# RUNTIME QoS CONTROL AND REVENUE OPTIMIZATION WITHIN SERVICE ORIENTED ARCHITECTURE.

Miroslav Zivkovic.

# RUNTIME QoS CONTROL AND REVENUE OPTIMIZATION WITHIN SERVICE ORIENTED ARCHITECTURE

Miroslav Živković

**Graduation committee:**

| | |
|---|---|
| Chairman: | prof. dr. ir. A. J. Mouthaan |
| Promoters: | prof. dr. J. L. van den Berg |
| | prof. dr. R. D. van der Mei |
| Members: | |
| prof. dr. ir. L. J. M. Nieuwenhuis | Universiteit Twente |
| prof. dr. ir. A. Pras | Universiteit Twente |
| prof. dr. R. Núñez–Queija | Universiteit van Amsterdam |
| prof. dr. ir. D. H. J. Epema | Technische Universiteit Eindhoven |
| prof. dr. B. Pernici | Politecnico di Milano |
| dr. ir. H. B. Meeuwissen | TNO, Delft |

UNIVERSITEIT TWENTE.

**CTIT Ph.D. thesis Series No. 13–291**
Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE Enschede, The Netherlands

This work has been conducted at Dutch Organization for Applied Scientific Research (TNO) and Design and Analysis of Communication Systems (DACS) group, Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente within the context of the IOP Gencom project Service Optimization and Quality (SeQual), supported by the Dutch Ministry of Economic Affairs via its agency Agentschap NL.

ISBN: 978–90–365–3581–6
ISSN: 1381–3617 (CTIT Ph.D. thesis Series No. 13–291)
DOI: 10.3990/1.9789036535816
http://dx.doi.org/10.3990/1.9789036535816

Typeset with LaTeX. Printed by Ipskamp Drukkers B.V.

# RUNTIME QoS CONTROL AND REVENUE OPTIMIZATION WITHIN SERVICE ORIENTED ARCHITECTURE

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. H. Brinksma,
volgens besluit van het College voor Promoties,
in het openbaar te verdedigen
op donderdag 16 januari 2014 om 14.45 uur

door

Miroslav Živković

geboren op 3 augustus 1969
te Beograd, Servië

Dit proefschrift is goedgekeurd door de promotoren
prof. dr. J. L. van den Berg
prof. dr. R. D. van der Mei

# Contents

# Acknowledgment

This thesis is a result of a research that I have conducted under the supervision of prof. dr. Hans van den Berg and prof. dr. Rob van der Mei. I am thankful to both of them for the pleasant co–operation, their advice with respect to this research and the knowledge and time they invested in my work. I have especially enjoyed the most advanced Dutch course there could be with them, absolutely free of charge. I would also like to thank all members of the promotion committee, for accepting the invitation to join the committee and careful consideration of the work presented in this thesis.

The research presented in this thesis has been conducted within the scope of the project Service Optimization and Quality (SeQual) supported by Dutch Ministry of Economic Affairs, through its agency Agentschap NL. The support of Joep van Wijk and Geert Wessel Boltje from Agentschap NL is appreciated. The research done within the SeQual project is by no means the work of a single person – I enjoyed the collaboration with all SeQual partners and the discussions we had during our project meetings. Next to this, I would like to thank to all co–authors of the papers we published together and to Bart Gijsen (TNO) for his comments on the work I did.

"Where there is a will, there is a way" is an old saying that has been confirmed by Erik Meeuwissen and Robert Kooij (TNO) who made it possible for me to attend a conference where I presented the last paper used as the basis for this thesis. It goes without saying that this is highly appreciated. I would also thank Erik for his (long–term) support and advice, not only within the scope of the presented work.

During my carrier I had the pleasure to work at the Bell Laboratories and to collaborate, among others, with the colleagues from the Netherlands and Murray Hill, NJ. Above all, I thank Debasis Mitra, the former Vice President of the Bell Labs, for his genuine interest in my PhD, Phil Whiting and Sem Borst for their advice at the starting phase of the research presented here, and Adriaan J. de Lind van Wijngaarden for his support throughout all these years.

I thank all my friends around the world for many good memories, and my family for their support in good and bad times.

# Abstract

The paradigms of service–oriented computing (SOC) and its underlying service–oriented architecture (SOA) have received a lot of attention recently and have changed the way software applications are designed, developed, deployed, and consumed. Due to these paradigms, software engineers can realize applications by service composition, using services offered by third parties. In the competitive market of composