework named E3 for solving SLA-aware cloud service composition problem. E3 - MOGA (Multi-Objective Genetic Algorithm) finds the optimal solutions which are equally distributed in a objective space and select any one of them based on the end-user SLA requirements. Moreover, E3 MOGA supports three different categories of users, namely silver, gold and platinum for generating optimal service composition plans. Xiong [33] developed a framework for SLA-aware service composition that supports SLA negotiation and QoS constraints-driven service provisioning according to SLA

Table 2.9: Representative examples for SLA-aware approach

Source	Method	SLA Parameters	Runtime-SLA Support	Optimisation
Wu et al. [82]	Greedy Algorithm, Genetic Algorithms	Response Time, Availability, Reputation, Reliability	Not Supported	Global
Leitner et al. [70]	Machine Learning Regression Technique	Response Time	Pridiction of SLA violation	None
Leitner et al. [79]	ANN-based regression , ARIMA	Response Time	Prediction of SLA violation	None
Leitner et al [73]	Regression Technique	Response Time	Monitoring, Prediction, Prevention of SLA violation	None
Wada et al. [67]	Multi-Objective Evolutionary Algorithm	Throughput, Cost, Latency	Not Supported	Global
Xiong [33]	Capacity Planning Approach	Response Time, Cost Availability,	SLA violation, SLA Negotiation	Global

requirements. Further, they described the SLA violation and penalty calculation model for the service composition and applied a capacity planning approach for optimising the service composition. Table 2.9 summarises the methods and SLA parameters used in the SLA-aware service composition.

– **Context-aware service composition :** This approach is derived from the context-awareness that could be generated by the users requirements, service providers or the service environment. Lin et al. [36] presented a service selection approach based on the context-aware factors of QoS attributes. They developed a QoS context for domain-specific QoS attributes, e.g., translation quality for machine translation service. And then, they argued that the aggregation of domain-specific QoS attributes withing other services QoS attribute could not be possible at the service composition level. They proposed a QoS prediction based algorithm for selecting the component services in the composition that meets the user's requirements. Zhang et al. [91] argued that the context could be derived from the parameter correlations that exist between the input parameter and the out parameter of the invoked web services in the service composition. The parameter correlation can be extracted from the service composition history, and based on this correlation

Table 2.10: Representative examples for context-aware approach

Source	**Method**	**Context Parameters**
Lin et al. [36]	Algorithm Based on Prediction Technique (average of past values)	Domain Specific QoS Atributes
Zhang et al. [91]	PersonalRank and SimRank Algorithms	Parameter Correlations
Keidl et al. [63]	Programmable Framework	Location Context and Consumer Context
Xu et al. [88]	Matrix Factorization	User Geographical and Service Information
Maamar et al. [65]	Software Agents	Details of Service Operating Environment
Kapitsaki[71]	Model-Driven Development	Context-Aware Development

information; they build the context network of all services. They adopted the PersonalRank and SimRank++ algorithms that construct the context network for mapping the similarity of any two web service for the service composition. Keidl et al. [63] introduced a context-aware framework for creating context-aware adaptable composite service. The proposed framework supports various contexts of service composition, e.g., location context (such as consumer's current location; address, GPS coordinates, country, timezone etc.) and consumer context (such as name and e-mail address etc.). Similarly, Xu et al. [88] presented a context-aware QoS prediction technique based on the user context information and the service context information. They adopted the Matrix Factorization (MF) method for developing two novel prediction models named User-context-aware MF model and Service-context-aware model. They studied the function that maps the relationship between the geographical distance and the similarity values and then based on the optimal mapping value it selects the best web service from the candidate web service set. Maamar et al. [65] provided an agent and context-based approach for web service composition. For simplicity, they used a software agent that works for the users and context denotes the relevant information extracted from the situation or an operating environment. Further, they presented three software agents, namely composite-service-agent, master-service-agent, and service-agent; these agents coordinate to the context

component (which has the service operating information) for selecting the web services in the composition. Table 2.10 summarises the techniques and context information used in context-aware service composition.

– **Uncertainty-aware service composition :** This approach is designed to deal with the QoS uncertainty in the service operating environment (e.g., unpredictable service failure or dynamic changing workload etc.). Mostafa et al. [13] proposed two meta-heuristic approaches for multi-objective optimisation of service composition under uncertainty where no prior QoS information is available. They adopted a reinforcement learning algorithm that deals with the uncertainty characteristics in the dynamic environment for solving multi-objective QoS problems. In the first approach, they presented single policy multiple multi-objective service compositions, and the second approach deals with the multi-policy multi-objective service composition. These approaches adopt a self-organisation mechanism that exploits the problem structure to derive the weights of different QoS objectives and finds a set of Pareto optimal solutions that satisfy the multiple QoS attributes in an uncertain environment. Niu et al. [90] adopted a multi-objective evolutionary algorithm for finding an optimal QoS of web service composition under uncertainty (UQ-WSC). They first modelled the UQ-WSC with matrix and interval number. Further, they provided a novel encoding representation of UQ-WSC using interval number based multi-objective optimisation problem (IMOP). And then, they applied MOEA/D on this encoding for optimising the optimal service composition plan. Wang et al. [86] employed a Graphplan based approach that deals with the service execution uncertainty for optimising the service composition. They explained that the uncertainty could be related to the service execution, and they presented an extended Graphplan which represents the services with uncertain effect. New rules are applied to identify the mutual exclusion, and then the Graphplan method produces the branch structure of the composite solution. Benouaret et al. [77] employed the skyline approach for selecting services under uncertain QoS environment. They modelled QoS attributes using possibility distribution and based on that they compute the dominating factor of web services over each other. Further,

Table 2.11: Representative examples for uncertain-aware approach

Source	Method	Uncertainty Parameters	Optimisation
Mostafa et al. [13]	Reinforcement Learning	Uncertain QoS	Global
Niu et al. [90]	Multi-Objective Evolutionary Algorithm	Uncertain QoS	Global
Wang et al. [86]	Graphplan	Uncertain Execution Effect	Local
Benouaret et al. [77]	Probabilistic Top-K Technique	Uncertain QoS Fluctuations	None
Wen et al. [83]	Skyline	Uncertain QoS	Global

they introduced the post-dominant skyline and nec-dominant skyline methods based on the dominance relationship. After that, they presented an efficient algorithm based on the post-dominant and nec-dominant skyline method that deals with QoS uncertainty in skyline service selection. Wen et al. [83] adopted a probabilistic technique for modelling uncertain QoS-aware service composition. They introduced an uncertain QoS model that examines the probability and dominating association between web services and a novel aR-tree data structure for storing and retrieving the data of an uncertain QoS model. After that, they applied a probabilistic top-k framework for determining the dominating ability of web services and then they employed heuristic rules for selecting the best web service for the composition. Table 2.11 summarises the techniques and uncertainty factors used in uncertainty-aware service composition.

– **Economic-Driven service composition :** This approach includes not only the economic aspects but also users and service providers requirements. Alzaghoul et al. [10] applied a real-option approach for managing technical debt in the cloud-based service selection. They identified technical debt of substitution decisions driven by the need to scale up of service capacity. Further, they employed the Binomial Real Options (BRO) approach to staging the selection decision in order to quantify the debt when the substituted web service starts to pay off and clear out the technical debt. Alzaghoul et al. [84] extended service substitution decision under uncertainty. The service substitution problem is formulated using the BRO approach. Further, they adopted a Design Structure Matrix (DSM) and time and cost-aware propagation matrix to estimate the value of the

Table 2.12: Representative examples for economic-driven approach

Source	Method	Economic Aspects	QoS Parameters
Alzaghoul and Bahsoon [10]	Binomial Option	Predicted Utility	Scalability
Alzaghoul and Bahsoon [84]	Binomial Option, Design Structure Matrix	Predicted Utility	Scalability
Ye et al. [78]	Bayesian Network	Long-Term Economic Model	Throughput, Cost, Response Time

Table 2.13: Representative examples for adaptive or reconfiguration of composition

Approaches	Parameters	Sources
QoS-Aware Approach	QoS Violation, Service Failure	[2, 98, 42, 15, 68]
Constraint-Aware Approach	End to End Constraitns Violation, External Constraints Service Failure	[14, 41, 30, 72]

postponed decisions. Ye et al. [78] employed a Discrete Bayesian Network to design an economic-driven model for the user and service provider in the cloud. Further, they applied Influence-Diagram (ID) for selecting the long-term based cloud service composition. Most of the approaches have used the design-time economic-driven web service selection for the service composition. Table 2.12 summarises the techniques and economic aspects used in economic-driven service composition.

From the above discussion, we summarise that the QoS-aware approach and constraint-aware approach had gained a lot of attention in the SOC research community other than existing approaches. Table 2.13 provides an overview of both approaches that further applied for dynamic adaptation of composition or service reconfiguration under the failure-prone environment or constraints violations. However, QoS-aware approach is the only approach that was employed in the multi-tenant service composition in the SaaS cloud [4, 11, 76]. From Table 2.12, we observed that the research community has a lack of interest in an economic-driven service composition. The proliferation of an Internet-based modern software delivery systems (e.g., SaaS cloud) in the cloud market provides huge market opportunities and service options to service providers and service consumers. It is obvious that the SaaS provider wants to generate more revenue (or maximise SaaS optimisation goals) from the offered web services in the Cloud market. In this sense, there

is a need to develop some economic-driven service composition techniques in the dynamic SaaS cloud environment. We will discuss more on economic-driven aspects in RQ 1.3.

Answering RQ 1.2: We provided extensive discussions on the service composition techniques/methods and classified the service quality factors involved in the service composition. Further, based on the results discussions, we observed that an economic-driven approach has significantly fewer studies and needs to investigate at large scale. We also summarised the current attention of these approaches in the SOC community.

2.3.3 RQ 1.3: Discussion and Future Outlook for Research

The results obtained from the SLR are shown that the field of service composition has gained a lot of attention in the past decades. The results provided some key observations that could help to guide future research. Notably, this SLR identified many research gaps in the field of service composition that are potentially related to SaaS-based service composition and the run-time service reconfiguration decisions. In this context, we aim to address the third sub-research question. **RQ 1.3 :** *What are the future directions in an economic-driven service composition research, in particular SaaS cloud?* This question provides the state-of-the-art discussions and gives useful insight into how we can benefit from the existing service composition approaches to draw the motivation on the key requirements and discussing the pitfalls when applying these service composition approaches in the SaaS cloud.

2.3.3.1 Benefiting existing approaches to develop an economic-drive service composition framework in the SaaS Cloud

From the SLR results, we observed that service composition approaches had been studied based on the service quality factors needed by either service consumer or service providers. Further, most of the approaches are designed to optimise the service composition plan

for the single end-user system [4]. However, these types of approaches have the limitations and would not be efficient for emerging computing paradigms such as SaaS cloud. Further, instead of serving a single user, the SaaS cloud is capable of processing diverse[3] requirements submitted by multiple end-users. But, the majority of approaches have the limitations to support such diverse requirements, and could not fit for utilizing the full capacity of the SaaS characteristics. However, a little work has done [4] that focused on the multi-tenant service selection with similar functional needs. In this context, a new multi-tenant service composition approach that can deal with such a complex requirements in the SaaS cloud is necessary.

Further, based on the discussion in the Section 2.3.2., we observed that the majority of approaches [30, 69, 82, 87] are intended to satisfy the end-users requirements without accounting economic aspects (e.g., service utilization that increase service revenue) in their service selection process. Only few research works [10, 78] have considered economic aspects but, at design time service selection. In particular, these approaches have not used the economic aspects (e.g., cost-benefits trade-off) in the dynamic SaaS cloud environment. Also, these approaches have not explored the predictive techniques (e.g., forecasting) perspective for the long-term based service selection decision that determines whether the service selection decision reduces the service debt and satisfies the QoS constraints in a dynamic environment or not.

Tha adaptive composition and reconfiguration of services have been studied under the failure-prone environment (e.g., service failure), changing response time and constraints violation [14, 42]. They only consider to replace the faulty component services [14] (including neighbour services in the faulty service set) by selecting high capacity component services that satisfy the QoS constraints at the moment, but without considering the service utilisation that could be a prominent source of incurring the service debt under changing workload in the dynamic environment, e.g., SaaS cloud. These component services selection decisions may be the potential source of accumulating the service debt

[3]Diverse requirements refer to different function and QoS requirements.

under changing workload, which is very undesirable in the SaaS cloud. We will discuss these issues in the next section.

2.3.3.2 Finding essential ingredients for developing an economic-driven service composition approach in the SaaS cloud using Technical Debt

The modern software delivery system, such as SaaS cloud is a challenge for existing service composition approaches. In particular, the dynamic nature of SaaS cloud tends to lead the uncertainty in an operating environment, and the composite application running under such operating environment always has some risks such as performance, scalability, delayed latency and SLA violations etc. On the other side, the SaaS provider has its own optimisation goals [4](e.g., minimum operating cost and maximum service revenue, better application performance, and minimum SLA violations etc.) for delivering web services in the SaaS cloud market. However, SaaS cloud provides economic-driven solutions to end-users by achieving true multi-tenancy at the composite application level, where each component service has the ability to serve several users simultaneously [4]. In this regard, we aim to identify the required ingredients for developing an economic-driven service composition framework in the SaaS cloud.

Based on the results of this SLR (Tables 2.7 - 2.11), most of these approaches are designed to build service composition that satisfies an end-user requirements. Moreover, there is a lack of adoption of these approaches in the SaaS cloud [4, 11, 76]. However, in the context of SaaS cloud, these approaches are not efficient due to optimising QoS for a single-user system [4]. Suppose, we apply these approaches in the SaaS cloud, where multiple users have different types of SLA (e.g., silver, gold, or platinum) [5, 67] that exhibits different functional and QoS requirements for the same service, e.g., CRM service. These approaches optimise service composition plan for each user one by one. As a result, they take more time to process all users requests in the SaaS cloud, and they reduce the SaaS efficiency (e.g., productivity) as well as service reputation. Within this context, we propose a novel evolutionary algorithmic approach that supports multi-tenant

service composition. (This limitation is addressed in Chapter 3).

Further, the existing methods (from Table 2.13) triggered the adaptive composition or service reconfiguration based on the predicted SLA violation, QoS constraints violation and the component service failure etc. They do not consider an economic-driven perspective in the process of service reconfiguration in a dynamic environment. In particular, composite application in the SaaS cloud would inevitably operate under unpredictable and dynamic changes of requests workload generated by the users and its consequences could be the degradation of composite application performance, reducing composite service revenue and the frequent violations of SLA and QoS constraints. All these issues bring a challenging task: when to (re)compose the component services such that the service revenue over time is maximized? It is obvious that there is an engineering cost of reconfiguring the composite service in the SaaS cloud and such frequent adaptation would not be desirable due to incurring an extra cost for the service provider. In this scenario, there is a need to reduce unnecessary adaptations by making an economic driven decision on whether to recompose the service or not. Moreover, in this direction, the existing methods [14, 15, 30, 41, 42, 98] ignored the fact that service over-utilisation[4] may not necessarily be a bad result. A penalty could be paid against the SLA violation if it can be the source that stimulates largely increased utility in the long term and also avoid the unnecessary service reconfiguration (and save the extra adatpatation cost).

However, we argue that the technical debt could be associated with an inappropriate engineering decision or poorly justified runtime decision for recomposing the composite service that carries short-term benefits (e.g., satisfying SLA requirements) but not geared for long-term benefits or future value creation in the composition. For example, the utilisation of component services participating in the composite service execution may be sub-optimal due to significant up and drop in the requests workload generated by the users in the SaaS cloud environment. The sub-optimal service composition leads the debt in a way to provides higher capacity service than the service demand by the

[4]over-utilization may be acceptable in short time, as long as the workload is only a 'spike' and the loss can be paid off by long term benefits.

users. Consequently, the operating cost may outweigh the service revenue. On the other hand, over-utilisation of service, capacity leads the SLA violations and the penalty cost against the response time violation of the service request can be count as interest over the technical debt. For addressing these runtime decision issues, we leverage the technical debt metaphor that supports an economic-driven decision for service recomposition in the SaaS cloud is presented in Chapter 4.

From an economic-driven perspective (Table 2.12), it is clear that most of the methods (Tables 2.7 - 2.11) have not used an economic driven based long-term service selection for meeting constraints requirements in a dynamic environment. Furthermore, existing service composition approaches are often rely on over-optimistic assumptions, such that both (or one) local and global constraints are hard [14, 64, 66, 107] and can always be satisfied. For example, SaaS providers imposed some hard constraints for the web service delivery in the cloud market, and they would be interested in getting at least 85% utilisation (e.g., local constraint) from each component web service and the entire composition should be utilised at least 90%. But these constraints can always not be possible to satisfy in dynamic changing workload generated by users in SaaS. Moreover, there may be no such component service that meets the constraints and consequently leading the hardness in composing the service plan. In this context, we advocate the development of a technical debt-aware two-level constraints reasoning framework for the long-term based economic-driven service selection. This issue is addressed in Chapter 5.

Answering RQ 1.3: We provided discussions and recommendations of the future research, specifically for an economic-driven service composition in the SaaS Cloud. In particular, we leveraged the existing service composition methods for identifying the necessary ingredients towards developing technical debt-aware economic-driven framework for service composition in the SaaS cloud.

2.4 Related Reviews

This section provides the discussions on the related surveys and SLR studies in the field of web service composition.

Regarding the concept of service composition, Dustdar et al. [94] discussed the need for service composition and presented an overview of the existing service composition approaches such as static and dynamic service composition. Similarly, Moghaddam et al. [108] presented a comparative review study on the web service composition and discussed different service selection and composition approaches such as optimisation based approach (e.g., local and global optimisation), Negotiation-based approach (e.g., SLA negotiation) and hybrid approach. Some studies are conducted based on the specific service quality factors and techniques (e.g., computational intelligence and AI planning). Jatoth et al. [92] conducted a systematic literature review with the focus on computational intelligence methods for QoS-aware web service composition. They presented a classical taxonomy of computational intelligence methods, such as heuristic and metaheuristics methods as well as non-heuristic techniques and then leads the discussions on each technique extensively for the perspective of QoS-aware service composition. Other QoS-aware surveys provided the general discussions on the techniques and service selection strategies [93, 95]. Truong et al. [97] presented a study on the context-aware web service composition system. They provided an extensive discussions on the context-aware information representation and the techniques for web service composition such as context reasoning techniques, security and privacy techniques, and context-adaptation techniques etc. Among them, fewer studies discussed the automated service composition approaches based on AI planning techniques (e.g., PDDL) [109, 110]. Wang et al. [111] surveyed service composition, specifically on the bio-inspired algorithms such as evolutionary algorithms, particle swarm optimisation, and ant colony algorithm etc. Jula et al. [96] performed the SLR on the cloud computing service composition with a particular focus on the adoption of service composition in the cloud computing environment.

They provided discussions on the categorisation of service composition techniques such as combinatorial algorithms, structure, classic and graph-based algorithms, and frameworks. These studies have provided extensive discussions on the concept of service composition, methods or techniques for implementing service composition but have not categorised the service composition approaches based on the service quality factors and not included economic-driven perspective discussions.

To the best of our knowledge, this systematic literature review bridge this gap by providing a classification framework of existing service composition approaches based on the service quality factors including an implementation technique and the discussions on an economic-driven perspective for web service composition in a dynamic environment, e.g., SaaS cloud.

2.5 Review Threats

This systemic literature review methodology left behind some restrictions that were needed to be explained further. Apparently, it was well developed and followed the guideline provided by [58, 59, 60, 62].

- For SLR validity, we viewed two main threats (i) selection bias and (ii) data extraction. The bias selection might appear in articles selection process. A research protocol was designed and performed to resolve the selection bias data with reference to the significant studies (Section 2.2.1). We used the protocol to mitigate these threats by conducting an array of activities: preliminary background, research questions, search process, selection criteria, search execution, quality assessment and data extraction. We did our best efforts to develop this protocol, but we acknowledge that selection criteria (inclusion and exclusion) may be missing some significant research contributions that could help us for guiding the future research.

- This protocol is reviewed and revised independently by other members in the research group. In particular, they examined that the designed search query can cover

the research questions and the objective of this SLR. They provided constructive feedback for mitigating the bias formulation in the protocol, particularly choosing the search keywords. However, web service is the common term over Internet-based computing that might be lead a risk of bias selection or missing the relevant research articles in the field of study. To mitigate these issues, we decided to add more specific data in the search keywords. Although, we did our best to include all relevant terms, still, we can't confirm the completeness in terms of retrieving all relevant data for the research questions.

- We applied the data extraction procedure defined in Section 2.2.5 that helps to enhance the consistency of data extraction. Moreover, the quality assessment criteria is used to ensure the quality of findings.

- We conduced the manual and automated search on the meta-data, which includes the keywords, title and paper abstract. However, we acknowledge that research articles might be used the service composition implicitly (e.g., service-based system) without mentioning it into the title or the keywords. Moreover, there may be a risk in the automated search that could not retrieve the relevant research articles from the digital bibliographic library due to search engine quality. However, we selected the most significant bibliographical sources for performing this SLR. To mitigate the above problems, we conducted the manual and automated search on the well-known venues (e.g., conferences and journals) recognised by the research community in the field of study.

2.6 Summary

In this chapter, we conducted a systematic literature review on service composition. In particular, we described the state-of-the-art service composition under the umbrella of well-defined taxonomy that systematically categorised the current service composition approaches based on their service quality factors. Further, we provided extensive dis-

cussions on the underlying techniques and algorithms taken by these approaches. The results obtained from this SLR shown that the existing service composition approaches have less adoption of economic-driven perspectives and its application in the SaaS cloud. We discussed the need for economic-driven aspects when composing the application in the SaaS cloud. Further, based on the SLR findings, we discussed the current research issues and identifying the research gap that guides the future research in an economic-driven service composition in the SaaS cloud.

CHAPTER 3

MULTI-TENANT SERVICE COMPOSITION IN SAAS CLOUD USING EVOLUTIONARY OPTIMISATION

In Chapter 2, we presented a systematic literature review on service composition that boosted our understanding of the field. Further, it allows us to identify the research gap (problem), particularly, the limited support of economic-driven perspective in the existing service composition approaches in the SaaS cloud. In this chapter, we present an evolutionary optimisation based approach to model multi-tenants service composition problem, thereby, addressing thesis Research Question 2. We modelled this problem as a multi-objective optimisation problem and employed one of the most popular Multi-Objective Evolutionary Algorithm (MOEA) named `MOEA/D-STM` *(Stable Matching-based Selection in Multi-Objective Evolutionary Algorithm based on Decomposition) [26]. In particular, we present new encoding representation and fitness function that model the service selection and composition as an evolutionary search. We incorporate our approach in* `MOEA/D-STM` *and develop an evolutionary optimisation based service composition engine. Further,* `MOEA/D-STM` *based service composition engine supports different types of users requests and optimise the service composition plan for each category of users in a multi-tenant SaaS cloud. We have also incorporated this encoding representation in* `NSGA-II` *[27] for performing the comparative study. The experiment results show that the* `MOEA/D-STM` *outperforms* `NSGA-II` *in terms of quality of solutions and computation time.*

3.1 Introduction

In the SaaS cloud environment, tenants may have a varied dimension of QoS and functional requirements for similar services (e.g., Sale CRM) [5]. In order to satisfy the tenants' SLA requirements, application engineer chooses the suitable web services from the service pool in the SaaS cloud and then optimises the service composition plans for each category of the tenant. However, in the SaaS cloud environment, several functionally equivalent services are available with different QoS values. The selection of candidate services from the SaaS cloud that satisfy the QoS constraints (e.g., throughput, response time, and availability etc.) in the composition is viewed as an NP-hard multi-objective optimisation problem [112, 113] which takes a significant amount of time and cost to find the optimal service composition plans from the huge search space. This can be particularly challenging in the real-time deployment scenarios, that characterised by scale, the large number of multi-tenants, functionalities and varying QoS.

Furthermore, with an increasing interest of Service-Oriented Computing (SOC) community in the non-functional requirements-driven service composition [114, 115, 116], many research studies were published on the QoS, SLA or constraints-aware web service composition problems for a single-user system [3, 4, 117, 118, 119, 120]. However, these approaches have the limitations and would not be efficient for the emerging computing paradigms such as SaaS cloud. Further, instead of serving a single user, the SaaS cloud is capable of processing diverse requirements submitted by multiple tenants. But, the majority of these approaches have the limitations to support such diverse requirements. For example, these services compositions approaches tend to support the execution plans that search for service provisions of equivalent functionalities but with varying QoS and cost constraints to meet the tenants' QoS requirements or respond to QoS changes dynamically. However, these approaches tend to ignore the fact that the multi-tenant service composition needs to provide variant service execution plans, each offering a customized plan for a given tenant with its functionalities, QoS and cost constraints.

To address the mentioned limitations, we propose a multi-tenant service composition approach using evolutionary optimisation. In a nutshell, the key contributions of this chapter are summarised as follows:

- We contribute a novel diverse requirements-driven multi-tenant service composition approach in the SaaS cloud using evolutionary optimisation.
- We present a novel encoding representation and fitness evaluation strategy that explicitly considers the multi-tenant and QoS requirements in the SaaS cloud.
- We propose a service composition engine accelerated by an evolutionary algorithm named `MOEA/D-STM` [26]. The service composition engine is capable of handling diverse requirements of multiple tenants and optimise the service composition plan for each tenant in the SaaS cloud.
- We evaluate the service composition engine, specifically `MOEA/D-STM` performance on a real-world web service QoS dataset named WSDream [19, 38]. The results show that, in contrast to `NSGA-II` [27], the `MOEA/D-STM` achieves better performance and the solution quality for optimising service composition plans.

3.2 Motivating Scenario

We take Sales CRM (Customer Relationship Management) service as our motivating scenario that illustrates the challenges of multi-objective optimisation of multi-tenant service composition in the SaaS cloud [5]. Let us consider different types of tenants[1] requests for the Sales CRM service, which is available in different service packages. For example, Salesforce [5] provides Sales CRM service in different packages namely professional, enterprise, and unlimited. These service packages are differentiated based on the number of functionalities in the service. In multi-tenant SaaS cloud, tenants can request for a different sales CRM service package based on their SLA requirements. The SaaS cloud facilitates

[1]Tenants refer the end-users in the SaaS cloud

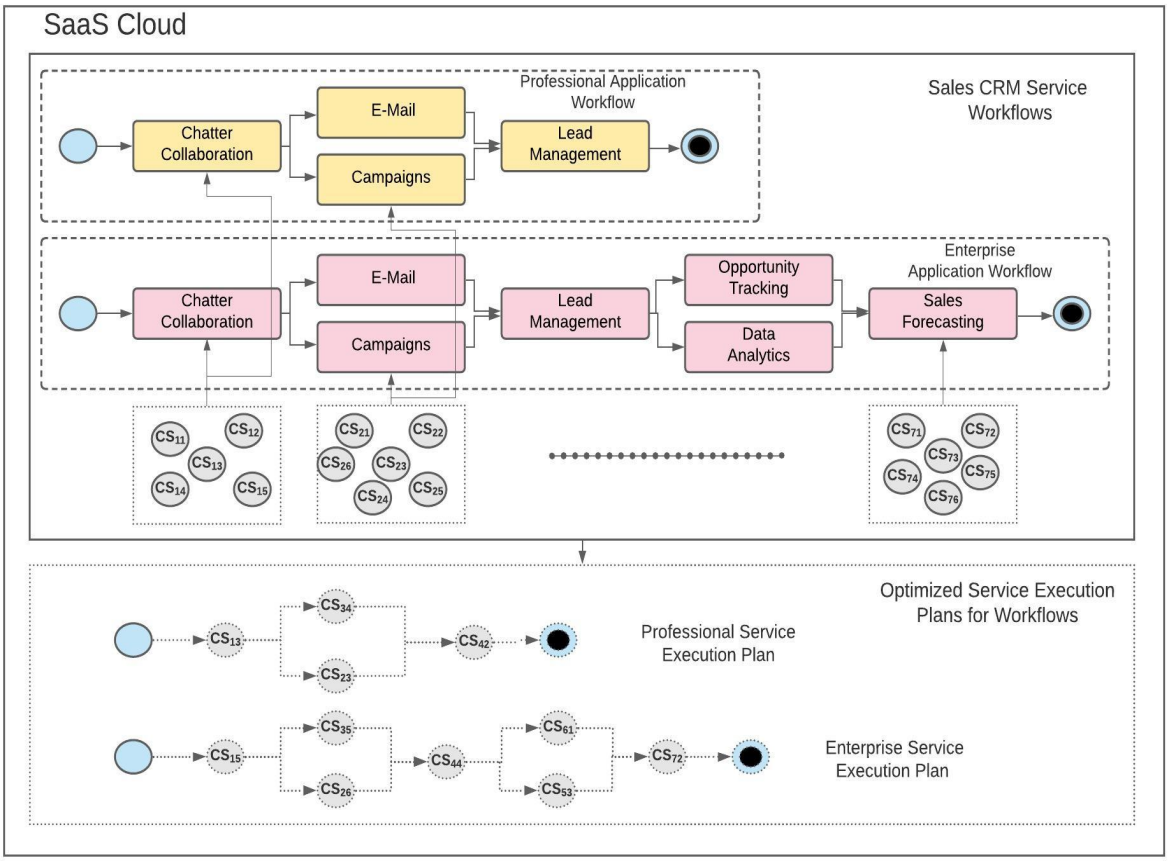


Figure 3.1: Motivation Example

the Sales CRM service to multiple tenants according to their SLA requirements. Suppose, several tenants submit their requests to the SaaS cloud; in response to the requests, SaaS cloud returns the Sales CRM service package as per tenant's specific SLA requirements. However, tenants may have the multi-dimensional QoS and functional requirements, one tenant may request for the high throughput despite the cost of professional service while another tenant is interested in getting enterprise service with lower response time and cost. In these scenarios, each tenant has diverse requirements, a service execution plan needs to be created for the tenant of each category by selecting suitable web service from the service pool.

Figure 3.1 depicts two different application instances (application workflow) of Sales CRM service namely professional and enterprise. These application workflows contain a different number of tasks (abstract services). In the SaaS cloud environment, several

functionally equivalent services are available with the different QoS (such as throughput and response time) values for attaining each task in the application workflow. Therefore, the selection of suitable candidate services from the SaaS cloud is a combinatorial optimisation problem [2] and become highly challenging when application engineer needs the dynamic optimisation of composing/re-composing the service execution plan for each category of user in the SaaS cloud.

3.3 Problem Formulation

According to the requirements discussed in Section 3.2, in this section, we formulate service composition problem and gives basic definitions of service composition in the multi-tenant SaaS cloud environment.

- **Definition 1 :** *Application Workflow* indicates the set of abstract services $A = \{a_1, a_2, a_3, ...a_n\}$ which are usually connected using sequential or parallel connectors. In such workflow, n indicates the total number of abstract services.

- **Definition 2 :** A set of *candidate concrete services* for each of abstract service in the workflow $CS = \{(cs_{11}, cs_{12}, ...cs_{1m}), (cs_{21}, cs_{22}, ...cs_{2m}), ...(cs_{n1}, cs_{n2}, ...cs_{nm})\}$; where m is the total number of candidate concrete services for a_n abstract service.

- **Definition 3 :** A *Service composition* is represented by an application workflow $A = \{a_1, a_2, a_3, ...a_n\}$. One concrete service from the set of candidate concrete service (CS) is selected to finish a_n abstract service in the application workflow $A = \{cs_{1m}, cs_{2m}, ...cs_{nm}\}$.

- **Definition 4 :** *In multi-tenant service composition*, suppose there are p application workflows A_p that consist n abstract services represented by $A_{pn} = \{A_{1n}, A_{2n}, ...A_{pn}\}$.

- **Definition 5 :** (QoS attributes) Suppose there are l QoS attributes in a service composition, Q_r, $r = (Q_1, Q_2, ..Q_l)$ and Q_r attribute indicates the r^{th} non-functional property of the composite service.

3.4 QoS Computing Model for Service Composition

QoS attributes are the non-functional properties of a web service and these QoS need to consider for differentiating the service composition plan during the service selection process. Usually, multiple QoS attributes are considered in the service composition. In particular, we consider three QoS attributes for the service composition namely throughput, response time, and cost.

- **Throughput :** Throughput of a SaaS application is defined as the number of requests the application is able to process per second.
- **Response Time :** Response time of a SaaS application is defined as the time required to send a request and receive the response from the server.
- **Cost :** The execution cost of a SaaS application is the fee that a tenant need to pay for invoking operations.

The aggregate value of each QoS attribute involved in the process of service composition is computed based on the QoS aggregation functions in Table 3.1 [87, 121, 122].

Table 3.1: QoS aggregation functions for sequence and parallel patterns

QoS Attributes	*Sequence*	*Parallel*
Cost (C)	$\Sigma_{i=1}^{t} C(s_i)$	$\Sigma_{i=1}^{t} C(s_i)$
Throughput (T)	$min_{i=1}^{t} T(s_i)$	$\min_{i=1}^{t} T(s_i)$
Response Time (RT)	$\Sigma_{i=1}^{t} RT(s_i)$	$\max_{i=1}^{t} RT(s_i)$

where cost function ($\Sigma_{i=1}^{t} C(s_i)$) calculates the total cost of invoked web services (s_i) in the composition. Throughput function ($min_{i=1}^{t} T(s_i)$) calculates the overall throughput of the composite application by choosing the web service which has minimum throughput among the invoked web services (s_i) in the composition. Both functions do not consider the connectors (sequence and parallel) impacts in the computation, specifically, how the web services are connected in the composition. Whereas, Response Time function ($RT(s_i)$) treats both connectors differently (i) the function ($\Sigma_{i=1}^{t} RT(s_i)$) combines the

response time of all web services (s_i) connected using a sequential connector (ii) the function ($\max_{i=1}^{t} RT(s_i)$) choose the web service which has maximum response time among the invoked web services (s_i) in the parallel connector.

Further, these QoS attributes can be exhibited in positive and negative criteria [2]. Service composition process should optimise the higher value for the positive QoS attribute (e.g., throughput, availability and reliability) and lower the values for the negative QoS criteria (response time, latency and cost etc.) [123]. Moreover, these QoS attributes have numerical values at the different scale of units. For example, response time is expressed in the milliseconds while reliability is expressed in the percentage. The opposite direction and the different scale units create the inconsistency for estimating the QoS of a composite service. To give the equal preference of all QoS attributes for computing the utility of a composite service, we calculate the normalized value of each QoS attributes in the range of (0,1). Equation 3.1 is used to normalize the negative QoS attributes and positive QoS attributes are normalized using Equation 3.2 [2, 123].

$$N(Q^{-}) = \begin{cases} \frac{Q_r^{max} - P}{Q_r^{max} - Q_r^{min}} & if Q_r^{max} \neq Q_r^{min} \\ 1 & if Q_r^{max} = Q_r^{min} \end{cases} \tag{3.1}$$

$$N(Q^{+}) = \begin{cases} \frac{P - Q_r^{max}}{Q_r^{max} - Q_r^{min}} & if Q_r^{max} \neq Q_r^{min} \\ 1 & if Q_r^{max} = Q_r^{min} \end{cases} \tag{3.2}$$

Where Q_r^{max} and Q_r^{min} indicate the maximum and minimum values of the r^{th} QoS attribute of all candidate services involved in service composition and P is the current attribute value of a candidate service.

3.5 Modelling of Service Composition using Evolutionary Optimisation

In Sections 3.1 and 3.2, we discussed the multi-tenant service composition problem in the SaaS cloud. And then, the problem is formulated in Section 3.3 as an optimisation problem. This problem brings two challenging tasks (i) How to optimise different service composition plans for the multiple tenants (ii) How to achieve the QoS of service composition plans that satisfy each tenants' requirements in SaaS cloud. Multi-Objective Evolutionary Algorithms (MOEA) are capable of dealing with these optimisation problems. Also, MOEA has been widely used by Service-Oriented Computing (SOC) community [112]. In this research, we adopt `MOEA/D-STM` (Stable Matching-based Selection in Multi-Objective Evolutionary Algorithm based on Decomposition) [26]. We provide a novel encoding representation which is discussed in the subsequent section.

3.5.1 Encoding representation

The important aspects of an evolutionary algorithms are its chromosomes and their representation because a chromosome capture all the relevant information required for a solution to the problem being considered [124]. A chromosome is represented by a vector of a_n genes, where n is the total number of abstract service in the service composition (application workflow). The formulation of a chromosome represents the composite service solution; in which a gene encodes the concrete service for each abstract service in the service composition that could be a possible candidate solution, as shown in Figure 3.2. Moreover, Figure 3.3 shows the solution representation encoded by the chromosome. The value of each gene represents which concrete service (its index value) has been selected for the corresponding abstract service such as abstract service a_1 selects the concrete service CS_{1_16}, abstract service a_2 selects the concrete service CS_{2_12} and abstract service a_3 selects the concrete service CS_{3_21} and so on.

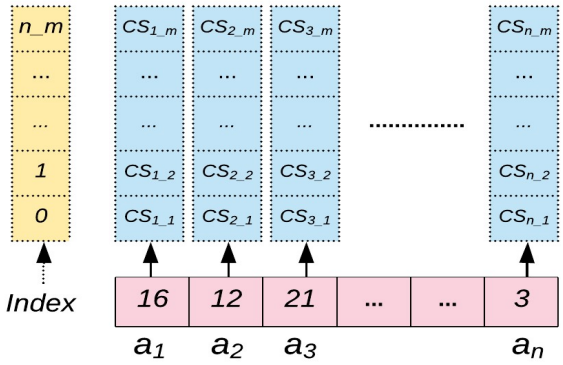


Figure 3.2: Chromosome encoding

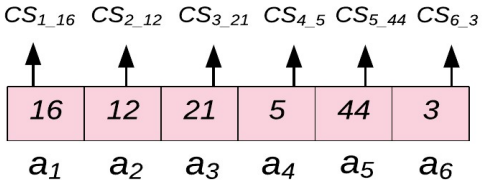


Figure 3.3: Solution representation in chromosome encoding

Further, we use two types of application workflows (e.g., professional and enterprise) in our multi-tenant service composition model that are represented by a chromosome, as shown in Figure 3.4. In particular, we design the chromosome in such a way that it dynamically splits itself into two parts (e.g., sub-chromosome) during the optimisation process. The first part processes the four genes (e.g., a_1 to a_4) of the chromosome for optimising professional service composition plan and the second part processes all genes of the chromosome for optimising an enterprise service composition plan. We incorporated this novel chromosome encoding into `MOEA/D-STM` and the current status of `MOEA/D-STM` is to support an independent execution of two application workflows and optimise them in each generation independently by applying genetic or problem-specific operators such as selection, crossover, mutation, and reproduction.

3.5.2 Optimisation process in `MOEA/D-STM`

The `MOEA/D-STM` is one of the most popular Evolutionary Algorithm (EA) for solving optimisation problems based on the principle of decomposing a Multi-Objective Optimisation Problem (MOP) into a set of scalar optimisation subproblems [26, 125]. `MOEA/D-STM` has

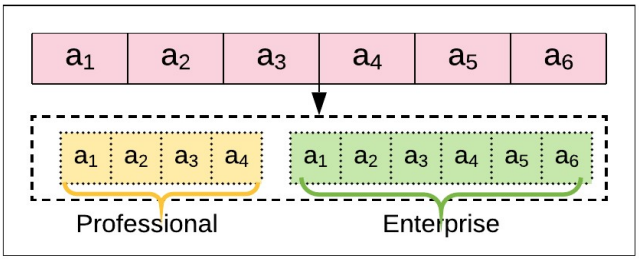


Figure 3.4: Chromosome encoding for the professional and enterprise application workflow

several advantages over other EA in terms of objective scalability, computational efficiency and better performance on combinatorial optimisation problems [26, 126].

In optimisation process, the service composition problem is model as a minimisation problem. In which, all QoS objectives values should be minimised during the evolutionary optimisation process. In particular, as part of an optimisation process, `MOEA/D-STM` maintains the population of the individuals that represent a candidate Service Composition (SC) solution. It uses Tchebycheff approach for decomposing MOP into N subproblems and each subproblem has one solution in the current population [26]. Each subproblem is characterized by a uniform spread N weight vectors (λ). The Tchebycheff approach finds the closest feasible solution SC to an ideal point (commonly known as reference point) by measuring the distance between feasible solution SC and ideal point as defined follows

$$D_\lambda(cs, q) = \min_{1 \leq i \leq m} \{\lambda_i | q_i(cs) - q_i^*\} \tag{3.3}$$

Where q_i and λ_i are the QoS value and weight of the i^{th} dimension of n^{th} feasible solution SC (sub-problem) respectively. For the i^{th} dimension, q_i indicates the good solution value from the set of all neighbour possible solutions of the n^{th} subproblem.

Algorithm 1 describes the process of optimising N subproblems and corresponding individuals in the set of population. We compute the Euclidean Distance between two weight vectors (sub-problems) and then form a group $B(i)$ of T closest weight vectors. At each generation, randomly select two solutions (parent) from the $B(i)$ or the current

Algorithm 1: Optimisation Process in `MOEA/D-STM`

Initialize the population $\mu \leftarrow \{x^1, x^2, .., x^N\}$, a set of weight vectors $\lambda \leftarrow \{\lambda^1, \lambda^2, .., \lambda^N\}$, the ideal and nadir objective vectors z^*, z^{nad}.
`/* Compute neighbour group B of T closest weight vector */`
for $i \leftarrow 1 \quad to \quad N$ **do**
  $B(i) \leftarrow \{i_1, i_2, .., i_T\}$, where$\lambda_{i1}, \lambda_{i2}, .., \lambda_{iT}$ are the T closest weight vector to λ_i
end
while *Stopping criterion is not satisfied* **do**
  $p \leftarrow \emptyset$
  for $i \leftarrow 1 \quad to \quad N$ **do**
    `/* Random selection of solution set between B(i) and` μ `*/`
    if *(rnd <* $neighbourSelectionProbability$*)* **then**
      $S \leftarrow B(i)$
    else
      $S \leftarrow \mu$
    end
    `/* Randomly select two solution from S */`
    $x^a, x^b \leftarrow parentSelection(S)$
    `/* Reproduction operations */`
    $y \leftarrow crossoverOperation(x^a, x^b)$
    $y' \leftarrow mutationOperation(y)$
    evaluate the fitness function value of y'
    $p \leftarrow y'$
    update the current z^* and z^{nad} objective vectors
  end
  $e \leftarrow \mu \cup p$
  $\mu \leftarrow STM(e, \lambda, z^*, z^{nad})$
end
return μ

population. The crossover operation is applied on the selected parents with the crossover probability of 0.9 for producing a new offspring, the new offspring genes are mutated with the probability of mutation rate $1/n$ using polynomial mutation operator, where n indicates the number of genes in the individuals. Using Tchebycheff approach, we evaluate the fitness of new offspring against an individual in $B(i)$ or the current population. If a new offspring indicates an improved solution quality then replace it with the unfeasible solution in the current population. After generating the new offspring population, STM maintains the diversity in search space by allocating most preferable parent solution to each subproblem in the current population. This reproduction process is repeated with

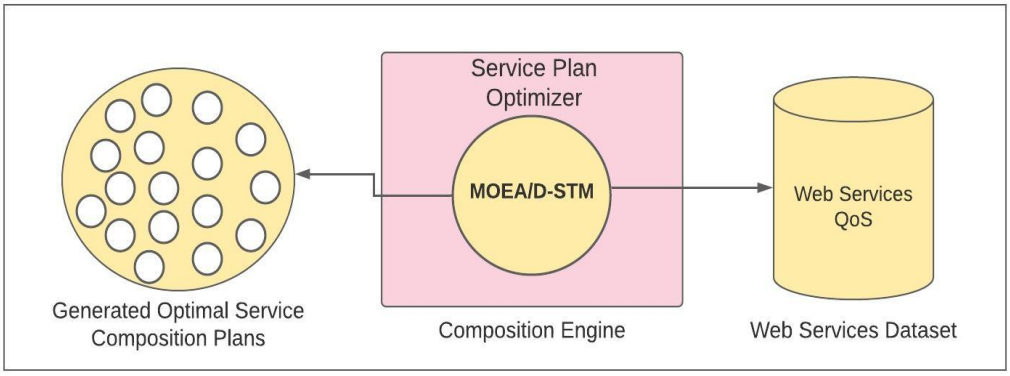


Figure 3.5: `MOEA/D-STM` based service composition engine

all genetic and STM operations until reaching the number of generations defined in the algorithm.

3.6 `MOEA/D-STM` Based Service Composition Engine

This section presents the *Service Composition Engine (SCE)* accelerated by an evolutionary algorithm named `MOEA/D-STM` . Figure 3.5 illustrates the overview of *SCE*. As can be seen, there are two levels of components, namely *QoS dataset* and *service plan optimizer*. The *QoS dataset* component stores the real-world web service QoS data (WSDream dataset [19, 38]) and classify the web service QoS values according to their functional properties. The *service plan optimizer* component is implemented using `MOEA/D-STM` and then a novel chromosome encoding presented in Section 3.5.1 is incorporated into `MOEA/D-STM`. Based on the feeding chromosome information, `MOEA/D-STM` interacts with the *QoS dataset* component and retrieves the relevant web services QoS values for performing optimisation process discussed in Section 3.5.2. Further, the *service plan optimizer* produced the set of optimal service composition plans, as shown in Figure 3.5. Currently, *SCE* supports to optimize the two different service composition plans for the application workflows named professional and enterprise. However, *SCE* is flexible in adding a varied number of services in the optimisation process to generate scalable service composition plans for satisfying the requirements of a new application workflow.

3.7 Evaluation

To evaluate the service composition engine, specifically, `MOEA/D-STM` algorithm performance, we design experiments to assess the performance (e.g., computational time and solution quality) of `MOEA/D-STM` by mean of comparing it with one of the most popular evolutionary algorithm (e.g., `NSGA-II` [27]). Specifically, our experiments aim to answer the following research questions.

- **RQ 3.1 :** What is the computational time of the `MOEA/D-STM` comparing with the state-of-art `NSGA-II`?
- **RQ 3.2 :** Whether `MOEA/D-STM` can achieve better solution quality than the state-of-art `NSGA-II`?

3.7.1 Experimental Setup and Results

For experimental purpose, we have used a Sales CRM system as our testing environment, which is formed as a service composition and also helps setting up the experimental parameters. Suppose, given Sales CRM service has two different application workflows which are connected using sequential and parallel connectors. Table 3.2 shows that the total number of abstract services are 4 in a professional workflow with one parallel connector, whereas an enterprise workflow contains 7 abstract services with two parallel connectors. We have used a diverse range of candidate concrete services (20 to 50) for attaining each abstract service in the workflow. Moreover, we used WSDream dataset, which records the QoS values of response time and throughput of 5825 real-world web services collected from 73 different countries [19, 38]. We picked 350 web services with valid response time and throughput values, which are randomly associated with the candidate concrete services. Further, we randomly partitioned these web services into 7 service categories corresponding to 7 different abstract services in the application workflows. We randomly generate the cost as an additional attribute added with each concrete candidate service. We conduct a

Table 3.2: Workflow Configuration (Note: AS-Abstract Service, CS-Concrete Service)

Composition Workflow	Number of AS	Max. Number of CS per AS	Sequential Connector	Parallel Connectors	AS per Parallel Group
Professional	4	20	2	1	2
Enterprise	7	50	4	2	4

Table 3.3: Parameters for the algorithms

Parameter Name	Values and Operators
Population Size	100
Crossover Operator	SBXCrossover with crossover probability 0.9
Mutation Operator	Polynomial mutation with mutation probability 1/n
Parent Selection	NSGA-II: Binary Tournament Selection MOEA/D-STM: Random Selection
Max. Generation	1000 - Professional and 3000 - Enterprise
Neighborhood Size	20
Neighborhood Selection	0.9 Probability
Max. replaced solutions	2
Function Type	Tchebycheff (TCH)

comparative study on the `MOEA/D-STM` to `NSGA-II` based on the computational time and solution quality estimated by quality indicators. We have used the JMetal framework for implementing our approach in MOEA [127]. Moreover, JMetal framework provides the support of automatic calculation of quality indicators (e.g., HV, GD, Spread, etc.), thanks to JMetal community [128]. We used the same parameter setting for both algorithms, as shown in Table 3.3.

All experiments are conducted on the same machine with Intel Core i7 2.60 Ghz. Processor, 8GB RAM and Windows 10 x64.

3.7.2 Comparative Approach

To answer all the research questions, we examine the performance of `MOEA/D-STM` against the following evolutionary algorithm

- `NSGA-II` : We have used `NSGA-II` [27] as the underlying evolutionary algorithm for our comparative study because it is one of the most popular algorithm in the search-based software engineering community [129]. Further, we modelled the ser-

vice composition approach in NSGA-II with the similar workflow setting used in the MOEA/D-STM.

3.7.2.1 Metrics

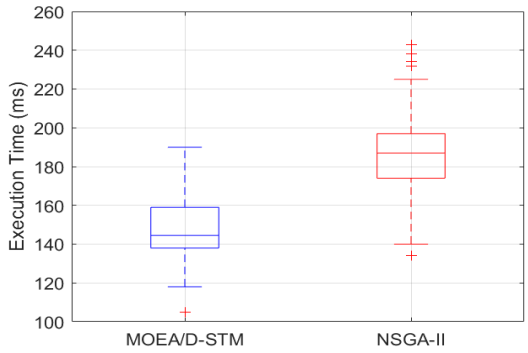
We leverage the following metrics to assess the results:

- **Computational Time :** For the fair assessment of our evaluation study, all experiments were executed 30 times independently and the average measure is used for the evaluation study.
- **Quality Indicators :** Quality Indicator is used to assess the quality of a set of Pareto optimal solutions [130]. Particularly, it is an effective method that maps the Pareto solution set to a real number that indicates many aspects of the solution quality [131, 132]. In this study, we adopted two most widely used quality indicators namely Hypervolume (HV) and Generational Distance (GD).
- **QoS Attribute Values :** We selected the best value of throughput, response time and cost objectives from the solution set for 30 independent runs and average measure are used for the evaluation study.

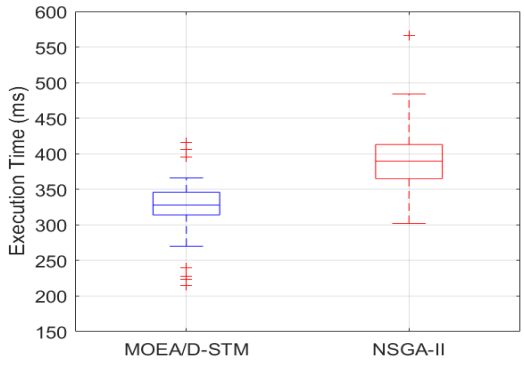
3.7.3 RQ 3.1: Computational time-based comparisons

To answer the research question (RQ 3.1), we conducted two sets of experiments on THE different workflows, namely professional and enterprise. Both experiments show the variation of execution time under the diverse range of candidate concrete services in the search space and the varied number of abstract services in the workflow.

Execution Time vs. Number of Concrete Services − We analyse the variations in execution time based on the number of candidate concrete services per abstract service in each composition workflow. We deployed 20, and 50 candidate concrete service for each abstract service in the professional and enterprise workflows respectively. In particular, Figure 3.6: (a) and (b) show that the execution time is increased over the number



(a) `Professional` - Running time on both algorithms (Comparisons between MOEA/D-STM and NSGA-II statistically significant ($p < .05$))



(b) `Enterprise` - Running time on both algorithms (Comparisons between MOEA/D-STM and NSGA-II statistically significant ($p < .05$))

Figure 3.6: Execution time yields `MOEA/D-STM` and `NSGA-II`

of candidate concrete service grows in the search space. However, both algorithms execution times are increased with respect to concrete service increment in the search space. Overall, `MOEA/D-STM` takes lower execution time than `NSGA-II` for both `professional` and `enterprise` workflows. We have a statistically sound conclusion, from the Kruskal Wallis test at the significant level of 5%, the P-value is less than 0.05 which strongly suggests that there is a significant difference in the mean of running time for both algorithms.

Answering RQ1: the running overhead imposed by both algorithms indicate that the `MOEA/D-STM` outperforms `NSGA-II` .

3.7.4 RQ 3.2: Assessment of Solutions Quality

To asses the solution quality obtained from 30 independent runs of `MOEA/D-STM` and `NSGA-II`, we choose two widely used performance indicators, namely `Hypervolume (HV)` and `GD` [133, 134]. These indicators are used to evaluate the convergence and diversity of Pareto solution set [132].

- **Quality Indicators :** HV calculates the volume of the dominated portion of the objective space [134]. GD determines the convergence by computing the average

Table 3.4: Mean value of HV and GD for `professional` and `enterprise` workflow

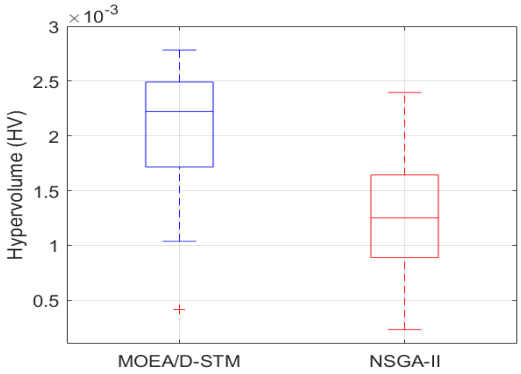
Composition Workflow	**MOEA/D-STM**		**NSGA-II**	
	HV	GD	HV	GD
Professional	2.07 E-03	2.88 E-01	1.26 E-03	3.27 E-01
Enterprise	1.06 E-04	2.69 E-01	3.21 E-05	2.60 E-01

Table 3.5: QoS achieved by algorithms on `professional` and `enterprise` workflow

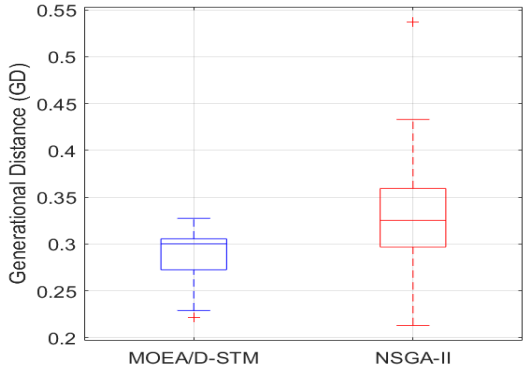
Evolutionary Algorithm	***Professional***			***Enterprise***		
	TH	RT	C	TH	RT	C
MOEA/D-STM	41.32	0.12	9.44	76.82	0.100	18.42
NSGA-II	37.98	0.28	13.22	67.36	0.103	19.75

distance of the obtained solutions points from the Pareto Front (PF) [134, 135]. Figure 3.7 and Figure 3.8 show the standard statistic boxplot about the HV and GD values obtained from the 30 independent runs of `MOEA/D-STM` and `NSGA-II` for the composition workflow `professional` and `enterprise`, respectively. Further, Table 3.4 shows the mean of HV and GD, which are obtained from the 30 repeated runs. The higher mean value of HV and the lower mean value of GD are desirable, and they also show that the algorithm finds a good approximation to the PF [131, 132]. The results of `professional` in Figure 3.7 and Table1 3.4 show that the `MOEA/D-STM` achieved better values of HV and GD than `NSGA-II`. Moreover, `enterprise` workflow results in Figure 3.8 shows that `MOEA/D-STM` yielded a better value of HV than `NSGA-II` but slightly degrade the value of GD. Overall, `MOEA/D-STM` achieved better results over `NSGA-II`. We have a statistically sound conclusion, from the Kruskal Wallis test at the significant level of 5%, the P-value is less than 0.05 which strongly suggests that there is the difference in mean of HV and GD; between the `MOEA/D-STM` and `NSGA-II`.

- **QoS Attributes Values :** We obtained the best value of throughput, response time and cost objectives from the Pareto optimal solution set of candidate service composition for the 30 independent repeated runs and mean value of each objective is used in the evaluation study. Table 3.5 shows that the `MOEA/D-STM` retrieved

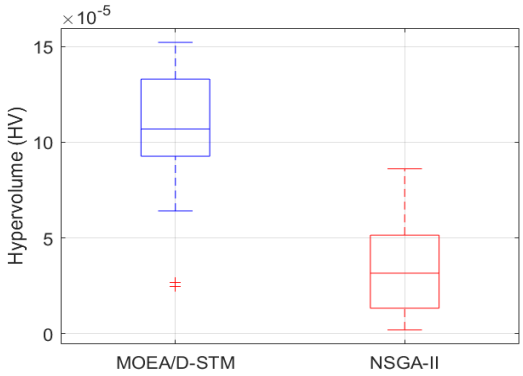


(a) `Professional` - HV on both algorithms (Comparisons between MOEA/D-STM and NSGA-II statistically significant ($p < .05$))

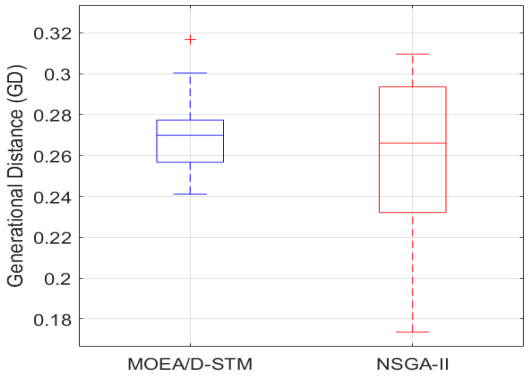


(b) `Professional` - GD on both algorithms (Comparisons between MOEA/D-STM and NSGA-II statistically significant ($p < .05$))

Figure 3.7: HV and GD yields `MOEA/D-STM` and `NSGA-II` on the `application` workflow



(a) `Enterprise` - HV on both algorithms (Comparisons between MOEA/D-STM and NSGA-II statistically significant ($p < .05$))



(b) `Enterprise` - GD on both algorithms (Comparisons between MOEA/D-STM and NSGA-II statistically significant ($p < .05$))

Figure 3.8: HV and GD yields `MOEA/D-STM` and `NSGA-II` on the `enterprise` workflow

better throughput and response time on the lower cost than `NSGA-II` for both the workflow: `professional` and `enterprise`.

Answering RQ 3.2: Both the quality indicators and objective values of QOS attributes indicate that the set of Pareto optimal solution obtained from the `MOEA/D-STM` has better solution quality than the `NSGA-II` .

3.8 Summary

Existing approaches for services composition tend to be limited when addressing multi-tenant service composition in the SaaS cloud. This is due to the fact that they are not fundamentally designed for the dynamic provision of variant execution plans, each offering a customized plan for a given tenant with its functionality, QoS and cost requirements. Additionally, these approaches can fail to scale with the number of tenants, their varying functionalities, QoS and cost, rendering them unfit for real-time dynamic composition and recomposition scenarios. To address this problem, we proposed multi-tenant service composition engine goes beyond the state-of-art to envision scenarios, where variants of functionalities can be supported. We model service composition as an evolutionary optimisation problem with a novel encoding representation. For evaluating the effectiveness of our approaches, we conducted several experiments, and our results shown that the `MOEA/D-STM` outperforms `NSGA-II` in terms of performance and quality of solutions.

CHAPTER 4

TECHNICAL DEBT-AWARE ADAPTIVE DECISIONS FOR SERVICE RECOMPOSITION IN SAAS CLOUD

Chapter 3 produced the service composition engine accelerated by an evolutionary algorithm. In this chapter, we take advantage of the service composition engine through integrating into our framework. In particular, We target the problem of service recomposition in the SaaS dynamics – specifically in the given changing workload from the tenants; thus, it is not uncommon for a service composition running in the multi-tenant SaaS cloud to encounter the problem of under-utilization and over-utilization on the component services. Both cases are undesirable and it is therefore nature to mitigate them by recomposing the services to a newly optimized composition plan once they have been detected. However, this ignores the fact that under-/over-utilization can be merely caused by temporary effects, and thus the advantages may be short-term, which hinders the long-term benefits that could have been created by the original composition plan, while generating unnecessary overhead and disturbance via recomposition. In this chapter, we propose `DebtCom`*, a framework that determines whether to trigger recomposition based on the technical debt metaphor and time-series prediction of workload. In particular, we propose a service debt model, which has been explicitly designed for the context of service composition, to quantify the debt. Our core idea is that the recomposition can be unnecessary if the under-/over-utilization only cause temporarily negative effects, and the current composition plan, although carries*

debt, can generate greater benefit in the long-term. We evaluate `DebtCom` *on a large scale service system with up to 10 abstract services, each of which has 100 component services, under real-world dataset and workload traces.*

4.1 Introduction

A composite service in the multi-tenant SaaS cloud would inevitably operate under dynamic changes on the workload from the tenants, and thus it is not uncommon for the composition to encounter the problem of under-utilization and over-utilization on the component services [6]. However, the workload of composite service can be changed rapidly during execution, causing dynamic behaviour of the composite service. On the one hand, an increasing workload can rise over-utilization for the component services within a composite service, which in turns, would negatively affect the Quality of Service (QoS) and violate the Service Level Agreement (SLA) [136]. On the other hand, the decreasing workload may lead to under-utilization of the capacity of component services, reducing the revenue that has been achieved as the infrastructural resources also impose a monetary cost. All those bring a challenging task: when to (re)compose the component services such that the service utility over time is maximized?

While the problem of service recomposition (or reconfiguration) has been widely studied [14, 16, 104, 137, 138, 139], existing research studies have ignored a perhaps obvious, but complicated fact: a short-term degradation of the utility may not necessarily be a bad results; in fact, it can be the source that stimulates largely increased utility in the long term. For example, under-utilization could be desirable temporarily in order to prepared for a largely increased workload for the long-term. Similarly, over-utilization may be acceptable in short time, as long as the workload is only a 'spike' and the loss can be paid off by the long term benefits. Simply ignoring such fact is non-trivial, because despite triggering recomposition immediately upon over-/under-utilization my have short-term advantages, it can easily create instability and hinder the possibility of achieving higher

benefits for the composite services in the long-term.

To address the mentioned challenges and limitations, we contribution to an economic-driven approach, namely `DebtCom`, for triggering dynamic service recomposition leveraging the principle of technical debt [140]. In this chapter, we propose:

- We introduce technical debt as a novel metric for dynamic service recomposition. Further, we identify some critical situations or the sources in the service operating environment that significantly contributes the technical debt.
- We tailor a time-series forecasting method, namely ARFIMA model, into the debt model for predictably learning future debt.
- We propose a service debt model that explicitly maps the concept of technical debt in the context of service composition. Such a model is capable of quantifying both good and bad debt.
- The proposed service debt model, enhanced by the time-series prediction, allows us to build a utility model based on which an algorithm is proposed to equip `DebtCom` with the ability to decide whether to trigger recompoistion, considering long-term benefits. In particular, the trade-off between short-term advantages and long-term benefits can be controlled by a single value k.

4.2 Preliminaries

4.2.1 Technical Debt

Technical debt is a widely recognized metaphor in software development [8, 9, 54, 141]. Its core idea is to describe the extra cost incurred by actions that compromises long-term benefits of the developed software, e.g., maintainability, in order to gain short-term advantages (e.g. timely software release).

The technical debt metaphor was initially introduced by Cunningham [7] in the context of agile software development, where the definition is described as:

"Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. The danger occurs when the debt is not repaid. Every minute spent on not quite right code counts as interest on that debt."

In this regards, technical debt is often used as an economic-driven decision approach for communicating technical trade-off between short-term advantages and long-term benefits in the software projects [140, 142, 143].

Intuitively, technical debt makes an analogy with financial debt as described in economics [144, 145]. Often, financial debt is employed to refer to the initial loan and the interest that accumulated over time. In this regards, technical debt leverages the similar concept of principal and interest; for example, the situation in which development team decides to take shortcuts (e.g., by skipping some technical tasks in software development) for getting benefits in terms of releasing timely software product [146]. In this case, technical debt denotes the cost of the fact that some tasks are skipped and interest that may incur due to the extra cost of maintaining the software. Despite the similarities on the concepts, technical debt metaphor is not treated in the same way as the financial debt, because the interest associated with technical debt may or may not be paid off [52, 144]. However, the intuitive nature of technical debt allows the software engineers to reason about the trade-off between the related short-term advantages and long-tern benefits, aiming to make informed decisions based on when (or whether) the technical debt can be paid off [147, 148].

4.2.2 Motivating Scenario

As shown in Figure 4.1 the SaaS provider leases the IaaS providers infrastructure and deploy several functionally equivalent web services into different computing capacities(e.g., container) at IaaS platform. According to an overall workflow of abstract services, the component services are selected and composed together, each of which matches the func-

tional requirement of an abstract service, to form a service composition. The component services for an abstract service implement similar functionalities in the SaaS cloud but offer diverse levels of Quality of Services (QoS). The goal is to improve the QoS of the composite service, since there is a SLA that specific penalty for any violation. At the same time, low operation cost of the service composition is also desirable.

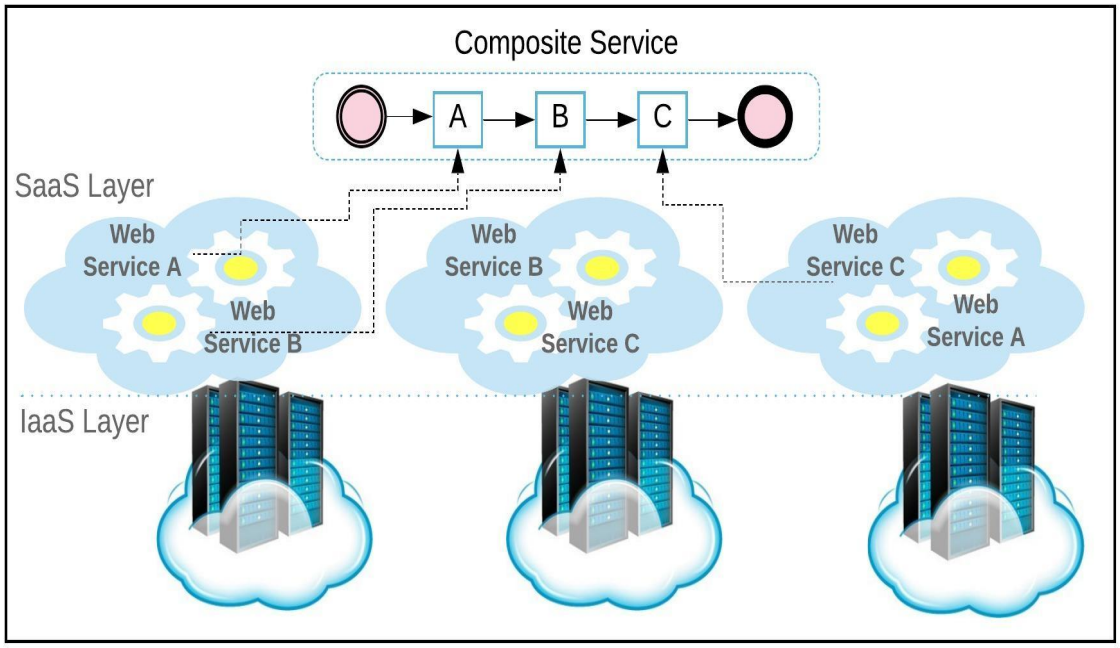


Figure 4.1: An example scenario

Given the changing workloads from tenants in this context, under-/over-utilization on each of the component services are likely to occur. In particular, under-utilization would spend less cost on the component services but could negatively affect the overall service revenue. In contrast, over-utilization could be good sometime for complying SLA, but it would incur unnecessary cost. Simply trigger recomposition as soon as over-/under-utilization is detected that may provide short-term advantages to resolve the situation to some extends, but it could also hinder the long-term benefits that could have been created by the original composition plan, creating extra operation cost, and more importantly, generating instability. In contrast, doing nothing may suffer the risk that the situation

would not change at all. The essential point is that, regardless to the negative effects and accumulated costs, both cases could be accepted as long as the cost can be paid off by benefit in the long-term. However, the fundamental difficulty is how to quantify such cost and benefit, especially taking into account the trade-off between short-/long-term effects.

In this regards, the technical debt metaphor naturally supports intuitive understanding and quantification on the trade-off between short-term advantages and long-term benefits for the service recomposition. In particular, the over-/under-utilization caused by the current composition plan can be viewed as debt, which may be temporarily and intentionally accepted as long as they can be cleared and start to create added values by a reasonable point in the long-term. However, the fundamental challenges are to identify what type of technical debt the service composition has (e.g. good or bad)?; how much debt has been incurred? and when it will be paid off for improving overall utility?; and finally to answer the question of when to trigger recomposition? These questions motivate the need of `DebtCom`, a technical debt-aware framework for recomposing services, which we propose in this chapter.

4.3 Technical Debt at Service Composition Level

In service composition, there are many situations when a composite service requires recomposition due to SLA violations, service failure, insufficient service revenue than operating cost (business objective), or QoS fluctuations, etc. [14, 16]. In this regard, we argue that the technical debt could be associated with an inappropriate engineering decision or poorly justified run-time decision of service recomposition that carries short-term advantages in terms of improving instant service utility but not geared for long-term benefits or future value creation in the composition. A little debt is not always bad if it can help the developers to speed the development process [56]. We look this argument as a valid point in service recomposition for creating values and avoiding unnecessary recomposition. This is of high significance as recomposition comes with an operation cost, especially in the

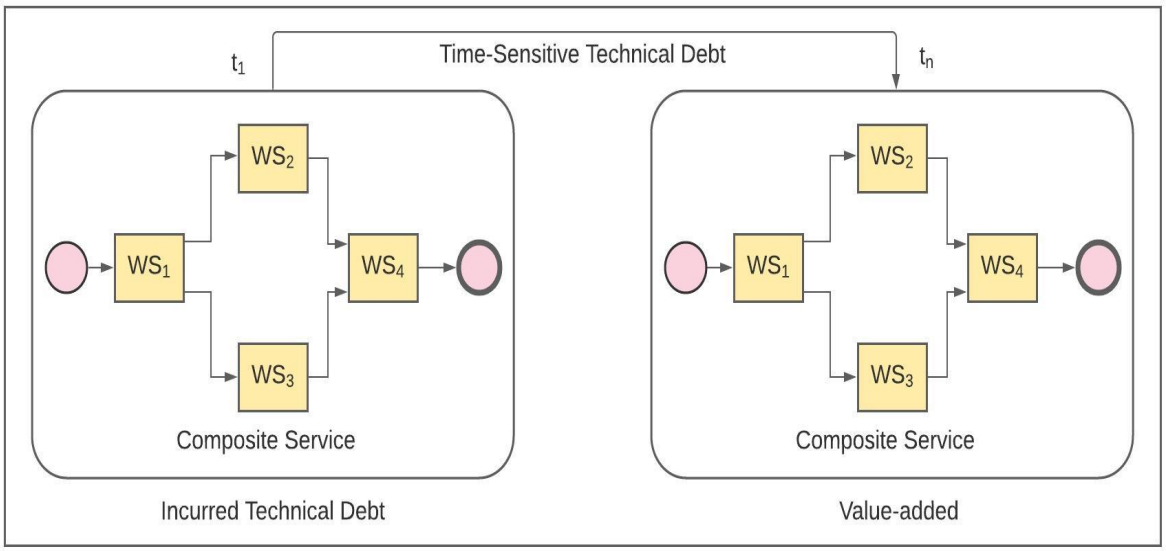


Figure 4.2: Intentional debt for exploring future values in the composition

SaaS cloud where the underlying resources are leased. Notably, technical debt could be incurred intentionally in service composition when we decide to defer the recomposition decision at time t_1 and take into account the possibility of generating future values in the current service composition plan, as shown in Figure 4.2. Moreover, it may be possible to intentionally incurred the technical debt, which is accumulated over time, for a period $(e.g., t_1, ..., t_n)$, because at each point in time $(e.g., t_1, t_2, t_3, ..., t_n)$, the debt may be increased or decreased due to dynamic changes in the requests workload generated by users in the SaaS cloud. We may accept such time-varying debt in a way to consider the future demand for scaling-up the service capacity that transforms the accumulated debt into future value. As a result, technical debt-aware decision saves unnecessary service recomposition cost and improve the composite service utility in the SaaS execution environment.

On the other hand, unintentional technical debt may be the consequences of the inappropriate or poorly justified runtime service recomposition decisions that produce weak composite service; which fails to process an incoming requests workload in a dynamic environment. Consequently, weak composite service violates the end-user SLA and the cost of penalty against each request violation could be count as interest over the incurred

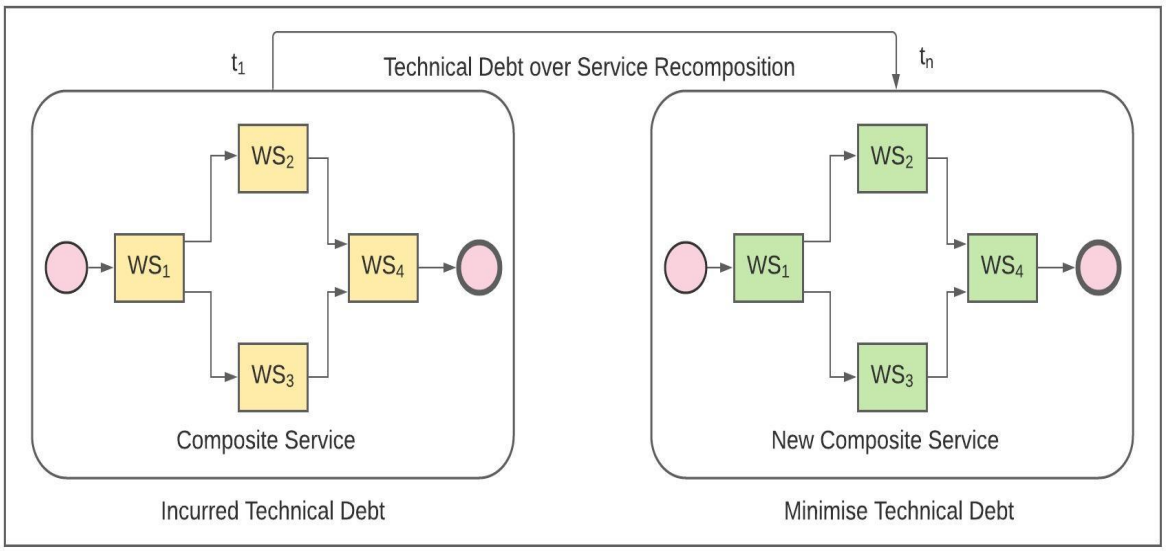


Figure 4.3: Technical Debt over Service Recomposition

technical debt. In this case, unintentional technical debt indicates the cost of efforts required to maintain end-user SLA by recomposing a new service composition plan to get better service utility value in changing request workload or to reduce the debt in the current service composition as shown in Figure 4.3. However, it is a rear condition when all participating component services (web services) in the service composition are meeting the full utilization of their capacity. Therefore, technical debt always exists during service composition. Our objectives are to reduce the technical debt and avoid unnecessary recomposition towards improving composite service utility.

4.3.1 Technical Debt Indicators

Technical Debt Indicators (TDI) consist information about what type of technical debt (good or bad debts) is, why and when was incurred, how much debt was estimated, when it will be pay off in the future [147]. We identified following key TD indicators in service composition [6].

1. **SLA Violation :** SLA violation constitutes the unintentional technical debt in service composition. When a composite service does not satisfy the predefined

response time mentioned in end-users SLA, then a penalty cost against each request violation would be counted as interest over the incurred technical debt.

2. **Runtime decisions :** An inappropriate or poorly justified runtime decisions for service recomposition may lead the technical debt in a way to select unsuitable component services for composing a new composite service which can not support the scalability requirements in changing requests workload.

3. **Service utility :** Service utility constitutes the technical debt when a composite service is sub-optimal from the utility point of view. For example, a sub-optimal composite service can incur an intentional debt by getting service scalability benefits in the future

4.3.2 Technical Debt Classification

A decision making needs to know the nature of accumulated debt in terms of good or bad debts [140][6]. We describe the good and bad debts from the composite service perspective and identifying their consequences.

1. **Good Debt:** A good technical debt in service composition is viewed as time-sensitive moving target that needs to monitor for transforming the accumulated debt into future value creation [6]. For example, Figure 4.4 shows that a composite service is underutilized in a way to deliver more than the required demand of the users at time t_1 and intentionally accumulates the debt for a time period (e.g., t_1 to t_n). We may accept such debt in a way to consider the future demand for scaling-up the service capacity that transforms the accumulated debt into future values.

2. **Bad Debt:** A bad debt in service composition may lead the situation of continuous under-utilization of composite service and will not be able to pay off the accumulated debt in the future as shown in Figure 4.5 [6]. As consequences, such accumulated

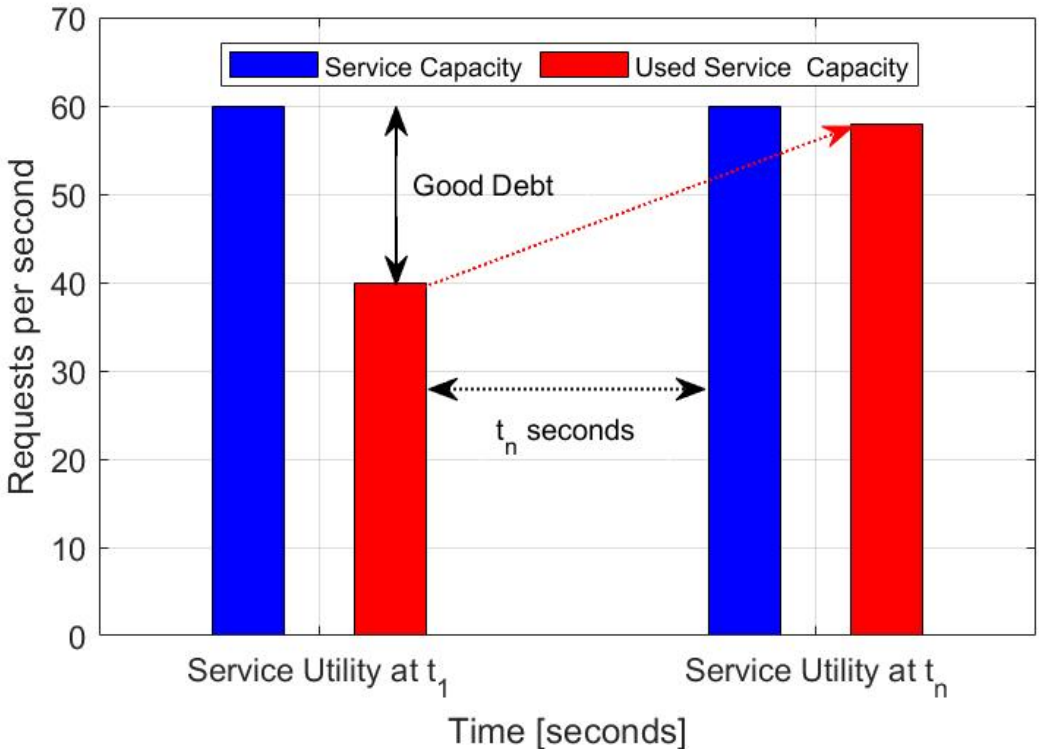


Figure 4.4: Example of Good Debt

debt negatively impacts the service utility that needs to manage by taking proactive decisions.

4.4 Time-Series Prediction of Service Workload

Proactive decision making is not uncommon, especially in the software development context where the concept of technical debt was originally created [7]. Often, the fact of whether a debt can be paid off depends on the present and future cost of the debt [147]. This is also an equivalent and important concept in our research, and therefore we seek to predict the future workload of the component services, which in turn, enabling proactive decision making for debt-aware recomposition.

In `DebtCom`, we use Autoregressive Fractionally Integrated Moving Average model (ARFIMA) [21], a widely used time-series model that guarantee the prediction accuracy when a time-series contains long memory pattern, to predict the workload of each component service.

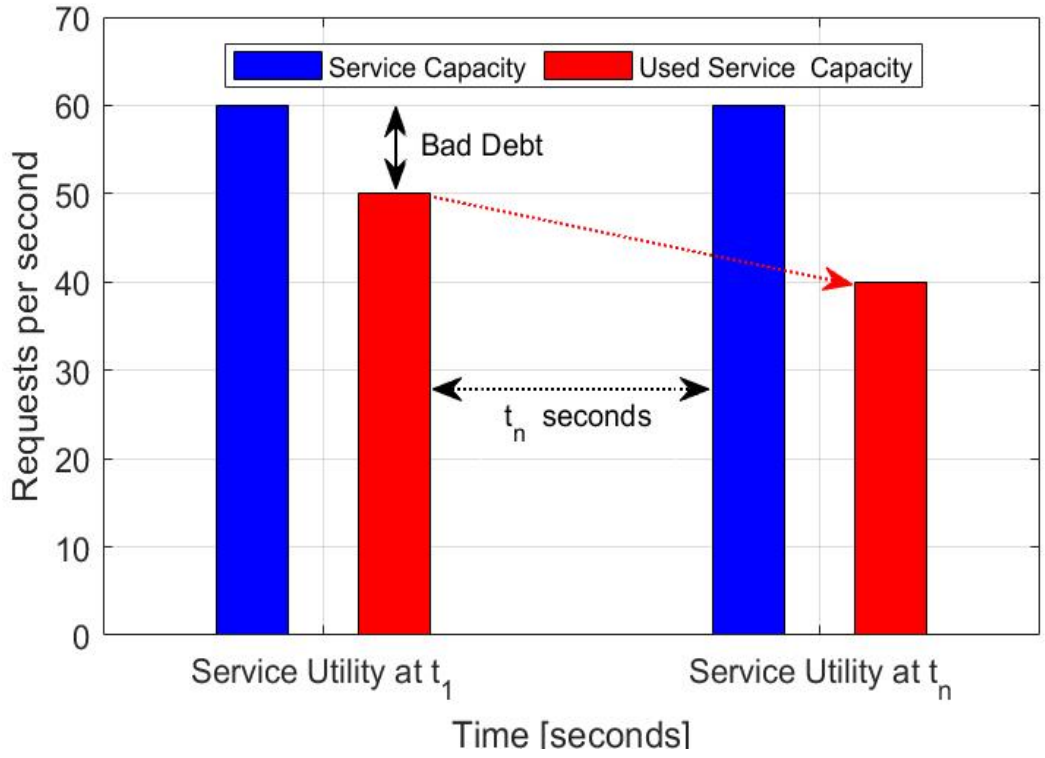


Figure 4.5: Example of Bad Debt

The workload prediction is required to capture an instant fraction of time (e.g., seconds), in which a request is either processed successfully or failed (SLA violation). Therefore, a time-series data (e.g., requests workload data) should handle such time patterns and the length of time-series interval can be adjustable to better fit the estimation. Accordingly, we prepared the data at each time point to contain a number of observed requests at each time interval (e.g., second) and feed this time-series data as an input to the ARFIMA for predicting the future requests workload at every timestep. The general expression of ARFIMA (p, d, q) for the process X_t is written as:

$$\Phi(B)(1-B)^d X_t = \Theta(B)\varepsilon_t \tag{4.1}$$

where $(1-B)^d$ is the fractional differencing operator and the fractional number d is the memory parameter, such that $d \in (-0.5, 0.5)$. $\Phi(B) = 1 - \phi_1 B - \phi_2 B^2 - ... - \phi_p B^p$ is the autoregressive polynomial of order p and $\Phi(B) = 1 + \theta_1 B + \theta_2 B^2 + ... + \theta_q B^q$ is the moving

average polynomial of order q in the lag operator B. The operator B is the backward shift operator; $BX_t = X_{t-1}$ and ε_t represent the white noise process.

The prediction process in `DebtCom` is written ARFIMA (p, d, q) process, in which we estimate the value of memory parameter d using `fdGPH()` function from the R `fracdiff` package [149]. Specifically, the value of memory parameter d must be between -0.5 and 0.5 that confirms the long memory patterns in time-series [150, 151]. The value of p is the autoregressive order that indicates the number of differenced lags appearing in the forecasting equation, and q is the moving average order that shows the number of lagged forecast error in the prediction equation. The values of p and q are identified based on the autocorrelation function and partial autocorrelation function, respectively, as supported by the `fdGPH()` function. After parameter estimation, we fit the ARFIMA model and evaluate the prediction accuracy using Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) [152].

4.5 Service Debt Model

To quantify the debt at the service composition level in the SaaS cloud, we adopt the notions of principal and interest [144, 147] from technical debt metaphor into a contextualized model for the analysis. In Section 4.3, we extensively discuss the technical debt in the context of service composition. Now we are in the position to present a formal model namely `service debt` which connects these notions such that they are made readily available to our problem.

Service debt: The service debt is a transformation of the technical debt concept, particularly for the context of service composition in the SaaS cloud. Similar to the technical debt, it quantifies the debt incurred for a certain period of time. In particular, it has two major components:

- **Recomposition principal:** This is the one-off cost of the processes that related to recompose a new set of component services.

- **Accumulated interest:** This is the cost of over-/under-utilization caused by workload changes, QoS fluctuation and inappropriate composition plan. The actual cost can be related to the penalty of SLA violation or the rented resources have not been fully utilized.

4.5.1 Recomposition Principal

In the context of service composition, we use principal to denote the invested cost of recomposing the entire composite service for improving service utility. The principal can be derived from the resources usages, such as the CPU time or the efforts spent by software engineer for the decision making of the service composition. Specifically, we compute the principal for recomposing a service using Equation 4.2:

$$Principal = E \times C_{cpu} \tag{4.2}$$

Suppose that the recomposition process requires 2 seconds (denoted as E) and the execution cost of CPU is \$ 0.0025 per second (denoted as C_{cpu}), then it takes a principal as $2 \times 0.0025 =$ \$ 0.005. The time for recomposing the services can be easily known by averaging the time for previous rounds of recomposition.

4.5.2 Accumulated Interest

An interest can be accumulated over time on the component service which may be under-utilized or over-utilized. In such context, the interests may be accumulated over time on the yth component service for the xth abstract service (denoted as CS_{xy}). For such a component service, the interests accumulated from the last recomposition time m to n can be derived from the actual service capacity (i.e., service throughput denoted as T) and the workload at time t (i.e., W_t), which may be the actual workload or predicted one

from Equation 4.1, as shown below:

$$Int(CS_{xy})_{m,n} = \begin{cases} \sum_{t=m}^{n}((T - W_t) \times C) & \text{if } W_t \leq T \\ \sum_{t=m}^{n}((\frac{W_t}{T} - R_{SLA}) \times P) & \text{otherwise} \end{cases} \tag{4.3}$$

Clearly, the interests are different depending on two different scenarios of utilizing the capacity of a component service:

- *(a) Service under-utilization:* When the component service is under-utilized, i.e., the workload is smaller than or equals to the capacity of component service ($W_t \leq T$), interest can be calculated as the accumulated cost of unused service capacity. For example, on a component service, suppose that the execution cost of processing each request is \$0.00015 (denoted as C), and a component service has the capacity to process 55 requests per second while the workload on this component service is 48 requests per second. Assuming that the accumulated interests till now is \$1.02, then this component service will carry the interest as \$1.02 + (55-48) × 0.0015= \$1.0305.

- *(b) Service over-utilization:* When the component service is over-utilized, i.e., the workload is greater than the capacity of component service ($W_t > T$), the SLA requirement on the response time (denoted as R_{SLA}) would often be violated [153], and thus a penalty rate (denoted as P) would be used to compute the extra cost to be paid. Suppose again, for a component service, that the accumulated interests till now is \$1.02, and that a given SLA contains the requirement of 2 seconds response time and the penalty rate of response time violation is \$ 0.0025 per second. Now, assuming that the average service response time, derived from the workload and its capacity, is 3.5 seconds, then the interest would be \$1.02 + (3.5-2.0) × 0.0025 = \$1.0237.

Finally, from the last recomposition time m to time n, the accumulated service debt (denoted as $D_{m,n}$) of a decision of recomposing the services can be identified and estimated

according to the principal and accumulated interests, as shown in Equation 4.4:

$$D_{m,n} = \text{Principal} + \sum_{x=1}^{h} \text{Int}(CS_{xy})_{m,n} \tag{4.4}$$

where h is the total number of abstract services and CS_{xy} is the selected component service (e.g., suppose that it is the yth component service) for the xth abstract service

4.6 Debt-Aware Recomposition

Deriving from the model of service debt, together with the time-series prediction, we developed a technical debt-aware decision approach to recompose services as part of `DebtCom`. In this approach, the service debt model is used to quantify the debt and utility for the period since the last recomposition. The quantified results would be used to compare with the quantification of future debt and utility, supported by the time-series prediction, and thereby taking into account the long term utility. The outcome is then used to trigger the optimisation of recomposition plan if there is needed.

4.6.1 Utility Model

To this end, quantifying the utility of service composition is important. To begin with, the revenue and the operation cost, which are fundamental parts in the utility of service composition, can be computed as follows:

$$R(CS_{xy}) = W_t \times C_{tenants} \tag{4.5}$$

$$C(CS_{xy}) = W_t \times C \tag{4.6}$$

whereby W_t indicates the requests workload at time t, $C_{tenants}$ is the charge to the tenants per request, which directly contribute to the revenue generated by the composite service. C is again the cost per request to the SaaS provider for using a component service and its infrastructure.

The utility at the nth timesteps, denoted as U_n, can be calculated as:

$$U_n = \sum_{x=1}^{h} R(CS_{xy}) - \sum_{x=1}^{h} C(CS_{xy}) - D_{m,n} \tag{4.7}$$

where x is again the total number of abstract services. In particular, such an equation can measure the actual utility of the service composition at time n, including the service debt accumulated from time m to n. Further, with the support of the time-series prediction, the utility can be used to quantify the future timesteps.

4.6.2 Good and Bad Debt

Our debt-aware trigger leverages on the notions of good and bad debt to drive the recomposition. According to our definition about the good and bad debts in Section 4.3.2, the debt depends on the service utility and service debt over a period of time. In particular, the current service debt from the period between the last recomposition point (time m) and n, denoted as $D_{m,n}$, is good or bad with respect to a future time $n + k$ can be determined as follows:

$$D_{m,n} = \begin{cases} D^{bad} & \text{if } U_n > U_{n+k} \text{ and } D_{m,n} < D_{n,n+k} \\ D^{good} & \text{otherwise} \end{cases} \tag{4.8}$$

whereby U_n and $D_{m,n}$ is the utility for time n and service debt from the last point of recomposition m to time n, receptively. Similarly, U_{n+k} and $D_{n,n+k}$ are respectively the estimated utility at time $n + k$ and service debt between time n and time $n + k$. When $U_n < U_{n+k}$ and $D_{m,n} > D_{n,n+k}$, the implication is that the estimated utility and

accumulated service debt between n and $n+k$ would become better, therefore the current service debt should be accepted. This is because the service debts would be paid off by time $n + k$, leading to an anticipated improvement on the overall utility. Otherwise, the service debt would not be paid off at time $n + k$, in which case another recomposition process should be triggered to seek alternative breakthrough.

4.6.3 Trigger and Decision Making of Recomposition

Deriving from the service debt model and time-series prediction, the basic idea of the debt-aware trigger in `DebtCom` is that if the current debt (at time n) is 'good' with respect to a future point in time, denoted as $n + k$, then no recomposition is needed. Otherwise, a recomposition should be trigger at the furthest future point from time n to $n + k$ by which the current debt is considered as 'good', as this is the longest period of time before the current debt becomes bad.

The algorithmic procedure of the Debt-Aware Recomposition Trigger and decision making process have been shown in Algorithm 2, which runs on the next timestep after each recomposition triggered. As can be seen, we calculate the utility since the last recomposition point till now as the current utility, denoted as U_n; the service debt for the same period is denoted as $D_{m,n}$ (line 4-5). Likewise, by leveraging the time-series prediction up to the future timestep $n + k$, the utility at time j ($n < j \leq k$) and service debt up to that timestamp can be estimated, denoted as U_j and $D_{n,j}$, respectively (line 7-8). From the current time n to a future timestep $n + k$, comparing the above utilities and service debt values allow us to verify the need of recomposition based on the future opportunity on value creation at each point in time in the future, as shown at line 9. In particular, if the accumulated debt $D_{m,n}$ is recorded as 'good' with respect to a future timestep $n + k$, then no further action is required (line 18-20) and the loop breaks.

Otherwise, the $D_{m,n}$ would be recorded as 'bad' with respect to $n + k$ (line 10-12), in which case the loops continue backwards till time $n + 1$, and the recomposition would be triggered at the furthest timestep based on which the $D_{m,n}$ is considered as 'good'

Algorithm 2: Debt-Aware Recomposition Trigger

Input: W_c, W_j, P_c, S
/* W_c:Current workload, W_j:Predicted workload at time j, P_c:current composition plans, S:Set of possible composition plans */
Output: P_j
/* P_j:Optimized composition plan for time j */
Initialization: $U_n \leftarrow 0, D_{m,n} \leftarrow 0, U_j \leftarrow 0, D_{n,j} \leftarrow 0, D^{good} \leftarrow 0, D^{bad} \leftarrow 0$
/* U_n and $D_{m,n}$ are the current utility and debt from last point of recomposition time m to current time n; U_j and $D_{n,j}$ are the predicted utility at j and debt from time n to j, where $n < j \leq n + k$; D^{good} and D^{bad} are the counter of good and bad debt, respectively */
$D_{m,n} \leftarrow$ calculateDebt(P_c, W_c)
/* using equation (4.2), (4.3) and (4.4) */
$U_n \leftarrow$ calculateUtility(P_c, W_c, $D_{m,n}$)
/* using equation (4.5), (4.6) and (4.7) */
for $j \leftarrow n + k$ ***to*** $n + 1$ **do**
    $D_{n,j} \leftarrow$ calculateDebt(P_c, W_j)
    $U_j \leftarrow$ calculateUtility(P_c, W_j, $D_{n,j}$)
    if *($U_n > U_j$* ***and*** *$D_{m,n} < D_{n,j}$)* **then**
        /* using equation (4.8) */
        if $j == n + k$ **then**
            $\mathrm{D}^{bad}++$;
        end
        if $j == n + 1$ **then**
            $S'_{demand} \leftarrow \frac{W_j}{j - n}$
            $P_j \leftarrow$ optimize(S, S'$_{demand}$)
        end
    else
        if $j == n + k$ **then**
            /* No recomposition needed */
            $\mathrm{D}^{good}++$;
        else
            $S'_{demand} \leftarrow \frac{W_j}{j - n}$
            $P_j \leftarrow$ optimize(S, S'$_{demand}$)
        end
        break;
    end
end
return P_j

(line 20-23). If all the timesteps between n and $n+k$ would make $D_{m,n}$ 'bad', then the recomposition happens at the next timestep $n+1$ (line 13-16).

Here, the actual recomposition process is based on the search-based evolutionary optimisation, as derived from our previous work [23]. It is worth noting that, the k value controls the preference between short-term advantages and long-term benefit. A larger k implies stronger preference towards long-term benefit, in which case it is likely that less number of recompositions is required but could intentionally accept more bad debt. On the other hand, smaller k favors short term advantages by taking relatively immediate recomposition, which could hinder the benefits in long-term and generate too much operation cost. Indeed, it is possible that the benefit of `DebtCom` can be related to k, and therefore in Section 4.8.8, we experimentally examine the sensitivity of `DebtCom` to k in terms of both the utility and running time.

4.7 Architecture of `DebtCom`

This section presents the architecture of `DebtCom` for debt-aware service composition in SaaS cloud. This architecture is designed into three hierarchical levels: *runtime management level*, *Service execution level*, and *Back-end process and data repositories level* as shown in Figure 4.6. Briefly, in the following, we discuss the general interaction between the components of these levels.

4.7.1 Runtime Management Level

In the SaaS cloud, the end-users can be distinguished based on what type of SLA they have retained, which is often handled by the *Service Broker*. At this level, the services offered by SaaS provider have been designed into a abstract business workflow. Moreover, abstract business workflow represents the interaction of services in the composition structure. At runtime, *Service Broker* invokes the business process workflow component based on the request type. The *Adaptation Manager* is the core component where our debt-aware

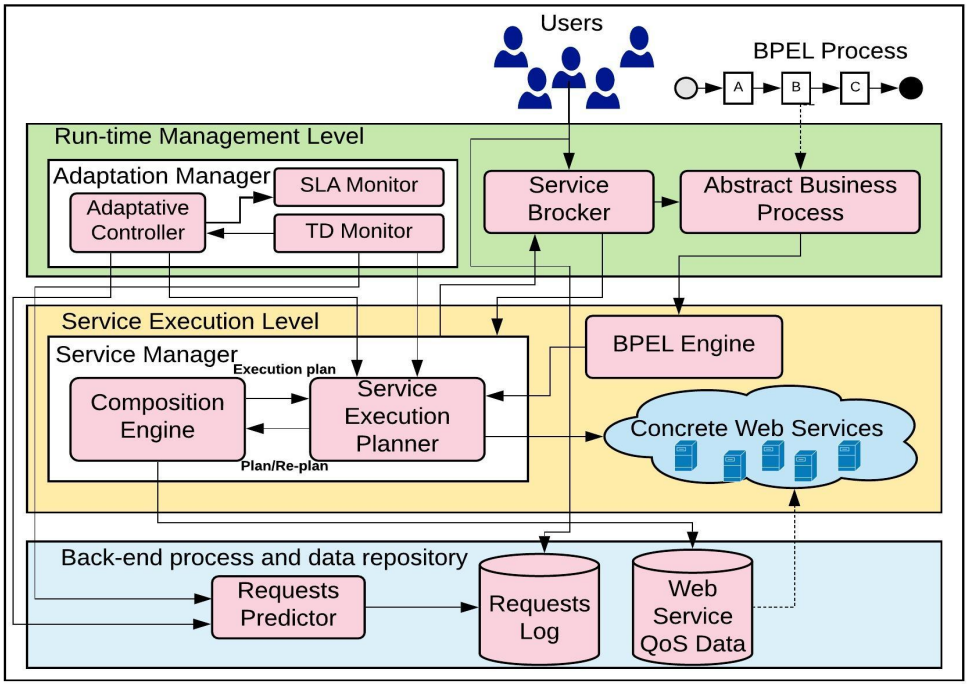


Figure 4.6: The architecture of `DebtCom`

trigger is implemented. In particular, it is responsible for carrying out runtime adaptation action (e.g., service recomposition) while determining the need of a new composite service plan in order to meet the end-users requirements. As shown in Figure 4.6, *Adaptation Manager* comprises three sub-components: *Adaptive Controller*, *SLA Monitor* and *TD Monitor*. Here, the *SLA Monitor* observes the runtime behaviour of service composition and proactively captures dynamic changes (e.g., workload or arrival/departure of users in service pool) in the execution environment that may contribute to SLA violation. Similarly, the *TD Monitor* examines the running service composition from a debt point of view by using the service debt model discussed in Section 4.5. In particular, the *TD Monitor* continuously observes the accrued debt and service utility carried by a currently executed service composition plan. Furthermore, it interacts with the *Request Predictor* from the lowest level, supported by time-series prediction, for monitoring the predicted workload over the composite service, based on which the proactive estimation of the future service debt and utility becomes possible.

Finally, the *Adaptive Controller* takes the information from the *SLA Monitor* and *TD Monitor*, after which it determine whether to trigger the recomposition based on the the approach discussed in Section 4.6.

4.7.2 Service Execution Level

When a recomposition is indeed required, this level enables runtime decision making for optimizing the composition plan and invokes component services in the service composition. The *BPEL Engine* is a software platform (e.g., WSO2 BPS) that executes the business process [154, 155], which represents the composite service produced by the *Service Execution Planner* as requested by the *Service Broker*. The *Service Execution Planner* also dynamically binds the end-users' request to the endpoint that exposes the service operation. The service endpoints are identified and selected from the *Concrete Web Service* pool based on the service composition plan generated by the *Composition Engine*.

In `DebtCom`, we design the *Composition Engine* based on our prior work [23], which is an evolutionary algorithm based optimisation approach. It is worth noting that the *Composition Engine* is triggered only when the *Adaptation Manager* from the above level requires a new composition plan.

4.7.3 Back-end Process and Data Repository Level

Here, as discussed in Section 4.4, the *Request Predictor* examines the past patterns of requests workload (*Requests Log*) generated by the system and predicts the requests workload over the executed composite service. The QoS values of each component service are stored in the *Web Service QoS Data* repository (e.g., WS- Dream QoS dataset [19]).

Table 4.1: Parameters of the experiments

Parameters	Numerical Value
C_{cpu}: Recomposition cost (e.g., engineering efforts plus CPU execution cost)	0.0025 ($)
C: Per request execution cost	0.0015 ($)
$C_{tenants}$: Per request tenant's cost	0.0025 ($)
P: Penalty per request	200% of its cost
R_{SLA}: Response time (SLA)	1 seconds

4.8 Experimental Evaluation

In the experiments, our goal is to assess the effectiveness of `DebtCom` in contrast to the baseline and state-of-the-art approaches for service composition in SaaS cloud. Further, we seek to understand whether the individual components of `DebtCom`, i.e., the time-series prediction and the debt-aware trigger, can indeed create benefit. Specifically, our experiments aim to answer the following research questions.

- **RQ 4.1:** How accurate does `DebtCom` predict the workload?
- **RQ 4.2:** Whether `DebtCom` can outperform the traditional baseline approach?
- **RQ 4.3:** In contrast to the state-of-the-art, whether the time-series prediction and the debt-aware trigger in `DebtCom` can create benefit individually?
- **RQ 4.4:** What is the running overhead of `DebtCom`?
- **RQ 4.5:** What is the sensitivity of `DebtCom` to the k value?

4.8.1 Experimental Setup

For experimental purpose, we developed an e-commerce system, which is formed as a service composition where there are 10 abstract services connected by *sequential* connectors. In particular, to emulate an environment of SaaS cloud, we deployed 100 web services over 10 Docker containers and each web service exhibits the differentQoS which is associated

with real-world WSDream dataset [19]. The relevant setups of the subject service systems have been shown in Table 4.1, which are the results of several runs of trial-and-error and tend to be the most reasonable settings as we observed in our experiment runs.

To emulate realistic workload for each component service, we extract the FIFA98 World Cup website trace [20] for the length of 6 hours, which forms the workload dataset. We pre-processed the first 4 hours workload trace as the samples for training the prediction model, while the remaining 2 hours workload data, which equals to 7200 seconds, is used for testing the accuracy. In `DebtCom`, we feed the training data into the ARFIMA, which is implemented using the `arfima` package in `R` [21]. In all experiments, the k, which determines how many future timesteps to be predicted, is set to 5. This means that `DebtCom` predicts the service debt associated with 5 timestep ahead and take it into account when deciding whether to trigger recomposition. Notably, the k value of 5 is the most ideal trade-off between short-term advantages and long-term benefits, achieving the best utility as discussed within the the sensitivity analysis of `DebtCom` in Section 4.8.8. Note that we use a sampling interval of 1 second in our experiments.

All experiments were carried out on a machine with Intel Core i7 2.60 GHz. CPU, 8GB RAM and Windows 10.

4.8.2 Comparative Approaches

According to the literature, we compare `DebtCom` with three different approaches for service composition in the SaaS cloud. They are specified as follows.

- `Baseline:` We implemented a traditional service recomposition approach from the literature as the baseline [14]. In the approach, a neighborhood region of component services is predefined for each abstract services. The recomposition occurs whenever the violation of SLA has been detected, after which an exhaustive search is conducted to find the best composition plans based on different neighborhood regions, which forms a relatively small search space.

- `Passive:` Another state-of-the-art method that triggers the recomposition based upon the detection of SLA violation [104]. In this work, to achieve a fair comparison, we have applied the evolutionary optimisation approach to search the composition plans, which are equivalent to `DebtCom`.

- `Proactive:` This is the state-of-the-art method that triggers recomposition when the workload is predicted to cause SLA violation [89]. However, the debt model is not explicitly captured during the triggering process. Again, we use the same ARFIMA for workload prediction as `DebtCom`. Further, similar to `Passive`, the acutal optimisation mechanism is also the same.

4.8.3 Metrics

The evaluations of `DebtCom` have made use of the following metrics:

- **Accuracy:** By using Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), we assess the accuracy of the ARFIMA in `DebtCom` for predicting workload of component services.

- **Utility:** For all approaches, we plot the utility of service composition over all timesteps, using Equation 4.7. We show both the individual value (i.e., for each timestep) and accumulated result throughout the time series.

- **Service Debt:** We examine the service debt incurred for all timesteps by using Equation 4.4. Again, we plot both the value for each timestep and accumulated result throughout the time series.

- **Operation Cost:** We assess the resulted cost for all timesteps, using Equation 4.6. Simlar to the others, we plot both the value for each timestep and accumulated result throughout the time series.

- **Running Overhead:** We evaluate the running overhead of the approaches in terms of the required running time.

Table 4.2: Accuracy of time-series prediction for workload

	MAE	RMSE	Theil Coefficient
Request Workload	4.0190	5.0817	0.7532

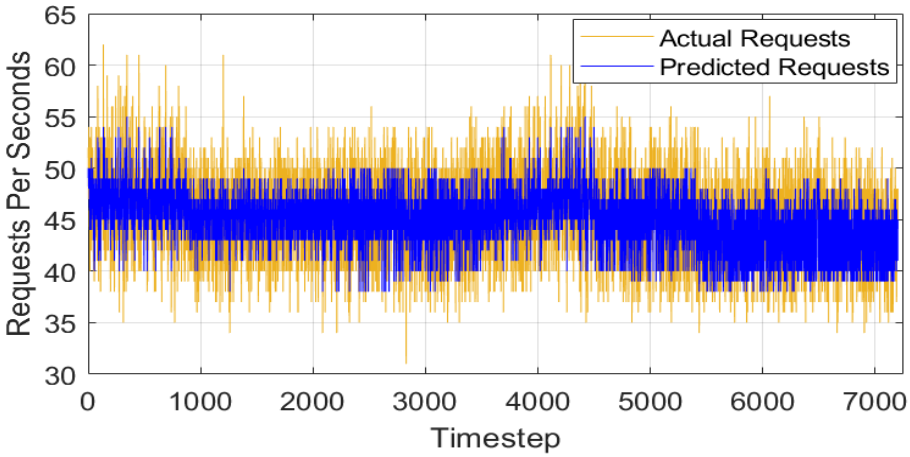


Figure 4.7: Predicted and actual workload on the component services

4.8.4 RQ 4.1: Accuracy on Workload Prediction

To answer **RQ 4.1**, we plot the mean accuracy when `DebtCom` predicts the workload for all component services using different metrics, as shown in Table 4.2. Considering that the general workload varies between around 35 and 60 requests per second, the MAE and RMSE are in fact relatively low and thus the accuracy is acceptable. In addition, we also report on the Theil's coefficient, which indicates a good prediction if it lies between 0 and 1; or the prediction is deemed as poor otherwise [156]. As can be seen, the resulted Theil's coefficient is in between 0 and 1, and thereby suggesting a sufficient accuracy.

As a more detailed example, Figure 4.7 illustrates the workload trace for a selected component service. As we can see, although there are some deviations between the predicted and the actual workload, the prediction has been able to capture the general pattern of trace, e.g., the spikes between 4000 and 5000 second points. We therefore conclude that:

Answering RQ 4.1: The workload prediction of ARFIMA in `DebtCom` is sufficiently accurate, as the deviation is small and the pattern of trace can be generally captured.

Table 4.3: Identified good and bad debt

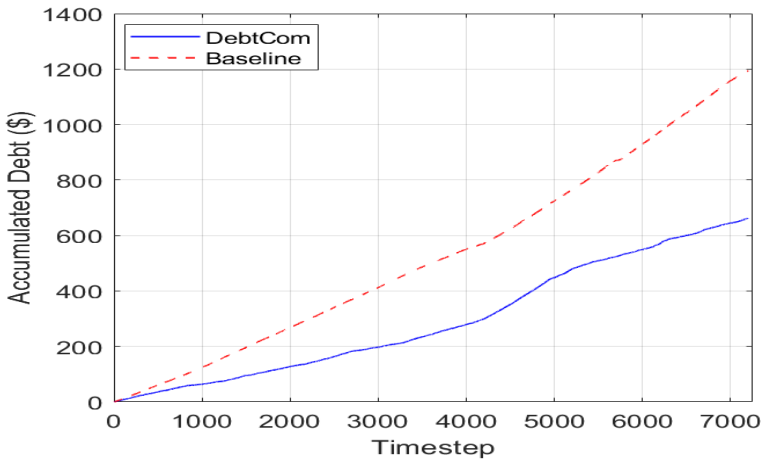
Debt Type	DebtCom	Baseline	Debt Impacts
Total Good Debts	1417	1132	Improving overall service utility
Total Bad Debts	983	1268	Degrades overall service utility

4.8.5 RQ 4.2: Results of DebtCom against Baseline

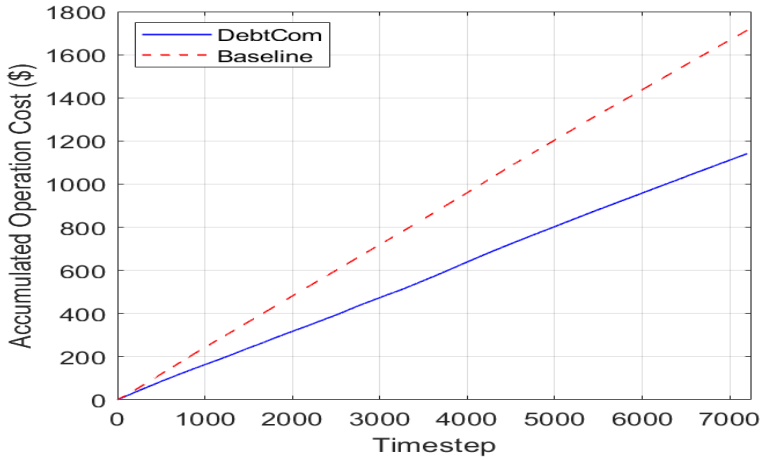
To investigate **RQ 4.2**, we compare DebtCom with the Baseline approach as discussed in Section 4.8.2. In particular, we assess their utility, service debt and operation cost for recomposing the services under the testing period of the workload.

Figure 4.8 shows the accumulated utility, service debt and operation cost of both approaches. As we can see, in contrast to Baseline, DebtCom performs significantly better on reducing the accumulated debt while keeping less accumulated operation cost. This has in turn, leading to considerably better result on the accumulated utility. Notably, the improvement of DebtCom on the utility and debt can be achieved with even less operation cost. To conduct a more detailed review, in Figure 4.9, Figure 4.10 and Figure 4.11, we illustrate the service debt, operation cost and utility measured at each timestep. It is clear to see that DebtCom outperforms Baseline on every timestep in terms of the operation cost. As for the service debt and overall utility, DebtCom is only slightly better before the point of 4000 second, because of the fact that the workload fluctuation till that point is relatively light. However, following the spike between 4000 and 5000 second points, the superiority of DebtCom becomes much more obvious, as the service debt and utility are both significantly improved for the long term. All the above evidence the ability of DebtCom to handle sudden changes, especially for making decision of wether to recompse that takes the long term benefits into account.

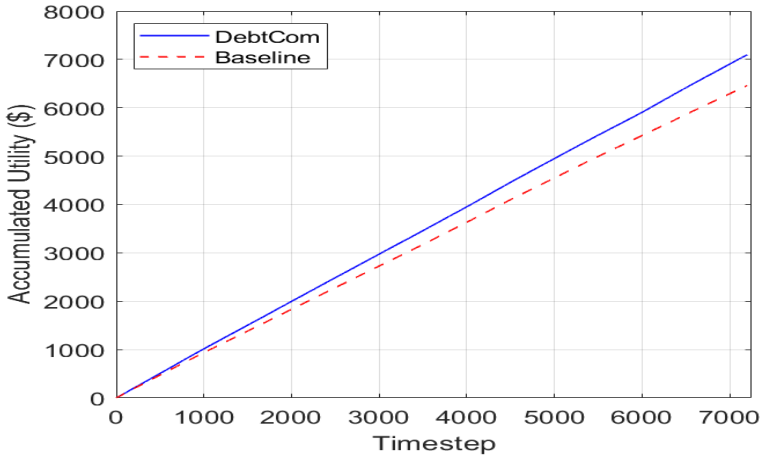
In Table 4.3, we compare the number of good/bad service debt achieved by DebtCom and Baseline. To achieve a fair comparison for Baseline, we assess its debt only on the



(a) Accumulated debt

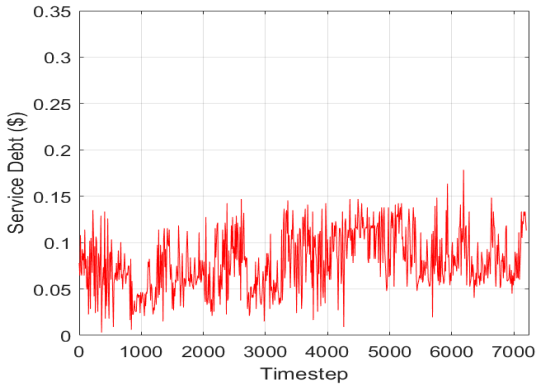


(b) Accumulated operation cost

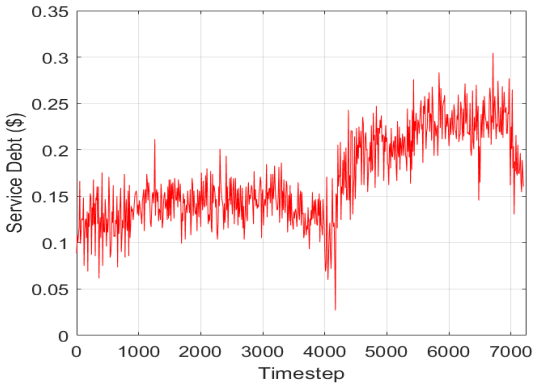


(c) Accumulated utility

Figure 4.8: Accumulated debt, accumulated operation cost and accumulated utility achieved by `DebtCom` and `Baseline` over all timesteps

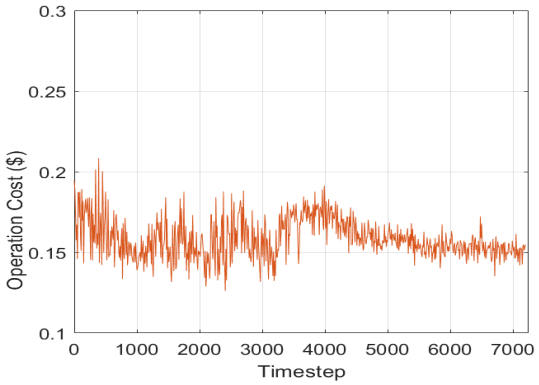


(a) Service Debt using `DebtCom`

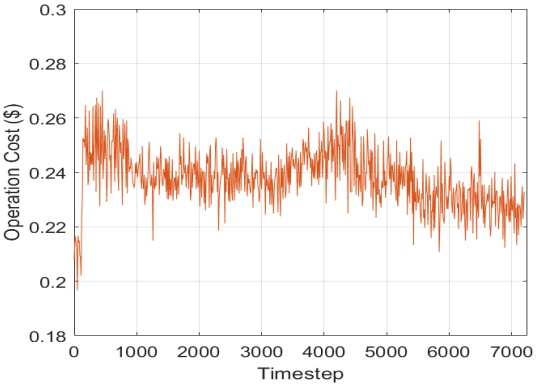


(b) Service Debt using `Baseline`

Figure 4.9: Service debt using `DebtCom` and `Baseline` over all timesteps

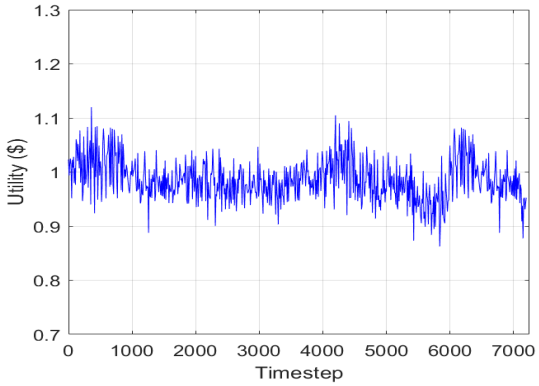


(a) Operating Cost using `DebtCom`

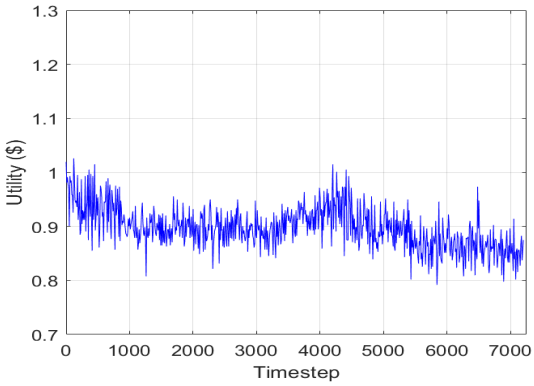


(b) Operating Cost using `Baseline`

Figure 4.10: Service operating cost using `DebtCom` and `Baseline` over all timesteps



(a) Service Utility using `DebtCom`



(b) Service Utility using `Baseline`

Figure 4.11: Service Utility yields `DebtCom` and `Baseline` over all timesteps

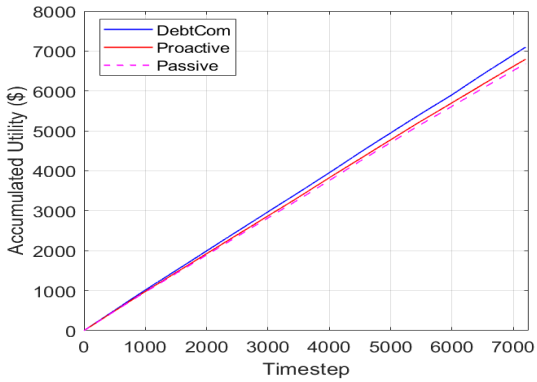
timesteps that DebtCom has checked whether the current debt is good or bad, and thereby the total number of debt to be compared is equivalent. As can been seen, Baseline produce more bad debt than the good ones, i.e., 1132 against 1268; while DebtCom achieves 1417 good debt, which is around 44% more than the 983 bad debt. This is a significant improvement, as higher number of good debt implies that DebtCom requires less number of recomposition, as each composition plan is more sustainable, thanks to the awareness of debt enabled by our service debt model. To conclude, we can summarize that:

Answering RQ 4.2: DebtCom performs significantly better than Baseline on the utility, service debt and operation cost. The benefits cover not only the accumulated results, but also the outcome of each individual timestep. Noteworthily, the benefit of DebtCom can be produced with less cost/number of recomposition, as each composition plan is more sustainable.

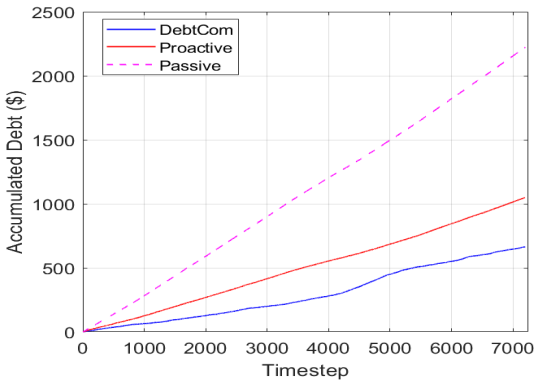
4.8.6 RQ 4.3: Effectiveness of Workload Prediction and Debt-Aware Trigger in DebtCom against Passive and Proactive

To assess the effectiveness of workload prediction and debt-aware trigger, which are the core components in DebtCom, we compare the results between Passive and Proactive, as well as those between Proactive and DebtCom. From Figure 4.12, we see that Proactive achieves better utility with less service debt than that of Passive. This indicates that the workload prediction is indeed beneficial to the service composition. It is also obvious that DebtCom outperforms Proactive on both metrics, which proves that the debt-aware trigger, supported by the service debt model and workload prediction, create greater benefit in the long term.

When observing the result for each timestep, as shown in Figure 4.13, Figure 4.14 and Figure 4.15, similar conclusion can be drawn. In particular, the ability of prediction in Proactive makes it robust to the spike after 4000 second point, but the fact that it only

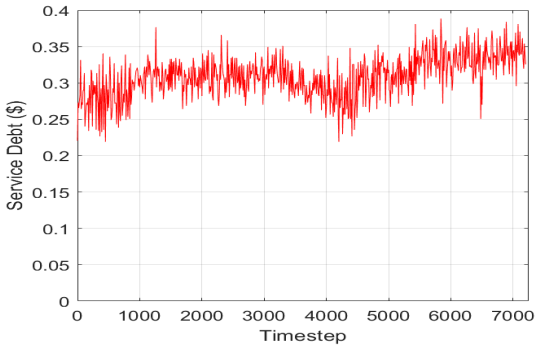


(a) Accumulated utility

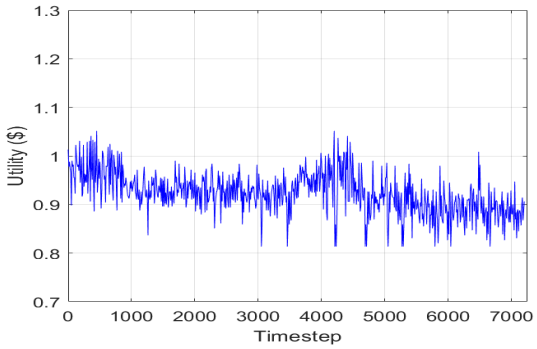


(b) Accumulated debt

Figure 4.12: Accumulated utility and accumulated debt achieved by `Passive`, `Proactive` and `DebtCom` over all timesteps

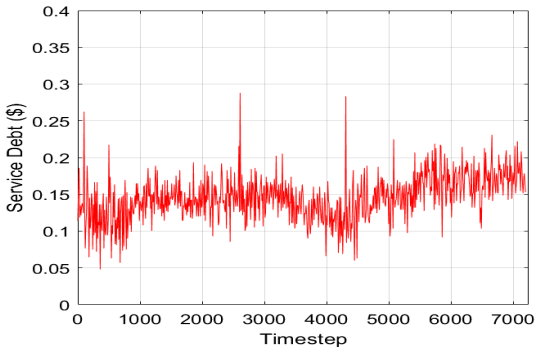


(a) Service Debt

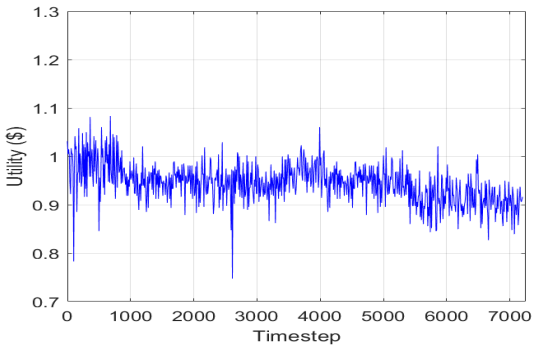


(b) Utility

Figure 4.13: Service utility and debt achieved by `Passive` over all timesteps



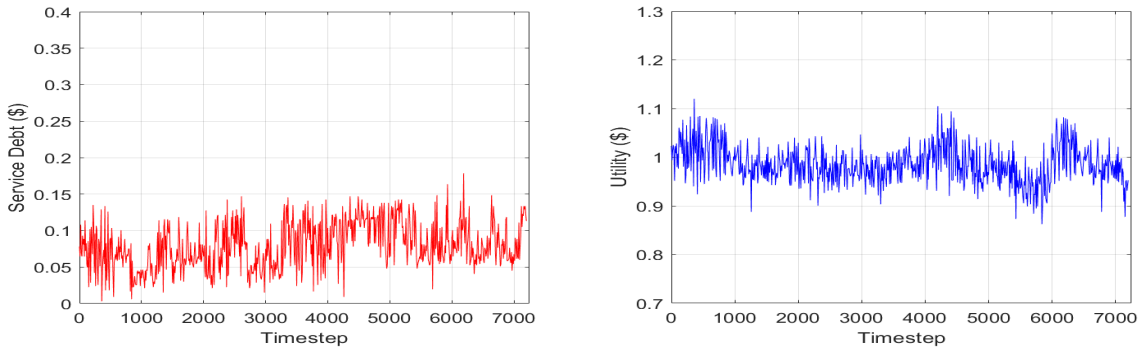
(a) Service Debt



(b) Utility

Figure 4.14: Service utility and debt achieved by `Proactive` over all timesteps

aims for short time benefit has caused a few suddenly increased debt (sudden drop on the utility). In contrast, `DebtCom` does not suffer such issue, thanks to the service debt model. As a result, we conclude that:



(a) Service Debt (b) Utility

Figure 4.15: Service utility and debt achieved by `DebtCom` over all timesteps

Answering RQ 4.3: Both the workload prediction and debt-aware trigger in `DebtCom` are effective in reducing the service debt, leading to better utility of service composition in SaaS cloud in contrast to other approaches.

4.8.7 RQ 4.4: Running Overhead of `DebtCom`

To understand **RQ 4.4**, we evaluate the running overhead of the decision making process in `DebtCom`, including both the reasoing process of service debt and the optimisation process that find the acutal composition plan. We proceed such by comparing `DebtCom` with `Baseline`. To this end, we run both approaches for 30 times and report on the running time required.

As shown in Figure 4.16, we see that `DebtCom` clearly runs faster than `Baseline`. Although the margin differs in the scale of milliseconds, it is worth noting that the need of recomposition can be rapid in service composition at SaaS cloud, which implies a matter of 10ms faster can be seen as a considerable improvement. For **RQ 4.4**, our answer is that:

Answering RQ 4.4: `DebtCom` runs considerably faster than `Baseline` when recomposing services in SaaS cloud.

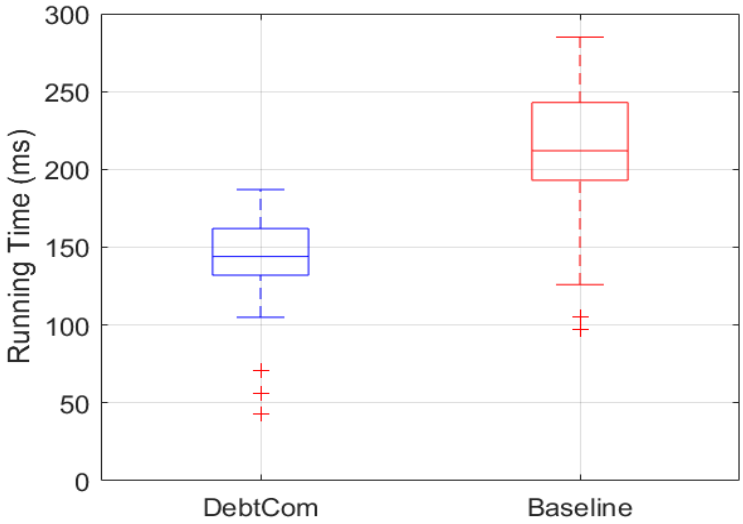
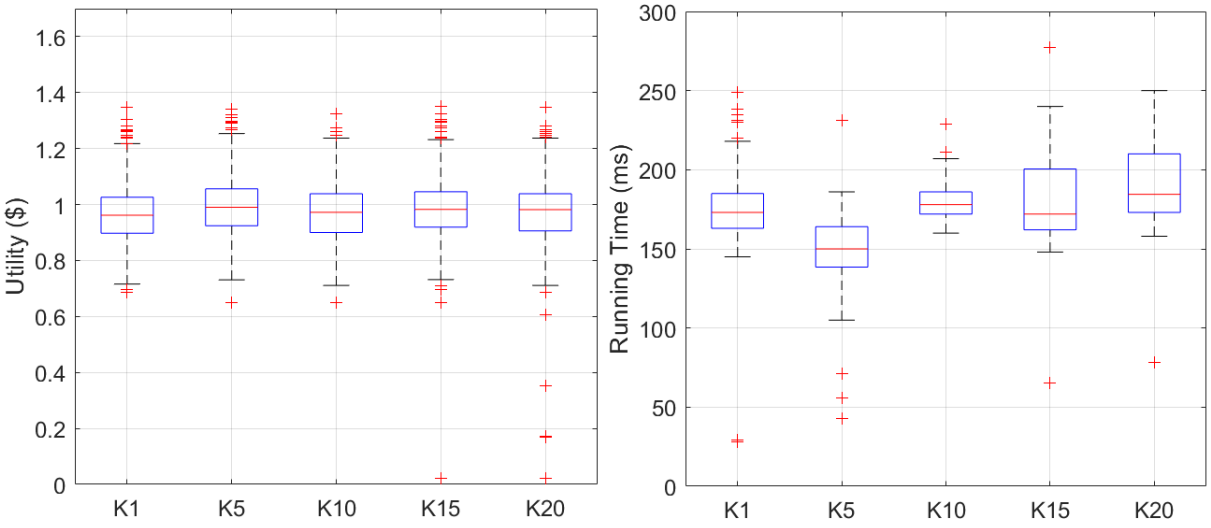


Figure 4.16: Running time on both approaches (Comparisons between DebtCom and Baseline statistically significant (p <.05) using Kruskal Wallis test)

4.8.8 RQ 4.5: Sensitivity of `DebtCom` to k Value

To answer **RQ 4.5**, we compare the utility and running time of `DebtCom` under five different k values that represent different preference between short-term advantages and long-term benefits. In particular, we repeat 120 runs for assessing running time and 7200 timesteps for the utility. The boxplots of the results are shown in Figure 4.17. As can been seen, `DebtCom` can be indeed sensitive to the k value, in which $k = 5$ tends to be the optimal setting, but neither the utility nor the running time exhibit clear monotonic trace. The results also suggest that both too small and too large k could be harmful, as they failed to gain long-term benefits and accept too much bad debt, respectively.

Clearly, although the case of $k = 5$ is better than the others on both metrics, their margins tend to be relatively small. Therefore, to confirm the statistical significance on the sensitivity of `DebtCom` to k value, we perform Kruskal Wallis test and calculate the η^2 as the effect size, which can be interpreted following the guidance by Tomczak and Tomczak [157]. The results reveal that the sensitivity of `DebtCom` to k value is statistically significant on both utility and running time, with $p < .05$ and non-trivial effect size. In conclusion, the answer for **RQ 4.5** is that:



(a) Utility (p <.001; η^2=.013 (small)) (b) Running time (p <.001; η^2=.294 (large))

Figure 4.17: Sensitivity of `DebtCom` to k values in terms of utility and running time.

Answering RQ 4.5: `DebtCom` is indeed sensitive to the k value in terms of both utility and running time, with statistical significance and non-trivial effect size. In particular, both sets of sensitivity exhibits non-monotonic traces, which implies potentially complex trade-off between short-term advantages and long-term benefits.

4.9 Threats to Validity

Threats to construct validity is used to determine whether the adopted metrics can undoubtedly reflect what we aim to measure. In this paper, we set up our experiments with a broad range of metrics for evaluating different aspects of `DebtCom`, including accuracy, utility (e.g., generated revenue), operation cost, service debt, and running overhead.

Threats to internal validity can be mainly related to the value of the parameters for the `DebtCom`. Particularly, the setup has been designed in a way that it produces good trade-off between the quality of services composition and the recomposition overhead. Further, threats to internal validity could be related to the randomness of the results obtained from different runs. Indeed, the actual optimisation of recomposition plan is

achieved by using our prior work [23], which relies on stochastic algorithm. To mitigate such, we repeat all the experiments across different timesteps and runs. We have also assess the sensitivity of `DebtCom` to the k value, which determines the preference between short-/long-term benefits, based on statistical analysis and effect sizes.

Threats to external validity can be associated with the testing environment and the dataset that are used in this experiment. To improve generalization of the experimental evaluation, we developed a real-world ecommerce system as a testing environment, with up to 10 abstract service, each of which has possible 100 component services and 10 dockers. Further, `DebtCom` has been evaluated on the real-world WSDream dataset [19] and FIFA98 workload trace [20] .

4.10 Summary

In this chapter, we propose a technical debt aware framework, namely DetbCom, for service recomposition in SaaS Cloud. In particular, we transform the notion of technical debt metaphor to form a new model called service debt, which fits explicitly within the context of service composition. Such a service debt model, together with time-series prediction using ARFIMA, forms a utility model based on which an algorithm is designed to make decision about when to trigger recomposition. Experiments have been conducted under a large scale service system based on real-world dataset and workload trace. The results confirm the superiority of DetbCom over the state-of-the-art, such that better overall utility can be obtained with less number of recomposition required, which implies that each composition plan becomes more sustainable.

CHAPTER 5

SELF-ADAPTING SERVICE COMPOSITION WITH DEBT-AWARE TWO LEVELS CONSTRAINTS REASONING

In this chapter, we present the technical debt-aware two-level constraints reasoning approach for the long-term based component service selection in the self-adapting service composition in the SaaS cloud − the rapidly changing workload of composite application can easily cause under-/over-utilisation on the component services, which can consequently violates the SaaS provider constraints such as latency and service utilisation. Self-adaptive services composition rectifies this problem, but poses several challenges: (i) the effectiveness of adaptation can deteriorate due to over-optimistic assumptions on the latency and utilisation constraints, at both local and global levels; and (ii) the benefits brought by component service selection is often short term and is not often designed for long-term economic-driven benefits,− a natural prerequisite for sustaining the system. To tackle these issues, we propose a two levels constraint reasoning framework for sustainable self-adaptive services composition, called DATESSO. In particular, DATESSO consists of a refined formulation that differentiates the 'strictness' for latency/utilisation constraints in two levels. To strive for long-term benefits, DATESSO leverages the concept of technical debt and time series prediction to model the utility contribution of the component services in the composition. The approach embeds a debt-aware two level constraint reasoning algorithm in DATESSO to improve the efficiency, effectiveness and sustainability

of self-adaptive service composition. We evaluate DATESSO on an e-commerce system with a real-world WS-DREAM dataset and comparing it with other state-of-the-art approaches. The results demonstrate the superiority of DATESSO over the others on the utilisation, latency and running time whilst likely to be more sustainable.

5.1 Introduction

Service composition allows the software to be built by seamlessly composing readily available service components, each of which offers a different guarantee on Quality-of-Services (QoS), where latency can be of paramount importance [2]. Dynamically composing services is an enabling property for service-based systems (e.g., composite application) supported by Cloud, Edge, and Internet-of-Things environments. However, a known difficulty in service-based systems is the presence of rapidly changing workload, leading to under-/over-utilisation on the services components [6]. At the same time, leading difficulties to satisfy the constraints imposed by SaaS provider under such dynamic changing workload. For example, the increasing workload can enhance the over-utilisation of a services component within a composite service, which in turns, would negatively affect the latency and violate the Service Level Agreement (SLA) [6, 158]. On the other hand, the decreasing workload may lead to under-utilisation of the capacity of component services that violates the service utilisation constraints. To address those issues, self-adaptation on service composition is promising, but the adaptation needs to be effective in terms of long-term based economic-driven component service selection in the composition while being efficient and render benefits over time (i.e., sustainable).

When reasoning about self-adaptation for service composition, there are often two levels of latency/utilisation constraints: the local constraint that relates to the individual constituent services and the global one for the entire service composition. Both of them are critical, as they can affect what the alternative composition plans to be searched during the adaptation [107]. However, existing work on service composition often rely on

over-optimistic assumptions, such that both local and global constraints are hard and can always be satisfied [30, 35, 107, 159, 160]. This can negatively influence the adaptation quality and efficiency, rendering lengthy reasoning process, especially when the given constraints are unrealistic/inappropriate.

To address the above challenges, we propose a framework that leverages **D**ebt-**A**ware **T**wo l**E**vels con**S**traint reasoning for **S**elf-adapt-ing service c**O**mposition (hence called DATESSO). We show that DATESSO can achieve better utilisation/latency in the long term while being faster than state-of-the-art approaches, providing more sustainable self-adaptive service-based systems. In a nutshell, the major contributions of this chapter are summarized as follows:

- Instead of formalizing the constraints at both local and global levels as hard ones, we refine the global constraints as the soft ones. This has enabled us to tailor the reasoning process in self-adaptation and mitigate over-optimism.

- We propose temporal debt-aware utility, a new concept that extends from the technical debt metaphor, to model the long-term benefit of component services selection that constitute to a composition plan.

- Drawing on the above, we design an efficient two level constraint reasoning algorithm in DATESSO that is debt-aware, and utilizes the different strictness of the two level constraints to reduce the search space.

- We evaluate DATESSO on a e-commerce system whose component services are derived from the real-world WS-DREAM dataset [19] and under the FIFA98 workload trace [20]. The results show that, in contrast to state-of-the-art approache [14], DATESSO achieves better utilisation and latency while having smaller overhead, leading to more sustainable self-adaptation in service composition.

5.2 Preliminaries

5.2.1 Self-Adaptation in Service Composition

A service composition is a special software form that consists of a particular workflow of connected abstract services, denoted as $\{a_1, a_2, ..., a_x\}$ [46]. Each of these abstract services can be realized by using a readily available component service selected from the Internet. Typically, there could be multiple component services to be selected, and the yth component service for the xth abstract service is denoted as c_{xy}. Therefore the possible component services for the xth abstract service form a set, denoted as $\{c_{x1}, c_{x2}, ...\}$, each of which has different generic latency guarantee on its capacity. For example, c_{xy} has a capacity to process 50 requests in 0.5 seconds.

In such a context, a SLA may be legally negotiated to ensure the performance of a service composition by contract. The most notable elements in the SLA are the constraints on the utilization of service capacity and the achieved latency level per request, which we will elaborate in the next section. As the workload changes, at runtime, the goal of self-adaptation for service composition is to find the composition plan, $\{c_{11}, c_{23}, ..., c_{xy}\}$, that improves utilization and latency so that they satisfy all the constraints for as long as possible.

5.2.2 Constraints in Service Composition

In service composition, constraints denote the stakeholders' expectation of the latency guarantee. Most commonly, a SLA can define these constraints by specifying the bound of the latency and utilization [160, 161]. For example, a service's latency should not exceed 10s or the utilisation is at least 85% (or translated into 0.85). Typically, there are two levels of constraints:

- **Global constraint:** The global constraint specifies the minimum expectation of latency/utilisation for the entire service composition. It is often the most common

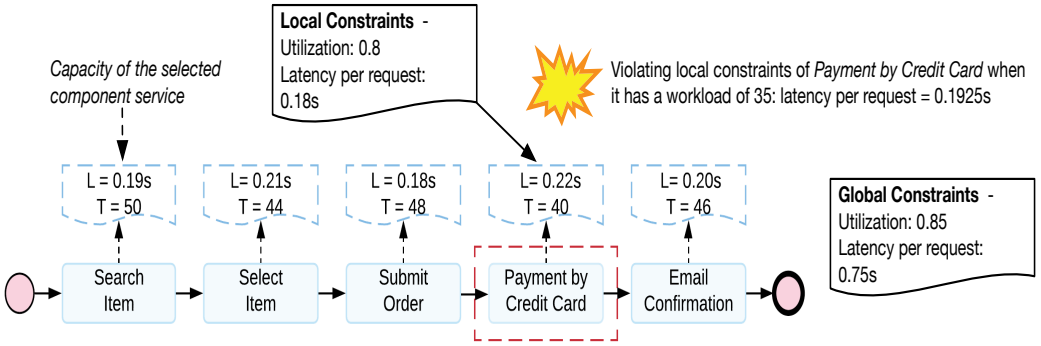


Figure 5.1: A running example of issues in service composition (L and T mean that the selected component service of an abstract service can process all T requests in L seconds)

requirement in a service-based systems [14, 159].

- **Local constraint:** The local constraints are specified for the latency/utilisation on each abstract service[1]. This is important, as each abstract services can be realized by the component service from different parties; any violation of the local constraint would in fact cause severe failure in the composition, leading to an outage [159, 160].

It is worth noting that, satisfying all local constraints does not necessarily mean that the global constraint can be satisfied, since each of the constraints is documented separately [35]

5.2.3 Running Example

In this section, we present a simple example of service composition to explain the problems. As shown in Figure 5.1, there is a service composition in the form of sequentially connected abstract service, each of which has been realized by a particular component service. In this case, each selected component service has its own overall capacity, e.g., the selected component service for Search Item abstract service can process all 50 requests in 0.19 seconds.

As mentioned, each abstract service, along with the entire service composition, are legally documented with separated constraints on the utilization and latency per request,

[1]For latency, this constraint would be applied for each request.

as specified in the SLA [162]. Suppose that in this scenario, the local constraint of utilization and latency of each request for the abstract service Payment by Credit Card could be 0.8 and 0.18 seconds, respectively. Meanwhile, the global constraint of utilization and latency of each request for the service composition is 0.85 and 0.75 seconds, respectively. Given the changing workload, it is likely that either (or both) levels of constraint may be violated, which requires self-adaptation to replace the component services. However, there are two issues with this:

1. In this context, the different constraints are negotiated independently to each others. While it is relatively easy to find the alternative component service that satisfy the local constraints, searching for the composition plan that satisfies the global constraints is difficult, or we may not know whether one exists. As a result, existing approaches that treats both levels of constraints as hard constraints suffers the issue of being over-optimistic: they may struggle to find a satisfactory composition plan, especially under a scenario where such a plan barely exists. Further, this would completely eliminate the composition plan that may cause temporary violation of the global constraint(s), but can create much larger long-term benefits.

2. When self-adaptation is required, a possible component service and the entire composition plan may provide short-term immediate benefit in relieving constraint violation, but it is difficult to know whether such a benefit can be sustainable. In contrast, it is possible to temporally accept a composition plan that may still violate the global constraint(s), but will generate larger benefit in the long term. Therefore, self-adapting service composition without having any guarantee on the long term can lead to frequent adaptations with merely short-term benefits, which generate unnecessary overhead.

The `DATESSO` proposed in this work was designed to explicitly address these two issues in self-adapting service composition.

5.3 DATESSO Overview

Figure 5.2 illustrates the overview of DATESSO. As can bee seen, there are three key stages, namely *Formalization*, *Modeling* and *Reasoning*, each of which is specified as follows:

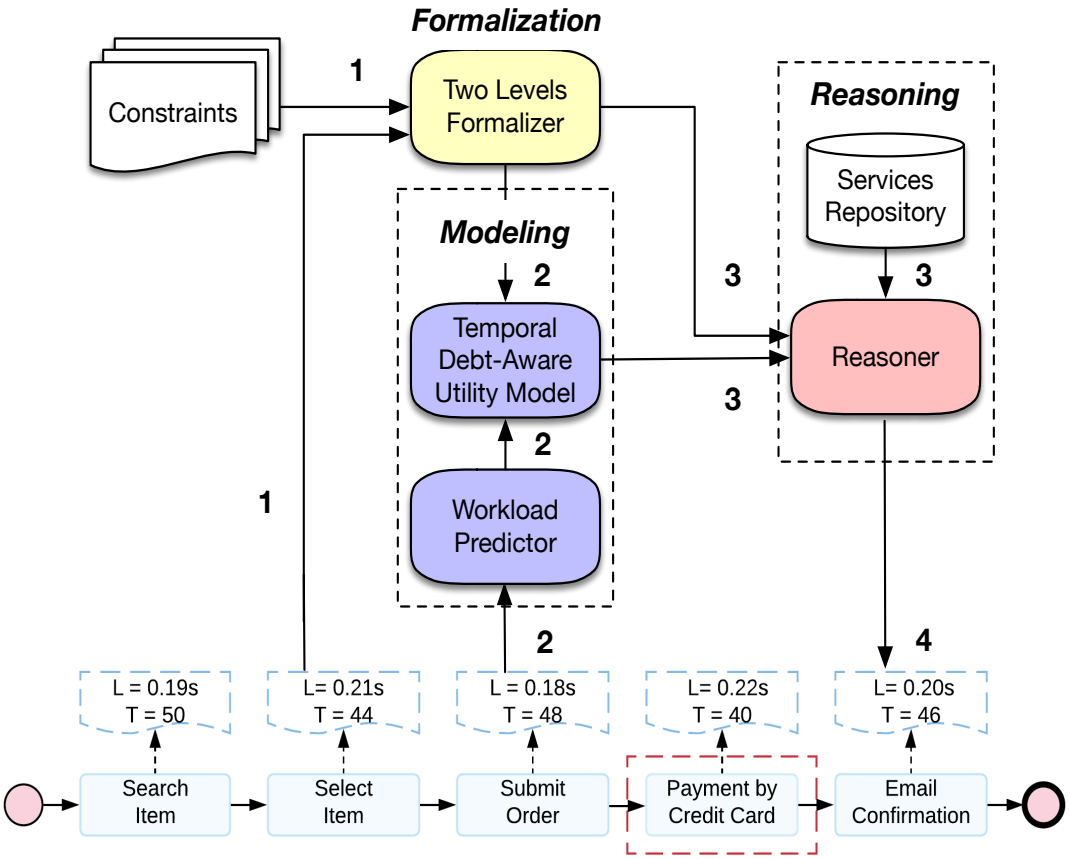


Figure 5.2: The general processes in DATESSO

1. ***Formalization:*** This is the very first stage in DATESSO and it relies on the *Two Levels Formalizer* component. Generally, it has two tasks at step 1: (i) formulating and recording the global/local level constraints as documented in the SLA; (ii) monitoring the service composition and informing the *Modeling* stage, along with any information of the constraints, when any violations are detected. More details are discussed in Section 5.4. Note that here, we trigger adaptation only based on local constraint violations, as we formalize the global ones as soft constraints. However, the global constraint is implicitly considered in the *Reasoning* stage.

2. ***Modeling:*** Once the local constraint violation has been detected, at step 2, the *Workload Predictor* keeps track of the historical workload on each abstract service, and provides a time-series model to be embedded with the constraint information, which together form the temporal debt-aware utility model. A detailed discussion will be presented in Section 5.5

3. ***Reasoning:*** At the final stage, the utility model that is debt-aware, the two level constraints and the *Service Repository* with all possible component services would be exploited by the *Reasoner* at step 3. Specifically, we design a debt-aware two levels constraint reasoning algorithm that (i) enables more efficient processing by reducing the original search space based on the constraint information, and (ii) produces a composition plan that is likely to have the highest long-term benefit, without explicitly using global constraints as caps or thresholds. Such a composition plan would then be sent for execution (step 4). The algorithm will be illustrated in greater details at Section 5.6.

5.4 Two Levels Constraints with different strictness

As mentioned, we consider both local and global constraints for latency/utilisation in the *Formalization* stage of `DATESSO`. Instead of assuming hard constraint for both of them, we treat the global constraint as a soft one, which helps to mitigate the problem of being over-optimistic. The formal model and strictness of the two level constraints are discussed in the following subsections.

For each level, constraint can be related to both utilisation and latency values. The utilisation is a direct measurement of under-utilised situation, whilst the latency value reflects the problem of over-utilisation, as a too high utilisation usually means the component service is over-stressed, which results in latency degradation.

5.4.1 Hard Local Constraints

As discussed in Section 5.2, the local constraint is usually hard [69, 159], which should not be violated. This is because at the service level, any violation of the constraint would in fact cause severe failure in the workflow execution. For example, a violation of latency/utilisation caused by a workload that exceeds the capacity would simply bring the individual service down, which cause outage of the entire service composition.

Locally, for each component service c_{xy} that has a capacity to process $T_{c_{xy}}$ requests in $L_{c_{xy}}$ seconds, we model the normalized constraint ($CL_{c_{xy}}$) on the normalized actual latency of each request ($L_{c_{xy}}$) to be satisfied as below, both of which are within $[0, 1]$[2]:

$$L_{c_{xy}} = \frac{L_{c_{xy}} \times W_{t,c_{xy}}}{T_{c_{xy}}} \leq CL_{c_{xy}} \tag{5.1}$$

where $W_{t,c_{xy}}$ is the workload for the corresponding abstract service (hence for c_{xy} too) at timestep t. Likewise, the local constraint ($CU_{c_{xy}}$) on utilisation ($U_{c_{xy}}$) to be satisfied can be formulated as[3]:

$$U_{c_{xy}} = \frac{L_{c_{xy}} \times W_{t,c_{xy}}}{CL_{c_{xy}} \times T_{c_{xy}}} \geq CU_{c_{xy}} \tag{5.2}$$

Since the local constraints are hard, we say a component service as *feasible* if, and only if, both utilization and latency constraints are satisfied. Otherwise it is termed *infeasible*.

5.4.2 Soft Global Constraints

Unlike existing work that model global constraint as hard threshold, we model its soft version that can tolerate certain violation, with an aim to mitigate the issue of over-optimism. Indeed, the way of aggregating the local latency toward the global value for the entire service composition depends on the connectors, which may be sequential,

[2]Normalization can be achieved by using the lower and upper bounds of possible latency values.

[3]Utilization naturally sits within $[0, 1]$, as any requests that go beyond the capacity would be discarded.

parallel or recursive etc. However, as shown in [163, 164], sequential connector is the most fundamental type and all other connectors can be converted into a sequential one. Therefore in this work, we focus on sequential connector in our models.

Similar to its local counterpart, for all selected component services in the entire service composition, the satisfaction on normalized global constraint ($CL_{global} \in [0, 1]$) and the normalized actual latency of each request ($L_{global} \in [0, 1]$) can be calculated by aggregating the locally achieved latency. Specifically, when all the connectors are sequential or they have been converted into sequential ones, the satisfaction of global latency can be formulated as[4]:

$$L_{global} = \sum_{x} \sum_{y} L_{c_{xy}} \preceq CL_{global} \tag{5.3}$$

Likewise, the global constraint (CU_{global}) on utilisation (U_{global}) to be satisfied can be formulated as:

$$U_{global} = \frac{1}{N} \times \sum_{x} \sum_{y} U_{c_{xy}} \succeq CU_{global} \tag{5.4}$$

whereby N denotes the total number of abstract services. As mentioned, there is no guarantee that satisfying the local parts at component level can lead to global satisfaction. However, it is easy to see that a violation of a global constraint is contributed by some (or all) of the component services selected, even though their local constraints may have been satisfied.

5.5 Temporal Debt-Aware Utility Model

In the *Modeling* stage of `DATESSO`, we propose temporal debt-aware utility model, a notion derived from technical debt metaphor [7], that quantifies the long-term benefit of

[4]We use $\preceq$ to reflect the 'soft' nature of global constraints.

component services selection that support a composition plan. To this end, we adopt the notion of principal and interest [8, 52, 144] to analyze the debt values related to a single component service selection that is feasible. Built on the concept of two level constraints and their different strictness, a debt can quantify each feasible component service's local contribution to the overall debt at the global level over a period of time.

5.5.1 Modeling Temporal Debt Value

5.5.1.1 Principal

The principal, denoted as $P_{c_{xy}}$, is the one-off cost of the processes on adapting a component services c_{xy}. It can be calculated as:

$$P_{c_{xy}} = O_{c_{xy}} \times C_{com} \tag{5.5}$$

Suppose that the searching suitable component service for adding in service composition requires an overhead of 5 seconds (denoted as $O_{c_{xy}}$) and the execution cost of computing resource is \$ 0.005 per second (denoted as C_{com}), then it takes a principal as 5 $\times$ 0.005 = \$ 0.025. Note that $P_{c_{xy}}$ here is a normalized value in the range of $[0, 1]$, based on the lower/upper bounds of the possible execution cost and composition time. The $O_{c_{xy}}$ can be easily known by analyzing the time for previous rounds of composition.

5.5.1.2 Accumulated interest

For searching and selecting feasible component service, we calculate the interests which can be accumulated due to continuous constraint violations. Since the local constraints are hard, there will be no interest incurred directly at this level. However, because we model the global constraints as the soft ones, any violation of a global constraint is contributed by the component services at the local level, even if the local constraint has been satisfied. In particular, according to Equation 5.3 and 5.4, over a period of time, any possible

violation of a global constraint would be contributed by all component services that have local utilization/latency worse than the global constraint, which causes potential interest. With this in mind, the accumulated interests of a component service c_{xy} between timestep n and m can be modeled as:

$$I_{n,m,c_{xy}} = \alpha_{n,m,c_{xy}} + \beta_{n,m,c_{xy}} \tag{5.6}$$

and

$$\alpha_{n,m,c_{xy}} = \sum_{t=n}^{m} (CU_{global} - U_{c_{xy}}),\ \forall t \ \overset{\bullet}{\equiv}\ CU_{global} \geq U_{c_{xy}} \tag{5.7}$$

$$\beta_{n,m,c_{xy}} = \sum_{t=n}^{m} (L_{c_{xy}} - CL_{global}),\ \forall t \ \overset{\bullet}{\equiv}\ L_{c_{xy}} \geq CL_{global} \tag{5.8}$$

whereby $\overset{\bullet}{\equiv}$ represents 'such that'. Hence, $\alpha_{n,m,c_{xy}}$ and $\beta_{n,m,c_{xy}}$ consider only those timesteps between n and m, at which contribution to the possible violation of a global constraint exists. In particular, these equations guarantee that $\alpha_{n,m,c_{xy}} \geq 0$ and $\beta_{n,m,c_{xy}} \geq 0$.

It is easy to know that in general, if $\alpha_{n,m,c_{xy}} = 0$ and $\beta_{n,m,c_{xy}} = 0$, which means c_{xy} does not contribute to any possible global violation at all, then the overall accumulated interest for c_{xy} over a period of time is 0. Otherwise, the interest, incurred by the contribution to the possible violation of either global utilization or latency constraint (or both), would be part of the debt. For example, when $CU_{global} = 0.9$ and $CL_{global} = 0.7$, at a particular timestep t, a feasible component service has utilization and latency of $U_{c_{23}} = 0.7$ and $L_{c_{23}} = 0.85$, respectively. In this case, for any possible violation of the global utilization and latency constraint at this timestep, c_{23} would contribute a total of $I_{t,t,c_{23}} = 0.9 - 0.7 + 0.85 - 0.7 = 0.35$ interest (and thus part of the debt) to cause the violations. The overall interest over a range of timesteps would be the sum of the interest incurred by the above

case under each timestep.

5.5.1.3 Connecting debt and utility

Finally, we calculate the debt for a feasible component service between timestep n and m as:

$$D_{n,m,c_{xy}} = P_{c_{xy}} + I_{n,m,c_{xy}} \tag{5.9}$$

Since both $P_{c_{xy}}$ and $I_{n,m,c_{xy}}$ are normalized or naturally sit between $[0, 1]$, the numeric stability can be improved. Drawing on the above, we then be able to obtain a debt-aware utility score ($S_{n,m,c_{xy}}$) for c_{xy} between n and m, defined as:

$$S_{n,m,c_{xy}} = \sum_{t=n}^{m} U_{c_{xy}} - \sum_{t=n}^{m} L_{c_{xy}} - D_{n,m,c_{xy}} \tag{5.10}$$

A larger $S_{n,m,c_{xy}}$ implies that the component service c_{xy} is more likely to contribute to the satisfaction of global constraints in the long term. Here, it is clear that we will accept certain debt, as long as it can be paid back by achieving better overall utility across the timesteps considered. In this way, during the reasoning process, `DATESSO` is able to quantify the long-term benefit of selecting feasible component service over a range of timesteps, based on which enabling better informed reasoning.

5.5.2 Time-Series Workload Prediction

Predicatively analyzing debt is not uncommon for managing technical debt in software development [7]. Often, the fact of whether a debt can be paid off depends on the present and future values of the debt [146, 165]. This is also an equivalent and important concept in our research, and therefore we seek to predict the future workload of the component services, which in turn, enabling us to select long-term based economic-driven component

service in the self-adaptation process.

In DATESSO, we use Autoregressive Fractionally Integrated Moving Average model (ARFIMA) [166], a widely used time-series model, to predict the workload of each abstract service. It is chosen over its counterparts (e.g., ARMA) because it handles a time-series with long memory pattern well.

Accordingly, for each abstract service that is realized by a component service, we prepared the data at each time point to contain a number of observed requests, which would be used by the ARFIMA to predict the likely requests workload for a future timestep. The general expression of ARFIMA (p, d, q) for the process X_t is written as:

$$\Phi(B)(1-B)^d X_t = \Theta(B)\varepsilon_t \tag{5.11}$$

where $(1-B)^d$ is the fractional differencing operator and the fractional number d is the memory parameter, such that $d \in (-0.5, 0.5)$. The operator B is the backward shift operator. For this, we have $\Phi(B) = 1 - \phi_1 B - \phi_2 B^2 - ... - \phi_p B^p$ is the autoregressive polynomial of order p and $\Phi(B) = 1 + \theta_1 B + \theta_2 B^2 + ... + \theta_q B^q$ is the moving average polynomial of order q. $BX_t = X_{t-1}$ and ε_t represent the white noise process.

5.6 Debt-Aware Two Levels Constraint Reasoning

Drawing on our formalization of soft/hard constraints at two levels, along with the proposed temporal debt-aware utility model, we design a simple yet efficient reasoning algorithm for self-adapting service composition in the *Reasoning* stage. In a nutshell, once violation on local constraints is detected, the algorithm has two main functions that are run in order:

1. **Identification:** In this function, we firstly identify which are the component services that violate the local constraints, as this was what triggered the adaptation.

Then, the identified infeasible component services would need to be replaced, as they also contribute to the likely violation of the global constraint(s).

2. **Search:** Once we identify the set of abstract services whose component service needs a replacement, this function works on each individual abstract service. The aim is to search for the best feasible component service for each identified abstract service, such that it satisfies the local constraint[5] while having the best long-term debt-aware utility, over all timesteps up to the future timestep m (Equation 5.10). As a result, the newly selected component services would less likely to cause local/global constraint violation in the future.

Each of the key steps are discussed in details as follows.

5.6.1 Identifying Infeasible Component Services

As mentioned, since the constraint at local level is hard, the IDENTIFICATION function is designed to filter all the service components that are 'working fine'. In fact, this steps is an effective way to reduce the search space, as only the problematic component services that violates the hard constraints are considered. These infeasible component services can actually contribute to the global constraint violation, if any.

The corresponding algorithmic procedure has been illustrated in Algorithm 3. As can be seen, the returned result is a set, denoted as S_{inf}, that contains every abstract service (i.e., a_x) whose component service becomes infeasible at the current timestep n.

5.6.2 Searching for the Best Long-term Debt-Aware Utility

The special design in the SEARCH function is that, instead of having to examine every combination of the service composition globally, we only search for the component

[5]Given that the local constraint is specified at the local level, there will be at least one readily available component service to satisfy such constraint at a particular timestep, or otherwise the constraint may be too strong and needs to be relaxed.

Algorithm 3: IDENTIFICATION

```
1 Input:
    S: Set of selected component services and their abstract services at current timestep n
2 Output:
    S_inf ← ∅: Set of abstract services whose component service needs a replacement
3 for ∀c_xy ∈ S do
4 |   if (L_{c_xy} > CL_{c_xy} or U_{c_xy} < CU_{c_xy}) then
5 |   |   S_inf ← a_x
6 |   end
7 end
8 return S_inf
```

service with the highest long-term debt-aware utility for each identified abstract service independently.

This is because, according to Equation 5.10, the problem of searching the highest long-term debt-aware utility (between timestep n and m) for the entire service composition can be defined as follow:

$$\textbf{argmax} \sum_{x=1}^{Z} S_{n,m,c_{xy}} \tag{5.12}$$

whereby Z is the total number of abstract services whose component service need a replacement. Clearly, this is a typical linear programming problem, in which achieving the best utility of the service composition is equal to finding the optimal value of each $S_{n,m,c_{xy}}$. From Equation 5.10, we know that the best $S_{n,m,c_{xy}}$ is solely equivalent to the highest debt-aware utility from all the feasible component services of the xth abstract service. In other words, the highest $S_{n,m,c_{xy}}$ can be searched on each abstract service locally, in order to have the highest utility for the service composition globally. With this consideration, our reasoning algorithm decomposes the problem and reduces the search complexity from $O(Y^X)$ (when all combinations need to be searched at the global level) down to $O(Y \times X)$, where X is the number of problematic abstract service, each with Y feasible component services[6].

[6] Y may differ for different abstract services, but in this example we assume that same as our aim is merely to intuitively illustrate the reduction of complexity.

Algorithm 4: SEARCH

Input: R_x: *The set of possible component services for the xth abstract service*
S_{inf}: *Set of abstract services whose component service needs a replacement*
Output: $S_{optimal}$: *Service composition plan with the optimal long-term debt-aware utility between current timestep n and the future timestep m*
for $\forall a_x \in S_{inf}$ **do**
    /* M_x denotes the ordered list of vectors of the feasible component services for the xth abstract service at every timestep from n to a future timestep m */
    /* $\mathbf{S}_{x,t}$ denotes the vector of the feasible component services for the xth abstract service at timestep t */
    $M_x = \{\mathbf{S}_{x,n}, \mathbf{S}_{x,n+1}, ..., \mathbf{S}_{x,m}\}$
    for $\forall c_{xy} \in R_x$ **do**
        for $t \leftarrow n+1$ ***to*** m **do**
            if *(*$L_{c_{xy}} \leq CL_{c_{xy}}$ ***and*** $U_{c_{xy}} \geq CU_{c_{xy}}$*)* **then**
                $\mathbf{S}_{x,t} \leftarrow c_{xy}$
            end
        end
    end
    $M \leftarrow M_x$
end
for $\forall M_x \in M$ **do**
    /* According to M_x, the function GETLARGESTFEASIBLESTEP returns the largest timestep m_x from n such that there is at least one component service that satisfies the local constraint on every timestep between n and m_x */
    m_x = GETLARGESTFEASIBLESTEP(M_x)
    if $m_x < m$ **then**
        $m = m_x$
    end
end
for $\forall M_x \in M$ **do**
    /* According to M_x and the new m, function GETFEASIBLESERVICES returns the component services that satisfy the global constraint on every timestep between n and m */
    S_x = GETFEASIBLESERVICES(M_x,m)
    /* Function SEARCHUTILITY returns the component service with the highest $S_{n,m,c_{xy}}$ for a_x */
    $\mathrm{S_{optimal}} \leftarrow$ SEARCHUTILITY(S_x,n,m)
end
return $\mathrm{S_{optimal}}$

The corresponding algorithmic procedure has been illustrated in Algorithm 4. Specifically, suppose that the S_{inf} has been found by Algorithm 1, and that the current timestep is n and we are interested in the debt up to a given timestep m in the future, there are three important steps:

1. From line 4 to 14, for each problematic abstract service a_x, we firstly construct an ordered list of vectors denoted as M_x. Each vector in M_x has a size of $m - n$ and it contains all the feasible component service for a_x under every particular timestep between n and m.

2. From line 15 to 20, for each M_x, we find the largest timestep m_x since n such that there is at least one feasible component service that satisfies the global constraint on every timestep between n and m_x. Next, we use the smallest m_x across all M_x to serve as the new m. This process ensures that all problematic abstract services would have at least one component service which can be treated as feasible on all timesteps considered. Here, since there is at least one feasible component service for a particular timestep, the worst case would be $m = n + 1$.

3. From line 21 to 24, for each a_x, we find the set of feasible component services (S_x) that satisfy the local constraints on every timestep between n and m. The SearchUtility function searches locally on the set S_x, and returns the one with the highest $S_{n,m,c_{xy}}$ as part of the composition plan. Note that, SearchUtility can be realized by any search algorithm, e.g., exhaustive search or stochastic search like Genetic Algorithm. Since in this work the S_x has been reduced to a computationally tractable size, we simply apply an exhaustive search.

As the global constraints are soft, the reasoning algorithm has never explicitly used them to act as caps or thresholds for the search (like what we did for the hard local constraints), but the global constraints, along with their potential violations contributed by the component services, are implicitly embedded in the debt-aware utility model. In this way, we aim to mitigate the problem of being over-optimism on the global constraint, while

at the same time, promoting larger chance to satisfy the global constraint in the long term.

5.7 Evaluation

To evaluate DATESSO, we design experiments to assess the performance of our technique on self-adapting service composition by means of comparing it with the state-of-the-art approaches. In particular, we aim to answer the following research questions (RQs):

- **RQ 5.1:** Can DATESSO achieve better global utilisation and latency than the state-of-the-art approaches? If so, which parts contribute to the improvement?
- **RQ 5.2:** Is DATESSO more sustainable than the state-of-the-art approaches?
- **RQ 5.3:** What is the running overhead of the reasoning process in DATESSO comparing with the others?

5.7.1 Experimental Setup

For experimental purpose, we developed an e-commerce system, which is formed as a service composition where there are 10 abstract services connected by sequential connectors. In particular, to emulate an environment of SaaS cloud, we deployed 100 web services over 10 Docker containers. All the values of latency and throughput capacity for the component services are randomly chosen from the WS-DREAM dataset [19].

To emulate realistic workload for each abstract service that is realized by a component service, we extracted the FIFA98 trace [20] for the length of 6 hours, which forms the workload dataset. Such a workload trace is used on all the different workflows of service composition. We pre-processed the first four hours of workload trace as the samples for training the time-series prediction model, while the remaining two hours of workload data, which equals to 7200 seconds, is used for testing the accuracy. In DATESSO, we feed the

Table 5.1: Parameters of the experiments

Parameter	**Value**
$CL_{c_{xy}}$: local latency constraint per request	0.09s
CL_{global}: global latency constraint per request	1s
$CU_{c_{xy}}$: local utilization constraint	0.8
CU_{global}: global utilization constraint	0.9
C_{com}: cost of computing resource	\$0.0025
m: future timestep m from current timestep n	$n + 5$

training data into the ARFIMA, which is implemented using the `arfima` package [167] and the FDGPH function in R [149]. The values of p, d and q are also identified therein.

Table 5.1 shows the parameter settings of the SLA used in the experiments, including the executing resource of selecting a component service (C_{com}), the local and global constraint for latency ($CL_{c_{xy}}$ and CL_{global}) and utilization ($CU_{c_{xy}}$ and CU_{global}). For simplicity of exposition, we have set the same local constraint for all abstract services. All the settings above have been tailored to be reasonable throughout the experiments.

All experiments were carried out on a machine with Intel Core i7 2.60 GHz. CPU, 8GB RAM and Windows 10.

5.7.2 Comparative Approaches

To answer all the RQs, we examine the performance of `DATESSO` against the following approaches:

— **Two Level Hard Constraints Approach (TLHCA):** This is similar to `DATESSO`, which differs only on the way about how the strictness of the two levels constraints is formulated. `TLHCA` assumes that both local and global constraints are hard, and thereby in the reasoning algorithm (Algorithm 4), when the final composition plan violates the global constraint (for every timestep between n and the newly defined m) then we examine whether all abstract services have been considered in this run. If not, we then return the algorithm with consideration that all the abstract services are subject to replacement; if all abstract services has been considered but

the global constraint(s) is still violated, we would have no choice but to trigger the adaptation. Here, the adaptation is triggered based on both local and global constraint violations. This approach follows the existing work [159] that makes the same formulation, and by this mean, we aim to examine the usefulness of formulating the global constraints as the soft ones.

— **Debt-Oblivious Approach (DOA):** This is a similar copy of `DATESSO` but without the temporal debt-aware utility model. Instead, `DOA` assumes the predicted utility of the aggregated latency and utilization, i.e., Equation 10 without the debt, which is then used in the reasoning algorithm to find the composition plan for self-adaptation. `DOA` helps us to examine the effectiveness of incorporating debt information for achieving long-term benefit in self-adaptation.

— **Region-Based Composition (RBC)** This is an implementation of a state-of-the-art approach, proposed by Lin et al. [14], that relies on regions, where for each abstract services, the component service is selected according to its region. Each of these regions are clustered based on the historical utilization and latency of the component services. Here, the adaptation is triggered based on global constraint violations only. `RBC` is chosen as it is one of the most widely known representative approaches for dynamic service composition.

5.7.3 Metrics

We leverage the following metrics to assess the results:

— **Global utilisation:** This is the value calculated by Equation 5.4 for each timestep.

— **Global latency:** This is the value calculated by Equation 5.3 for each timestep.

— **Accumulated debt:** Since the interests are accumulated, so does the debt. A lower debt means that component services, which are less likely to contribute to global constraint violation in the long term, are preferred. Therefore, we measure

the accumulated debt of the service composition from the beginning to the timestep t using:

$$D_{1,t} = \sum_{x} \sum_{y} D_{1,t,c_{xy}} \tag{5.13}$$

— **Sustainability score:** We measure sustainability as follows:

$$Score_{n,m} = \frac{1}{V} \times (\frac{S_{n,m} - S_{min,n,m}}{S_{max,n,m} - S_{min,n,m}} + 1) \tag{5.14}$$

whereby $S_{n,m} = \sum_{x=1}^{Z} S_{n,m,c_{xy}}$; $n = 1$ and $m = 7200$; Z is the total number of abstract services; V is the total number of local and global constraint violations. $S_{min,n,m}$ and $S_{max,n,m}$ are the lower and upper value among all approaches. $Score_{n,m} \in [0, 1]$ and a higher value means that the adaptations would generate more benefits in general when mitigating each constraint violation.

— **Running time:** This is the required running time for the reasoning process to produce a composition plan.

Whenever overall results are reported, we use the pairwise version of the Kruskal Wallis test ($p < .05$) [168] and η^2 value [169] to measure statistical significance and effect size, respectively.

5.7.4 RQ 5.1: Performance of DATESSO

Figure 5.3 and 5.4 respectively illustrate the global utilisation and latency for all approaches and timesteps. As can be seen, the comparison between DATESSO and any other three are statistically significant with large effect size. In particular, when comparing with RBC, DATESSO achieves much better utilization and latency overall, while at the same time, it has smaller variance than RBC.

To better understand which of our contributions in DATESSO enable such improvement, we firstly compare it with TLHCA and DOA. As shown in the boxplots, we see that

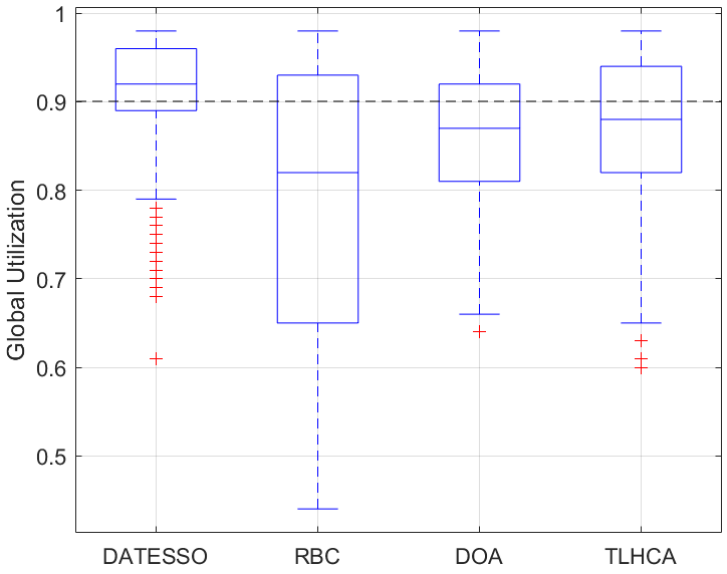


Figure 5.3: Global utilization yield by all approaches over 7200 timesteps (Comparisons between `DATESSO` and others are statistically significant ($p < .05$) and with large effect size)

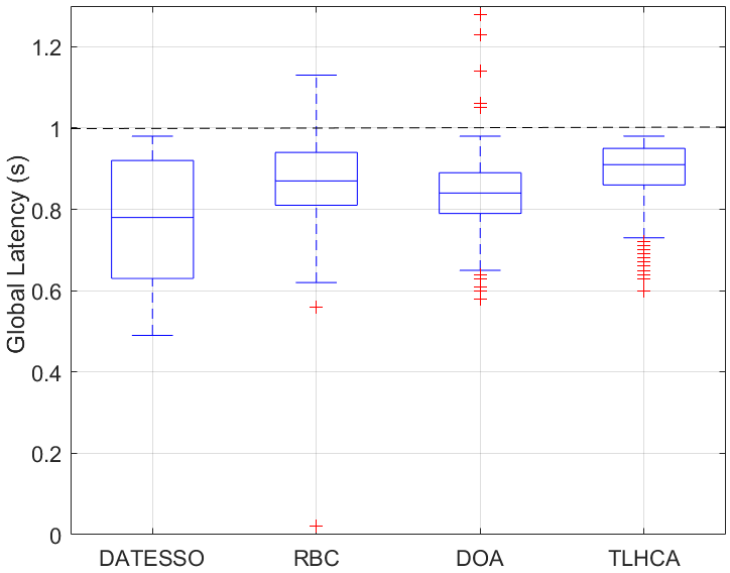


Figure 5.4: Global latency yield by all approaches over 7200 timesteps (Comparisons between `DATESSO` and others are statistically significant ($p < .05$) and with large effect size)

DATESSO achieves much better utilization and smaller variance. For latency, DATESSO is slightly more deviated, but provides overall better result. This has proved that, in general, the formalization of two levels constraints with different strictness can help to improve self-adaptation performance. Next, we compare DATESSO with DOA, for which we see that again, DATESSO achieves generally better and more stable results on utilisation and latency. This evidences that the predicted debt model can provide more benefit than simply having a predicted model based solely on utilization and latency.

Remarkably, DATESSO achieves full satisfaction for the global constraint on latency and satisfy that of utilization for majority of the cases, which are generally superior to the other three. Therefore, for **RQ 5.1**, we conclude that:

Answering RQ 5.1: DATESSO is more effective than the state-of-the-arts in terms of the utilization and latency, with better satisfactions. Both the design of formalizing global constraints as the soft ones and the temporal debt-aware utility model have contributed to the improvement.

5.7.5 RQ 5.2: Sustainability of DATESSO

We now assess the sustainability of adaptation achieved by using the accumulated debt and sustainability score. Figure 5.5 shows the accumulated debt, in which we see that all approaches have accumulated debt over time. However, clearly, DATESSO results in significantly less debt than the other three approaches, suggesting that DATESSO favours component services that is less likely to contribute to global constraint violation in the long term.

Table 5.2 shows the sustainability scores for all approaches. As can been seen, despite that DATESSOand DOA have similar total number of constant violations, DATESSO has achieved the best $Score_{n,m}$ value among others. This implies that the adaptations in DATESSO would create the greatest benefit in mitigating per violation. All the above con-

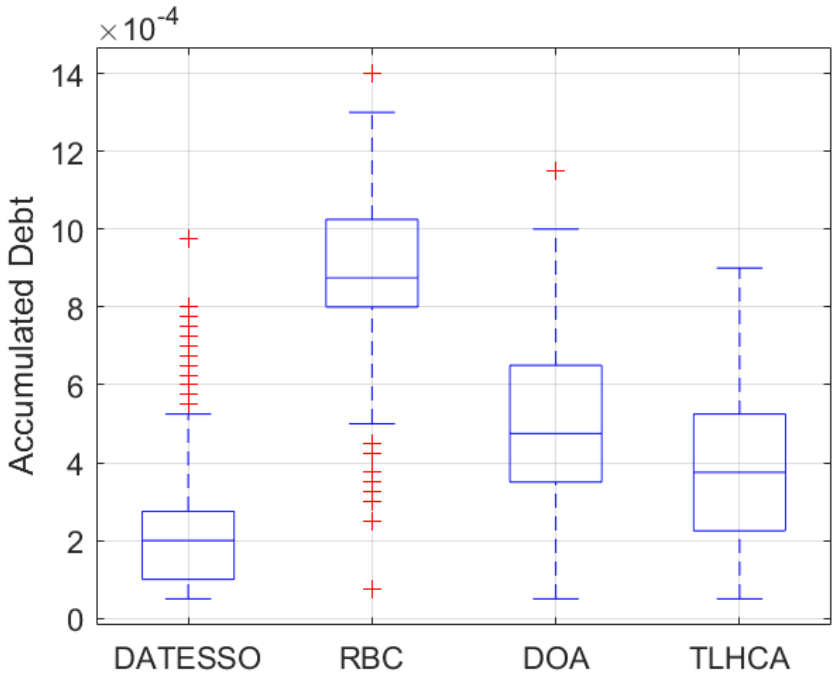


Figure 5.5: Debt yield by all approaches over 7200 timesteps (Comparisons between DATESSO and others are statistically significant ($p < .05$) and with large effect size)

Table 5.2: Sustainability scores

Approach	$\sum_{x=1}^{Z} S_{n,m,c_{xy}}$	V	$Score_{n,m}$
DATESSO	417.10	113	.0177
RBC	−3146.66	187	.0053
DOA	−910.61	102	.0160
TLHCA	−1478.67	133	.0110

clude that:

Answering RQ 5.2: DATESSO is more sustainable than the other three, as it has less accumulated debt and with the highest sustainability score. This means that DATESSO favors more reliable component services in the long term, and that it offers greater benefit when dealing with each violation overall.

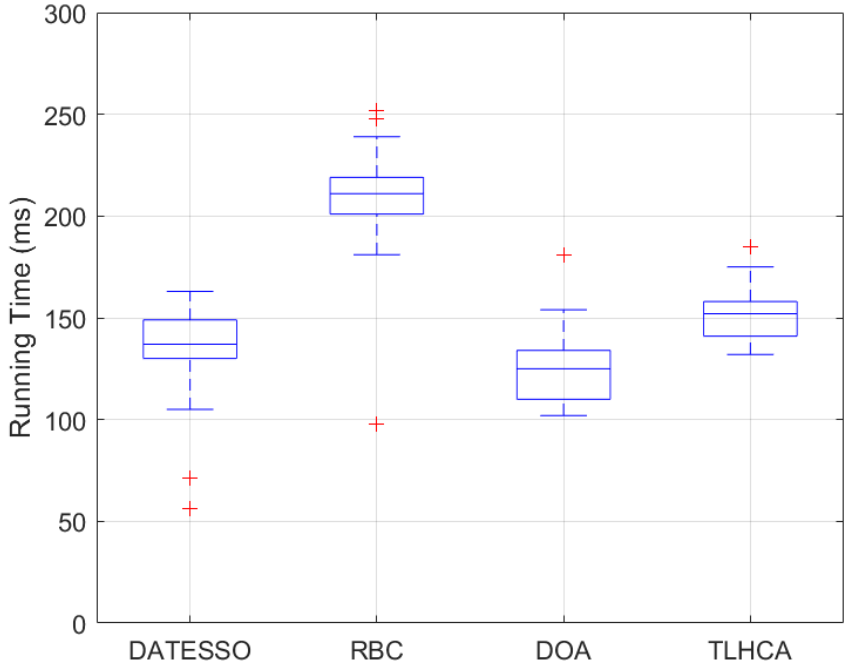


Figure 5.6: Running time on all approaches (Comparisons between DATESSO and others are statistically significant ($p < .05$) and with large effect size, except for DOA)

5.7.6 RQ 5.3: Running Time of DATESSO

Figure 5.6 illustrates the running time for all approaches. We can clearly see that RBC is the slowest due to the region based algorithm. TLHCA is the 2nd slowest because of the frequent need of replacing all component services. Since DATESSO and DOA differ only on whether having the debt calculation, they have similar running overhead ($p > .05$) but are significantly faster than the others. This is because only the problematic abstract services, along with those component services that satisfy all considered timesteps, are involved in the actual search, which reduces the search space. However, as we have shown, DATESSO offers much better performance and sustainability than DOA. In summary, we have:

Answering RQ 5.3: DATESSO and DOA both have similar running time, but they are faster than the other two.

5.8 Threats to Validity

Threats to construct validity can be related to the metric and evaluation methods used. To mitigate such, we use a broad range of metrics for evaluating different aspects of DATESSO, including utilization, latency and sustainability etc. To examine the effectiveness of each contribution, we have compared DATESSO with specifically designed approaches, i.e., TLHCA and DOA, in addition to a direct implementation of existing work (RBC). Further, we plot all the data points in a trace, and applied statistical test and effect size interpretation when it is difficult to show all the data points.

Threats to internal validity can be mainly related to the value of the parameters for DATESSO. Particularly, the setup has been designed in a way that it produces good trade-off between the quality and overhead. They have been shown to be reasonable following preliminary runs in our experiments. The future timestep m is also specifically tailored and the used value tends to be sufficient. However, it is worth noting that the actual future timesteps to use is updated dynamically depending on whether there is a feasible component service that satisfies all considered timesteps.

Threats to external validity can be associated with the environment and the dataset that are used in the experiment. To improve generalization, we use e-commerce system, whose data is randomly sampled from the real-world WS-Dream dataset [19], along with the FIFA98 workload trace [20].

5.9 Summary

In this chapter, we propose a debt-aware two level constraint reasoning approach, dubbed DATESSO, for self-adapting service composition. DATESSO formalizes the global constraints as the soft ones while leaving only the local ones as hard constraints. Such formalization is then used to built a temporal debt-aware utility model, supported by time-series prediction. The utility model, together with the different strictness of the two level con-

straints, enables us to design a simple yet efficient and effective reasoning algorithm in DATESSO. Experimental results demonstrate that DATESSO is more effective that state-of-the-art in terms of utilisation, latency and running time, while being about to make each self-adaptation more sustainable.

CHAPTER 6

CONCLUSIONS, REFLECTIONS, AND FUTURE DIRECTIONS

In this chapter, we revisit the research questions that were discussed in Chapter 1. We systematically review how our contributions have been addressed these research questions. Further, we review our contributions using different qualitative aspects, e.g., practical deployment and computational overhead and SaaS dynamics.

6.1 How the research questions have been addressed

6.1.1 Research Question 1

RQ1 : *Reviewing state-of-the-art service composition approach and identifying the research gaps in the area of economic-driven service composition in the SaaS cloud* – **What is the state-of-the-art service composition approaches with a particular interest in economic aspects? and what are the pending research challenges in an economic-driven service composition in the SaaS cloud?**

We conduct a systematic literature review that provides a deeper understanding of the state-of-the-art service composition approach. We found that service composition has been studied and mentioned under different terms such as (i) QoS-aware; (ii) constraints-aware; (iii) SLA-aware; (iv) context-aware; (v) uncertainty-aware; (vi) economic-driven. In addition, we identified the service quality factors involved in these approaches and its

implementation techniques. Hence, we consolidated these efforts and provided a classification framework that categories these approaches based on the requirements imposed by end-users, service provider or the operating environment. From the SLR results, we examined that the current service composition approaches have the less adoption of an economic aspect and quite less number of research works studied in the SaaS cloud.

However, based on the SLR findings, we identified the pending challenges for exploring cutting-edge research in the multi-tenant SaaS cloud as follows (i) The lack of support to process diverse requirements(e.g., different functional and QoS requirements for the same service) submitted by multiple tenants in the SaaS cloud (ii) The lack of an economic-driven decision support at run-time for service recomposition under dynamic changing workload in the SaaS cloud; (iii) The lack of considering the different strictness of the local and global constraints for an economic-driven based long-term service selection.

6.1.2 Research Question 2

RQ2 : *Realising diverse functional and QoS requirements from the tenants in the SaaS cloud* – **how can we leverage an evolutionary algorithm to support dynamic optimisation of multi-tenant service compositions in the SaaS Cloud?**

In Chapter 3, we developed a service composition engine accelerated by an evolutionary algorithm. We first modelled the multi-tenant service composition approach as a multi-objective optimisation problem with a novel encoding representation and fitness function [23]. We employed the Multi-objective Evolutionary Algorithm named MOEA/D-STM that uses our encoding representation for optimising the service composition. In particular, we design the chromosome in such a way that it dynamically splits itself into two parts (e.g., sub-chromosome) during the optimisation process. The current status of MOEA/D-STM is to support an independent execution of these chromosomes in the optimisation process and then optimise the service composition plans for two different categories of tenants; they requesting the same application with varied functional and QoS requirements. Further, we develop a service composition engine accelerated by MOEA/D-

STM algorithm. However, the service composition engine is extensible in terms of flexible services (e.g., varied number of services in the application workflow) for optimising new service composition plan.

6.1.3 Research Question 3

RQ3 : *Realising an economic-driven service recomposition decisions in the changing workload from the tenants in the SaaS cloud* − **how can we leverage the technical debt metaphor to support an economic-driven decisions for dynamic service recomposition in the SaaS Cloud? and how can the use of predictive analytics of workload improves the decision making and evaluates the service debt?**

We extended the scope of technical debt metaphor in the context of service composition for making an economic-driven decisions for service recomposition in the dynamic SaaS environment [6, 24]. We presented the service debt definition, which is a transformation of the technical debt concept. Further, we posited that the technical debt could be the consequences of taking imperfect or poorly justified runtime decisions for dynamic service recomposition. These run-time decisions have produced two different types of debts in service composition, namely good debt and bad debt. The former implies that a good debt will be paid off by the time k in the future. Specifically, this can be reflected by the fact that, by time k, the debt has been made smaller or the overall service utility has been improved. The latter opposed to the good debt, bad debt is the service debt that will not be paid off by the time k in the future. That is, by time k, there is no sign of improvement on either the service debt and the overall service utility.

We presented a service debt model that used the time series forecasting method for quantifying the future service debt in the composite application execution. Further, we implemented a debt-aware trigger for recomposing the service; in which the service debt model and time series prediction serve as an integrated driver. This economic-driven approach supports controllable trade-off between short-term advantages and long-term benefits. Overall, we combined all components and developed a holistic debt-aware framework

for recomposing service in the SaaS cloud, namely DebtCom. The proposed DebtCom framework is tested on different timestamps, and the DebtCom is extensible in terms of flexible services in the composition and adjustable timestamps. Further, to the best of our knowledge, the current state of DebtCom framework is based on the integrated guideline in the algorithm; further, which is guided by the service debt and parameters values.

6.1.4 Research Question 4

RQ4 : *Realising the strictness of soft and hard constraints on the different level of service composition* – **how can a debt-aware two-levels constraints reasoning of a service selection create the long-term values in the self-adaptive service composition in the SaaS cloud?**

In Chapter 5, we used a time-sensitive application case study that motivates the need for considering different strictness at the local and global level of service composition. From Chapter 4 results, we realised that the debt is a time-sensitive moving target that encouraged us to reformulate these constraints. As a result, we presented two-level constraints with different strictness, namely local constraints as hard and global constraints as soft [22]. In this scenario, the debt may be acceptably incurred at global constraints if all local constraints are satisfied in the service composition.

Further, we developed a DATESSO framework that comprises of three components at different levels in the service composition. (i) The formalizer component monitored the service composition environment. It triggered the service adaptation when any constraints violation is detected and reported all faulty component services (infeasible component services) to the next stage component (modelling component). (ii) In the modelling component, we developed a service debt model that uses the two-level constraints information and the time series forecasting method for predicting workload on the component service in the composition. (iii) Reasoner is the core component in the DATESSO framework that uses all the information provided by formalizer and modelling components. Reasoner component exploited debt-aware two-level constraints reasoning for searching

a long-term based feasible candidate service that maximises the constraints satisfaction and the service utility over n timesteps in the service composition.

6.2 Reflections on the Research

This section aims to reflect on the presented approach and its evaluation by mean of design concern of simulation environment, computational overhead and scalability.

6.2.1 Simulation Environment

In this thesis, we made our best efforts to implement an e-commerce system as an illustrative example of a simulation environment; in which our service composition approach could be examined. However, we appreciate that our evaluation is in a controlled environment instead of a real SaaS cloud environment, and it enables us to conduct repeatable free of cost experiments. We have used sequential connector in the composition for modelling the simulation environment. Therefore, further research is needed for evaluating our approach under different connectors (e.g., parallel) in the real setting of SaaS cloud environment.

Apart from that we developed a more realistic experiment using real-world WSDream dataset [19] and real workload trace (FIFA 1998 world cup trace) [20]. In particular, to emulate the SaaS cloud environment, we deployed all web services in the Docker containers they were created with the different capacities of resources (e.g., CPU and RAM). Moreover, the collected 6 hours workload trace represents a controllable amount of service demand in the simulation environment, but in the real SaaS cloud environment, such workload may have a diverse pattern over time. Although, we attempted to deliver our best for developing the simulation environment. But we appreciate that further research will be required to evaluate the effectiveness of our approach in the real setting of SaaS cloud environment.

6.2.2 Computational Overhead

The experiments may have carried some hidden computational overhead that may be generated from the docker container; in which all web services are running. Moreover, in the context of service composition, the computational overhead of our proposed approach comes from the following sources.

- Runtime estimation of service debt.
- Searching and optimising the set of web services from the service repository.
- Monitoring service composition environment.

The service debt estimation process consumes an extra computational time when compared to another state-of-the-art approach. The approach presented in Chapter 4, the computational overhead is directly linked to the predicted debt-aware decisions for service recomposition. Further, we have used different timesteps which affects the computation time for estimating service recomposition decisions. After triggering the service recomposition decision, the optimisation engine consumes the computational time for searching the entire service repository and optimising the new service composition plan by selecting suitable candidate web services from the service repository. However, the computational time mainly depends on the size of the service repository (e.g., the number of candidate services in the repository), which is presented in Chapter 3. Further, our approach presented in the Chapter 5, an extra computational time is directly linked to the monitoring of the composite service execution environment and estimating the service debt incurred by constraints violations. In nutshell, this process requires computational time for calculating the service debt and the values of the defined constraints over the service composition.

In general, the critical source of consuming computational overhead is the size of search space exposed by the service repository. We appreciate the further extension of our approach but it requires the rigorous analysis of computational overhead to reduce

the side effects (e.g., latency for the debt-aware decision) and to maximise the service utility and performance.

6.2.3 Dealing with SaaS Dynamics

In the context of self-adaptive software systems, dynamics denotes the changing conditions of the operating environment in which the software is running [170]. Our simulation environment emulates the SaaS cloud environment; in which the main dynamics is related to the uncertainty of tenants; they generate unpredictable request workload for consuming web services in the SaaS cloud. The selection of an appropriate requests workload is always a non-trivial decision for conducting a fair evaluation of an experiment [171].

We made our best efforts to use a more realistic workload in the experiment. Therefore, we decided to use the real-world workload trace (FIFA 1998 world cup trace) [20]. Our proposed framework is flexible in terms of using other requests workload, and we appreciate the further testing of our approach under diverse workload in the real setting of SaaS cloud environment.

6.3 Future Directions

In this section, we discuss the potential future directions derived from the presented research work in this thesis and other state-of-art research direction in the cloud computing environment from the technical debt perspective.

6.3.1 Exploring technical debt-aware supports for service composition in the SaaS cloud

In this thesis, we demonstrated the effectiveness of the technical debt-aware approach for creating long-term values in the service composition in SaaS cloud. In particular, we adopted the time series forecasting method for estimating the predictive service debt, which geared an economic-drive decision for generating values in the composition. Since

the underlying decision technique is equipped on the proactive method, and the debt estimation may have a hidden impact on optimising the present values in the system [147]. In this context, future research is needed to learn the past behaviour of debt for making better economic-driven decision to optimise the present value in the composition with different underlying techniques such as reinforcement learning or classical machine learning algorithms. Further, we studied run-time perspective of technical debt for the service quality attributes such as throughput, response time and service utility. However, more QoS attributes and uncertainties are associated with the service composition in SaaS cloud. Consequently, another future research direction is to explore the new methods for evaluating the technical debt on these attributes for creating an economic driven service composition in the SaaS cloud.

6.3.2 Dealing uncertainties in the SaaS cloud environment

In this thesis, we demonstrated the feasibility of our technical debt-aware service composition approach under uncertain and the dynamic SaaS cloud environment. Our technical debt-aware approach handled only service scalability and workload uncertainties; resulted from the unpredictable and dynamic changing workload on the composite service. However, the SaaS cloud environment tends to lead more uncertainties such as delayed latency, service availability, fault-tolerance, performance and failure-prone environment [12, 172]. And, most of the existing service composition approaches have dealt with design-time QoS uncertainty [13, 77, 83, 101]. We appreciate that further research will require to consider these uncertainties for developing new economic-driven service composition in the SaaS cloud environment.

6.3.3 A technical debt perspective for the selection and optimisation of cloud services/resources

This thesis shows the constructive application of technical debt metaphor in the area of service composition in the SaaS cloud. Gomez et al. [173, 174] explored the concept of technical debt for elasticity management in the cloud computing environment. In this thesis, we argued that the technical debt could be related to any situation contributing to sub-optimal execution environment such as under/over utilisation of services. This argument is valid for cloud resources and service instances. For example, the execution of Virtual Machines (VM) in the cloud data centre may encounter the problem of sub-optimal utilisation of allocated VM resources, and it requires run-time decision for VM migration. This particular situation could be the potential source of incurring technical debt in the cloud data centre. Based on this thesis's research finding, we believe that the technical debt metaphor could be a potential technique for providing economic-driven decision-making for migrating and optimising the VM resources in the cloud data centre. We appreciate the future direction in this area, specifically investigating the potential sources of technical debt in the cloud data centre and developing Artificial Intelligence (AI) inspired new methods for handling and managing technical debt towards optimising cloud data centre.

Another potential future direction would be the technical debt-aware cloud instance selection in the multi-cloud environment. In the cloud data centre, it is a rear condition when a running cloud service instance uses its full capacity, and its consequences could be the sub-optimal utilisation of instance resources. In this context, how to take critical decisions on whether to use the current cloud service instance or select the new cloud service instance that maximises the service revenue over time or contributes optimal utilisation of cloud resources. We appreciate that the researchers can take advantage of technical debt metaphor and combine it with AI and Machine Learning (ML) algorithms to develop an economic-driven enabled predictive decision-making framework for cloud

instance selection in the multi-cloud environment.

6.4 Conclusion Remarks

This thesis makes a novel contribution to the field of service composition by presenting an economic-drive service composition approach based on the principle of technical debt framework. The results of our experimental evaluation present many useful insight on the effectiveness of our debt aware-approach to provide an economic-driven long-term decision for the specified problems in this thesis. In particular, the result of our debt-aware approach indicates that (i) It prepares long-term based value creation decisions in the service composition; (ii) It mitigates the imperfect run-time decisions that degrade the service utilisation; (iii) It supports long-term based economic-driven service selection decision for improving service utilisation.

We hope that adopting a technical debt framework will motivate further research in this direction. Our results will inspire future research in an economic-driven service composition for the SaaS cloud environment. Moreover, we also provided an extensive discussion on the potential future directions such as (i) revisiting an economic-driven service composition approach in the SaaS cloud and investigating the impact of hidden debt accumulated in the past. The research is needed to know the past behaviour of incurred debt using ML algorithms to optimise the present composition value; (ii) Usually, the running resources (e.g., VM or service instances) in the cloud data centre exhibit the sub-optimal utilisation of their capacities. Its consequences would be the accumulation of debt in the cloud data centre. However, the research is needed to combine the technical debt metaphor and AI & ML algorithms for developing an economic-driven predictive method. Such a method would make an informed decision on whether to use the current cloud resources or select new cloud resources for creating long-term benefits (e.g., maximum resource utilisation over time) in the cloud data centre.

BIBLIOGRAPHY

[1] Gartner Inc. Gartner forecasts worldwide public cloud revenue to grow 17% in 2020. November, 2019 [online]. Available from: https://www.gartner.com/en/newsroom/press-releases/2019-11-13-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2020[Last accessed 26th January 2020].

[2] Liangzhao Zeng, Boualem Benatallah, Anne HH Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Transactions on software engineering*, 30(5):311–327, 2004.

[3] Yanchun Wang, Qiang He, and Yun Yang. Qos-aware service recommendation for multi-tenant saas on the cloud. In *2015 IEEE International Conference on Services Computing*, pages 178–185. IEEE, 2015.

[4] Qiang He, Jun Han, Yun Yang, John Grundy, and Hai Jin. Qos-driven service selection for multi-tenant saas. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 566–573. IEEE, 2012.

[5] Salesforce. Explore the CRM software features that can help you grow sales faster [online]. Available from: https://www.salesforce.com/products/sales-cloud/features/[Last accessed 25th March 2019].

[6] Satish Kumar, Rami Bahsoon, Tao Chen, and Rajkumar Buyya. Identifying and estimating technical debt for service composition in saas cloud. In *2019 IEEE International Conference on Web Services (ICWS)*, pages 121–125. IEEE, 2019.

[7] Ward Cunningham. The wycash portfolio management system. *ACM SIGPLAN OOPS Messenger*, 4(2):29–30, 1992.

[8] Edith Tom, AybüKe Aurum, and Richard Vidgen. An exploration of technical debt. *Journal of Systems and Software*, 86(6):1498–1516, 2013.

[9] Zengyang Li, Paris Avgeriou, and Peng Liang. A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101:193–220, 2015.

[10] Esra Alzaghoul and Rami Bahsoon. Cloudmtd: Using real options to manage technical debt in cloud-based service selection. In *2013 4th International Workshop on Managing Technical Debt (MTD)*, pages 55–62. IEEE, 2013.

[11] Yanchun Wang, Qiang He, Xuyun Zhang, Dayong Ye, and Yun Yang. Efficient qos-aware service recommendation for multi-tenant service-based systems in cloud. *IEEE Transactions on Services Computing*, 2017.

[12] Yaser Yadekar, Essam Shehab, and Jörn Mehnen. Taxonomy and uncertainties of cloud manufacturing. *International Journal of Agile Systems and Management*, 9(1):48–66, 2016.

[13] Ahmed Mostafa and Minjie Zhang. Multi-objective service composition in uncertain environments. *IEEE Transactions on Services Computing*, 2015.

[14] Kwei-Jay Lin, Jing Zhang, Yanlong Zhai, and Bin Xu. The design and implementation of service process reconfiguration with end-to-end qos constraints in soa. *Service Oriented Computing and Applications*, 4(3):157–168, 2010.

[15] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. A framework for qos-aware binding and re-binding of composite web services. *Journal of Systems and Software*, 81(10):1754–1769, 2008.

[16] Ying Li, Yuanlei Lu, Yuyu Yin, Shuiguang Deng, and Jianwei Yin. Towards qos-based dynamic reconfiguration of soa-based applications. In *2010 IEEE Asia-Pacific Services Computing Conference*, pages 107–114. IEEE, 2010.

[17] San-Yih Hwang, Ee-Peng Lim, Chien-Hsiang Lee, and Cheng-Hung Chen. Dynamic web service selection for reliable web service composition. *IEEE Transactions on services computing*, 1(2):104–116, 2008.

[18] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[19] Zibin Zheng, Yilei Zhang, and Michael R Lyu. Investigating qos of real-world web services. *IEEE transactions on services computing*, 7(1):32–39, 2012.

[20] Martin Arlitt and Tai Jin. A workload characterization study of the 1998 world cup web site. *IEEE network*, 14(3):30–37, 2000.

[21] Jurgen A Doornik and Marius Ooms. A package for estimating, forecasting and simulating arfima models: Arfima package 1.0 for ox, 1999 [online]. Available from: http://fmwww.bc.edu/ec-p/software/ox/0x.arfima.v2.1.pdf[Last accessed 12th January 2019].

[22] Satish Kumar, Tao Chen, Rami Bahsoon, and Rajkumar Buyya. Datesso: Self-adapting servicecomposition with debt-aware two levels constraints reasoning. In *ACM/IEEE 15th International Symposium on Software Engineering for Adaptive- and Self-Managing Systems*. IEEE, 2020.

[23] Satish Kumar, Rami Bahsoon, Tao Chen, Ke Li, and Rajkumar Buyya. Multi-tenant cloud service composition using evolutionary optimization. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 972–979. IEEE, 2018.

[24] Satish Kumar, Tao Chen, Rami Bahsoon, and Rajkumar Buyya. Debtcom: Technical debt-aware service recomposition in saas cloud. *IEEE transactions on Service Computing (under review)*, 2020.

[25] Satish Kumar, Rami Bahsoon, Tao Chen, and Rajkumar Buyya. A systematic review and taxonomy on service composition based on service quality factors (in preparation for submission). 2020.

[26] Ke Li, Qingfu Zhang, Sam Kwong, Miqing Li, and Ran Wang. Stable matching-based selection in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 18(6):909–923, 2013.

[27] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[28] Mohammad Alrifai, Dimitrios Skoutas, and Thomas Risse. Selecting skyline services for qos-based web service composition. In *Proceedings of the 19th international conference on World wide web*, pages 11–20, 2010.

[29] Alexandre Sawczuk da Silva, Hui Ma, and Mengjie Zhang. A graph-based particle swarm optimisation approach to qos-aware web service composition and selection. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 3127–3134. IEEE, 2014.

[30] Touraj Laleh, Joey Paquet, Serguei Mokhov, and Yuhong Yan. Constraint adaptation in web service composition. In *2017 IEEE International Conference on Services Computing (SCC)*, pages 156–163. IEEE, 2017.

[31] Ahmed Moustafa and Minjie Zhang. Multi-objective service composition using reinforcement learning. In *International Conference on Service-Oriented Computing*, pages 298–312. Springer, 2013.

[32] X Liang, K Qin, K Tang, and K Tan. Qos-aware web service composition with internal complementarity. *IEEE Transactions on Services Computing*, 12(2):1–14, 2019.

[33] Kaiqi Xiong and Harry Perros. Sla-based service composition in enterprise computing. In *2008 16th Interntional Workshop on Quality of Service*, pages 30–39. IEEE, 2008.

[34] Dandan Wang, Hao Ding, Yang Yang, Zhenqiang Mi, Li Liu, and Zenggang Xiong. Qos and sla aware web service composition in cloud environment. *TIIS*, 10(12):5231–5248, 2016.

[35] PengWei Wang, ZhiJun Ding, ChangJun Jiang, and MengChu Zhou. Constraint-aware approach to web service composition. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(6):770–784, 2013.

[36] Donghui Lin, Chunqi Shi, and Toru Ishida. Dynamic service selection based on context-aware qos. In *2012 IEEE Ninth International Conference on Services Computing*, pages 641–648. IEEE, 2012.

[37] Aiqiang Gao, Dongqing Yang, Shiwei Tang, and Ming Zhang. Web service composition using integer programming-based models. In *IEEE International Conference on e-Business Engineering (ICEBE'05)*, pages 603–606. IEEE, 2005.

[38] Ying Chen, Jiwei Huang, Chuang Lin, and Xuemin Shen. Multi-objective service composition with qos dependencies. *IEEE Transactions on Cloud Computing*, 2016.

[39] Hamed Rezaie, Naser NematBaksh, and Farhad Mardukhi. A multi-objective particle swarm optimization for web service composition. In *International Conference on Networked Digital Technologies*, pages 112–122. Springer, 2010.

[40] Wei Zhang, Carl K Chang, Taiming Feng, and Hsin-yi Jiang. Qos-based dynamic web service composition with ant colony optimization. In *2010 IEEE 34th Annual Computer Software and Applications Conference*, pages 493–502. IEEE, 2010.

[41] Yanlong Zhai, Jing Zhang, and Kwei-Jay Lin. Soa middleware support for service process reconfiguration with end-to-end qos constraints. In *2009 IEEE International Conference on Web Services*, pages 815–822. IEEE, 2009.

[42] Rafael Aschoff and Andrea Zisman. Qos-driven proactive adaptation of service composition. In *International Conference on Service-Oriented Computing*, pages 421–435. Springer, 2011.

[43] Karl Gottschalk, Stephen Graham, Heather Kreger, and James Snell. Introduction to web services architecture. *IBM systems Journal*, 41(2):170–177, 2002.

[44] Biplav Srivastava and Jana Koehler. Web service composition-current solutions and open problems. In *ICAPS 2003 workshop on Planning for Web Services*, volume 35, pages 28–35, 2003.

[45] Dieter Fensel and Christoph Bussler. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[46] Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on software engineering*, 33(6):369–384, 2007.

[47] Angus FM Huang, Ci-Wei Lan, and Stephen JH Yang. An optimal qos-based web service selection scheme. *Information Sciences*, 179(19):3309–3322, 2009.

[48] Rainer Berbner, Michael Spahn, Nicolas Repp, Oliver Heckmann, and Ralf Steinmetz. Heuristics for qos-aware web service composition. In *2006 IEEE International Conference on Web Services (ICWS'06)*, pages 72–82. IEEE, 2006.

[49] Peter Mell and Tim Grance. The NIST definition of cloud computing, 2011 [online]. Available from: http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf[Last accessed 12th December 2019].

[50] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: issues and challenges. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 27–33. Ieee, 2010.

[51] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. Technical debt: From metaphor to theory and practice. *IEEE software*, 29(6):18–21, 2012.

[52] Nicolli SR Alves, Thiago S Mendes, Manoel G de Mendonça, Rodrigo O Spínola, Forrest Shull, and Carolyn Seaman. Identification and management of technical debt: A systematic mapping study. *Information and Software Technology*, 70:100–121, 2016.

[53] Aniket Potdar and Emad Shihab. An exploratory study on self-admitted technical debt. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 91–100. IEEE, 2014.

[54] Paris Avgeriou, Philippe Kruchten, Ipek Ozkaya, and Carolyn Seaman. Managing technical debt in software engineering (dagstuhl seminar 16162). In *Dagstuhl Reports*, volume 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[55] Yuepu Guo, Carolyn Seaman, Rebeka Gomes, Antonio Cavalcanti, Graziela Tonin, Fabio QB Da Silva, Andre LM Santos, and Clauirton Siebra. Tracking technical debt—an exploratory case study. In *2011 27th IEEE international conference on software maintenance (ICSM)*, pages 528–531. IEEE, 2011.

[56] Radu Marinescu. Assessing technical debt by identifying design flaws in software systems. *IBM Journal of Research and Development*, 56(5):9–1, 2012.

[57] Steve McConnell. Managing technical debt. *Construx Software Builders, Inc*, pages 1–14, 2008, [online], Available from: http://2013.icse-conferences.org/documents/publicity/MTD-WS-McConnell-slides.pdf[Last accessed 15th July 2017].

[58] Barbara Kitchenham, O Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, 51(1):7–15, 2009.

[59] Staffs Keele et al. Guidelines for performing systematic literature reviews in software engineering. Technical report, Technical report, Ver. 2.3 EBSE Technical Report. EBSE, 2007.

[60] He Zhang, Muhammad Ali Babar, and Paolo Tell. Identifying relevant studies in software engineering. *Information and Software Technology*, 53(6):625–637, 2011.

[61] Pearl Brereton, Barbara A Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software*, 80(4):571–583, 2007.

[62] Hongyu Pei Breivold, Ivica Crnkovic, and Magnus Larsson. A systematic review of software architecture evolution research. *Information and Software Technology*, 54(1):16–40, 2012.

[63] Markus Keidl and Alfons Kemper. Towards context-aware adaptable web services. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 55–65, 2004.

[64] Tao Yu and Kwei-Jay Lin. Service selection algorithms for web services with end-to-end qos constraints. *Information systems and e-business management*, 3(2):103–126, 2005.

[65] Zakaria Maamar, Soraya Kouadri Mostefaoui, and Hamdi Yahyaoui. Toward an agent-based and context-oriented approach for web services composition. *IEEE transactions on knowledge and data engineering*, 17(5):686–697, 2005.

[66] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Transactions on the Web (TWEB)*, 1(1):6–es, 2007.

[67] Hiroshi Wada, Paskorn Champrasert, Junichi Suzuki, and Katsuya Oba. Multiobjective optimization of sla-aware service composition. In *2008 IEEE Congress on Services-Part I*, pages 368–375. IEEE, 2008.

[68] Riadh Ben Halima, Khalil Drira, and Mohamed Jmaiel. A qos-oriented reconfigurable middleware for self-healing web services. In *2008 IEEE International Conference on Web Services*, pages 104–111. IEEE, 2008.

[69] Mohammad Alrifai and Thomas Risse. Combining global optimization with local selection for efficient qos-aware service composition. In *Proceedings of the 18th international conference on World wide web*, pages 881–890, 2009.

[70] Philipp Leitner, Branimir Wetzstein, Florian Rosenberg, Anton Michlmayr, Schahram Dustdar, and Frank Leymann. Runtime prediction of service level agreement violations for composite services. In *Service-oriented computing. ICSOC/ServiceWave 2009 workshops*, pages 176–186. Springer, 2009.

[71] Georgia M Kapitsaki, Dimitrios A Kateros, George N Prezerakos, and Iakovos S Venieris. Model-driven development of composite context-aware web applications. *Information and Software technology*, 51(8):1244–1260, 2009.

[72] Kwei-Jay Lin, Jing Zhang, and Yanlong Zhai. An efficient approach for service process reconfiguration in soa with end-to-end qos constraints. In *2009 IEEE Conference on Commerce and Enterprise Computing*, pages 146–153. IEEE, 2009.

[73] Philipp Leitner, Anton Michlmayr, Florian Rosenberg, and Schahram Dustdar. Monitoring, prediction and prevention of sla violations in composite services. In *2010 IEEE International Conference on Web Services*, pages 369–376. IEEE, 2010.

[74] Maolin Tang and Lifeng Ai. A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[75] Chia-Feng Lin, Ruey-Kai Sheu, Yue-Shan Chang, and Shyan-Ming Yuan. A relaxable service selection algorithm for qos-based web service composition. *Information and Software Technology*, 53(12):1370–1381, 2011.

[76] Ching-Seh Wu and Ibrahim Khoury. Tree-based search algorithm for web service composition in saas. In *2012 Ninth International Conference on Information Technology-New Generations*, pages 132–138. IEEE, 2012.

[77] Karim Benouaret, Djamal Benslimane, and Allel Hadjali. Selecting skyline web services from uncertain qos. In *2012 IEEE Ninth International Conference on Services Computing*, pages 523–530. IEEE, 2012.

[78] Zhen Ye, Athman Bouguettaya, and Xiaofang Zhou. Qos-aware cloud service composition based on economic models. In *International Conference on Service-Oriented Computing*, pages 111–126. Springer, 2012.

[79] Philipp Leitner, Johannes Ferner, Waldemar Hummer, and Schahram Dustdar. Data-driven and automated prediction of service level agreement violations in service compositions. *Distributed and Parallel Databases*, 31(3):447–470, 2013.

[80] Yuzhang Feng, Rajaraman Kanagasabai, et al. Dynamic service composition with service-dependent qos attributes. In *2013 IEEE 20th International Conference on Web Services*, pages 10–17. IEEE, 2013.

[81] Shuiguang Deng, Hongyue Wu, Daning Hu, and J Leon Zhao. Service selection for composition with qos correlations. *IEEE Transactions on Services Computing*, 9(2):291–303, 2014.

[82] Quanwang Wu, Qingsheng Zhu, Xing Jian, and Fuyuki Ishikawa. Broker-based sla-aware composite service provisioning. *Journal of Systems and Software*, 96:194–201, 2014.

[83] Shiting Wen, Chaogang Tang, Qing Li, Dickson KW Chiu, An Liu, and Xianglan Han. Probabilistic top-k dominating services composition with uncertain qos. *Service Oriented Computing and Applications*, 8(1):91–103, 2014.

[84] Esra Alzaghoul and Rami Bahsoon. Evaluating technical debt in cloud-based architectures using real options. In *2014 23rd Australian Software Engineering Conference*, pages 1–10. IEEE, 2014.

[85] ZhiJun Ding, JunJun Liu, YouQing Sun, ChangJun Jiang, and MengChu Zhou. A transaction and qos-aware service selection approach based on genetic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(7):1035–1046, 2015.

[86] PengWei Wang, ZhiJun Ding, ChangJun Jiang, MengChu Zhou, and YuWei Zheng. Automatic web service composition based on uncertainty execution effects. *IEEE Transactions on Services Computing*, 9(4):551–565, 2015.

[87] Fuzan Chen, Runliang Dou, Minqiang Li, and Harris Wu. A flexible qos-aware web service composition method by multi-objective optimization in cloud manufacturing. *Computers & Industrial Engineering*, 99:423–431, 2016.

[88] Yueshen Xu, Jianwei Yin, Shuiguang Deng, Neal N Xiong, and Jianbin Huang. Context-aware qos prediction for web service recommendation and selection. *Expert Systems with Applications*, 53:75–86, 2016.

[89] Xiaoning Sun, Jiangchuan Chen, Yunni Xia, Qiang He, Yuandou Wang, Xin Luo, Rongqing Zhang, Wuhong Han, and Quanwang Wu. A fluctuation-aware approach for predictive web service composition. In *2018 IEEE International Conference on Services Computing (SCC)*, pages 121–128. IEEE, 2018.

[90] Sen Niu, Guobing Zou, Yanglan Gan, Yang Xiang, and Bofeng Zhang. Towards the optimality of qos-aware web service composition with uncertainty. *International Journal of Web and Grid Services*, 15(1):1–28, 2019.

[91] Feng Zhang, Qingtian Zeng, Hua Duan, and Cong Liu. Composition context-based web services similarity measure. *IEEE Access*, 7:65195–65206, 2019.

[92] Chandrashekar Jatoth, GR Gangadharan, and Rajkumar Buyya. Computational intelligence based qos-aware web service composition: a systematic literature review. *IEEE Transactions on Services Computing*, 10(3):475–492, 2015.

[93] Anja Strunk. Qos-aware service composition: A survey. In *2010 Eighth IEEE European Conference on Web Services*, pages 67–74. IEEE, 2010.

[94] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *International journal of web and grid services*, 1(1):1–30, 2005.

[95] Umar Galadima Shehu, Gregory Epiphaniou, and Ghazanfar Ali Safdar. A survey of qos-aware web service composition techniques. *International Journal of Computer Applications*, 2014.

[96] Amin Jula, Elankovan Sundararajan, and Zalinda Othman. Cloud computing service composition: A systematic literature review. *Expert systems with applications*, 41(8):3809–3824, 2014.

[97] Hong-Linh Truong and Schahram Dustdar. A survey on context-aware web service systems. *International Journal of Web Information Systems*, 2009.

[98] Jing Li, Dianfu Ma, Xiupei Mei, Hailong Sun, and Zibin Zheng. Adaptive qos-aware service process reconfiguration. In *2011 IEEE International Conference on Services Computing*, pages 282–289. IEEE, 2011.

[99] Ahlem Ben Hassine, Shigeo Matsubara, and Toru Ishida. A constraint-based approach to horizontal web service composition. In *International semantic Web conference*, pages 130–143. Springer, 2006.

[100] Steve Cuddy, Michael Katchabaw, and Hanan Lutfiyya. Context-aware service selection based on dynamic and static service attributes. In *WiMob'2005), IEEE International Conference on Wireless And Mobile Computing, Networking And Communications, 2005.*, volume 4, pages 13–20. IEEE, 2005.

[101] Sen Niu, Guobing Zou, Yanglan Gan, Yang Xiang, and Bofeng Zhang. Towards uncertain qos-aware service composition via multi-objective optimization. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 894–897. IEEE, 2017.

[102] Germán H Alférez and Vicente Pelechano. Facing uncertainty in web service compositions. In *2013 IEEE 20th International Conference on Web Services*, pages 219–226. IEEE, 2013.

[103] Arun Mukhija and Martin Glinz. Runtime adaptation of applications through dynamic recomposition of components. In *International Conference on Architecture of Computing Systems*, pages 124–138. Springer, 2005.

[104] Fouzia Boudries, Samia Sadouki, and Abdelkamel Tari. A bio-inspired algorithm for dynamic reconfiguration with end-to-end constraints in web services composition. *Service Oriented Computing and Applications*, 13(3):251–260, 2019.

[105] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Stefano Iannucci, Francesco Lo Presti, and Raffaela Mirandola. Moses: A framework for qos driven runtime adaptation of service-oriented systems. *IEEE Transactions on Software Engineering*, 38(5):1138–1159, 2011.

[106] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z Sheng. Quality driven web services composition. In *Proceedings of the 12th international conference on World Wide Web*, pages 411–421, 2003.

[107] Florian Rosenberg, Predrag Celikovic, Anton Michlmayr, Philipp Leitner, and Schahram Dustdar. An end-to-end approach for qos-aware service composition. In *2009 IEEE International Enterprise Distributed Object Computing Conference*, pages 151–160. IEEE, 2009.

[108] Mahboobeh Moghaddam and Joseph G Davis. Service selection in web service composition: A comparative review of existing approaches. In *Web Services Foundations*, pages 321–346. Springer, 2014.

[109] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *International Workshop on Semantic Web Services and Web Process Composition*, pages 43–54. Springer, 2004.

[110] Yang Syu, Shang-Pin Ma, Jong-Yih Kuo, and Yong-Yi FanJiang. A survey on automated service composition methods and related techniques. In *2012 IEEE Ninth International Conference on Services Computing*, pages 290–297. IEEE, 2012.

[111] Lijuan Wang, Jun Shen, and Jianming Yong. A survey on bio-inspired algorithms for web service composition. In *Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 569–574. IEEE, 2012.

[112] Marcel Cremene, Mihai Suciu, Denis Pallez, and Dumitru Dumitrescu. Comparative analysis of multi-objective evolutionary algorithms for qos-aware web service composition. *Applied Soft Computing*, 39:124–139, 2016.

[113] Vivek Nallur and Rami Bahsoon. A decentralized self-adaptation mechanism for service-based applications in the cloud. *IEEE Transactions on Software Engineering*, 39(5):591–612, 2012.

[114] Mike P Papazoglou and Dimitrios Georgakopoulos. Introduction: Service-oriented computing. *Communications of the ACM*, 46(10):24–28, 2003.

[115] Amina Bekkouche, Sidi Mohammed Benslimane, Marianne Huchard, Chouki Tibermacine, Fethallah Hadjila, and Mohammed Merzoug. Qos-aware optimal and automated semantic web service composition with user's constraints. *Service Oriented Computing and Applications*, 11(2):183–201, 2017.

[116] Fadl Dahan, Khalil El Hindi, and Ahmed Ghoneim. Enhanced artificial bee colony algorithm for qos-aware web service selection problem. *Computing*, 99(5):507–517, 2017.

[117] Xinchao Zhao, Zichao Wen, and Xingmei Li. Qos-aware web service selection with negative selection algorithm. *Knowledge and Information Systems*, 40(2):349–373, 2014.

[118] Tongguang Zhang. Qos-aware web service selection based on particle swarm optimization. *Journal of Networks*, 9(3):565, 2014.

[119] Dmytro Pukhkaiev, Tetiana Kot, Larysa Globa, and Alexander Schill. A novel sla-aware approach for web service composition. In *Eurocon 2013*, pages 327–334. IEEE, 2013.

[120] Marco Aiello, Elie El Khoury, Alexander Lazovik, and Patrick Ratelband. Optimal qos-aware web service composition. In *2009 IEEE Conference on Commerce and Enterprise Computing*, pages 491–494. IEEE, 2009.

[121] Aurora Ramírez, José Antonio Parejo, José Raúl Romero, Sergio Segura, and Antonio Ruiz-Cortés. Evolutionary composition of qos-aware web services: a many-objective perspective. *Expert Systems with Applications*, 72:357–370, 2017.

[122] A Erdinc Yilmaz and Pinar Karagoz. Improved genetic algorithm based approach for qos aware web service composition. In *2014 IEEE International Conference on Web Services*, pages 463–470. IEEE, 2014.

[123] Chandrashekar Jatoth, GR Gangadharan, Ugo Fiore, and Rajkumar Buyya. Qos-aware big service composition using mapreduce based evolutionary algorithm with guided mutation. *Future Generation Computer Systems*, 86:1008–1018, 2018.

[124] Joc Cing Tay and Djoko Wibowo. An effective chromosome representation for evolving flexible job shop schedules. In *Genetic and Evolutionary Computation Conference*, pages 210–221. Springer, 2004.

[125] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.

[126] Hui Li and Qingfu Zhang. Multiobjective optimization problems with complicated pareto sets, moea/d and nsga-ii. *IEEE transactions on evolutionary computation*, 13(2):284–302, 2008.

[127] Juan J Durillo and Antonio J Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011.

[128] Antonio J Nebro, Juan J Durillo, and Matthieu Vergne. Redesigning the jmetal multi-objective optimization framework. In *Proceedings of the companion publication of the 2015 annual conference on genetic and evolutionary computation*, pages 1093–1100, 2015.

[129] Tao Chen, Miqing Li, and Xin Yao. On the effects of seeding strategies: A case for search-based multi-objective service composition. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1419–1426, 2018.

[130] Juan J Durillo, Antonio J Nebro, and Enrique Alba. The jmetal framework for multi-objective optimization: Design and architecture. In *IEEE congress on evolutionary computation*, pages 1–8. IEEE, 2010.

[131] Tao Chen, Miqing Li, and Xin Yao. How to evaluate solutions in pareto-based search-based software engineering? a critical review and methodological guidance. *arXiv preprint arXiv:2002.09040*, 2020.

[132] Miqing Li and Xin Yao. Quality evaluation of solution sets in multiobjective optimisation: A survey. *ACM Computing Surveys (CSUR)*, 52(2):1–38, 2019.

[133] Shuai Wang, Shaukat Ali, Tao Yue, Yan Li, and Marius Liaaen. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In *Proceedings of the 38th International Conference on Software Engineering*, pages 631–642, 2016.

[134] Mengyuan Wu, Ke Li, Sam Kwong, Yu Zhou, and Qingfu Zhang. Matching-based selection with incomplete lists for decomposition multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 21(4):554–568, 2017.

[135] Miha Ravber, Marjan Mernik, and Matej Črepinšek. The impact of quality indicators on the rating of multi-objective evolutionary algorithms. *Applied Soft Computing*, 55:265–275, 2017.

[136] Tao Chen and Rami Bahsoon. Self-adaptive and online qos modeling for cloud-based software services. *IEEE Transactions on Software Engineering*, 43(5):453–475, 2016.

[137] Jinhwan Lee, Jing Zhang, Zhengqiu Huang, and Kwei-Jay Lin. Context-based reputation management for service composition and reconfiguration. In *2012 IEEE 14th International Conference on Commerce and Enterprise Computing*, pages 57–61. IEEE, 2012.

[138] Yuhong Yan, Pascal Poizat, and Ludeng Zhao. Repair vs. recomposition for broken service compositions. In *International Conference on Service-Oriented Computing*, pages 152–166. Springer, 2010.

[139] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. Qos-aware replanning of composite web services. In *IEEE International Conference on Web Services (ICWS'05)*, pages 121–129. IEEE, 2005.

[140] Carolyn Seaman, Yuepu Guo, Nico Zazworka, Forrest Shull, Clemente Izurieta, Yuanfang Cai, and Antonio Vetrò. Using technical debt data in decision making: Potential decision approaches. In *2012 Third International Workshop on Managing Technical Debt (MTD)*, pages 45–48. IEEE, 2012.

[141] Nanette Brown, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert Nord, Ipek Ozkaya, et al. Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 47–52, 2010.

[142] Yuepu Guo and Carolyn Seaman. A portfolio approach to technical debt management. In *Proceedings of the 2nd Workshop on Managing Technical Debt*, pages 31–34, 2011.

[143] Terese Besker, Antonio Martini, and Jan Bosch. Managing architectural technical debt: A unified model and systematic literature review. *Journal of Systems and Software*, 135:1–16, 2018.

[144] Areti Ampatzoglou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Paris Avgeriou. The financial aspect of managing technical debt: A systematic literature review. *Information and Software Technology*, 64:52–73, 2015.

[145] Areti Ampatzoglou, Apostolos Ampatzoglou, Paris Avgeriou, and Alexander Chatzigeorgiou. A financial approach for managing interest in technical debt. In *International Symposium on Business Modeling and Software Design*, pages 117–133. Springer, 2015.

[146] Will Snipes, Brian Robinson, Yuepu Guo, and Carolyn Seaman. Defining the decision factors for managing defects: a technical debt perspective. In *2012 Third International Workshop on Managing Technical Debt (MTD)*, pages 54–60. IEEE, 2012.

[147] Yuepu Guo, Rodrigo Oliveira Spínola, and Carolyn Seaman. Exploring the costs of technical debt management–a case study. *Empirical Software Engineering*, 21(1):159–182, 2016.

[148] Nico Zazworka, Rodrigo O Spínola, Antonio Vetro', Forrest Shull, and Carolyn Seaman. A case study on effectively identifying technical debt. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, pages 42–47, 2013.

[149] Martin Maechler. Package 'fracdiff', 2020 [online]. Available from: http://rsync.udc.es/CRAN/web/packages/fracdiff/fracdiff.pdf[Last accessed 20th March 2020].

[150] John Geweke and Susan Porter-Hudak. The estimation and application of long memory time series models. *Journal of time series analysis*, 4(4):221–238, 1983.

[151] Peter M Robinson. Long memory time series, 2018 [online]. Available from: https://personal.lse.ac.uk/robinso1/long-memory-time-series.pdf[Last accessed 17th December 2018].

[152] Ratnadip Adhikari and Ramesh K Agrawal. An introductory study on time series modeling and forecasting, 2013. *arXiv preprint arXiv:1302.6613, [online]*, Available from: https://arxiv.org/abs/1302.6613[Last accessed 24th May 2017].

[153] Tao Chen, Rami Bahsoon, Shuo Wang, and Xin Yao. To adapt or not to adapt? technical debt and learning driven self-adaptation for managing runtime performance. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 48–55, 2018.

[154] Milinda Pathirage, Srinath Perera, Indika Kumara, and Sanjiva Weerawarana. A multi-tenant architecture for business process executions. In *2011 IEEE International Conference on Web Services*, pages 121–128. IEEE, 2011.

[155] Krasimir Baylov, Dessislava Petrova-Antonova, and Aleksandar Dimov. Web service qos specification in bpel descriptions. In *Proceedings of the 15th International Conference on Computer Systems and Technologies*, pages 264–271, 2014.

[156] Friedhelm Bliemel. Theil's forecast accuracy coefficient: A clarification. *Journal of Marketing Research*, 10(4):444–446, 1973.

[157] Maciej Tomczak and Ewa Tomczak. The need to report effect size estimates revisited. an overview of some recommended measures of effect size, 2014, [online]. Available from: http://tss.awf.poznan.pl/files/3_Trends_Vol21_2014__no1_20.pdf[Last accessed 30th August 2019].

[158] Franco Raimondi, James Skene, and Wolfgang Emmerich. Efficient online monitoring of web-service slas. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 170–180, 2008.

[159] Danilo Ardagna and Barbara Pernici. Global and local qos constraints guarantee in web service selection. In *IEEE International Conference on Web Services (ICWS'05)*. IEEE, 2005.

[160] Martine De Cock, Sam Chung, and Omar Hafeez. Selection of web services with imprecise qos constraints. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI'07)*, pages 535–541. IEEE, 2007.

[161] Allan H Vermeulen and Jeffrey P Bezos. Method and system for dynamic pricing of web services utilization, June 22 2010. US Patent 7,743,001.

[162] Farhana Zulkernine, Patrick Martin, Chris Craddock, and Kirk Wilson. A policy-based middleware for web services sla negotiation. In *2009 IEEE International Conference on Web Services*, pages 1043–1050. IEEE, 2009.

[163] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Transactions on the Web (TWEB)*, 1(1):6, 2007.

[164] Mohammad Alrifai, Thomas Risse, and Wolfgang Nejdl. A hybrid approach for efficient web service composition with end-to-end qos constraints. *ACM Transactions on the Web (TWEB)*, 6(2):7:1–7:31, 2012.

[165] Frank Buschmann. To pay or not to pay technical debt. *IEEE Software*, 28(6):29–31, 2011.

[166] Jin Xiu and Yao Jin. Empirical study of arfima model based on fractional differencing. *Physica A: Statistical Mechanics and its Applications*, 377(1):138–154, 2007.

[167] Justin Q Veenstra and AI McLeod. Package arfima, 2015, [online]. Available from: https://mran.microsoft.com/snapshot/2017-12-11/web/packages/arfima/arfima.pdf[Last accessed 27th July 2017].

[168] William H Kruskal and W Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952.

[169] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. Academic press, 2013.

[170] Didac Gil De La Iglesia and Danny Weyns. Mape-k formal templates to rigorously design behaviors for self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(3):1–31, 2015.

[171] Rami Bahsoon Tao Chen and Xin Yao. A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems. *ACM Computing Surveys*, 51(3), 2018.

[172] Tiago Oliveira, Ricardo Martins, Saonee Sarker, Manoj Thomas, and Aleš Popovič. Understanding saas adoption: The moderating impact of the environment context. *International Journal of Information Management*, 49:1–12, 2019.

[173] Carlos Mera-Gómez, Rami Bahsoon, Rajkumar Buyya, and Escuela Superior Politécnica. Elasticity debt: a debt-aware approach to reason about elasticity decisions in the cloud. In *Proceedings of the 9th International Conference on Utility and Cloud Computing*, pages 79–88, 2016.

[174] Carlos Mera-Gómez, Francisco Ramírez, Rami Bahsoon, and Rajkumar Buyya. A debt-aware learning approach for resource adaptations in cloud elasticity management. In *International Conference on Service-Oriented Computing*, pages 367–382. Springer, 2017.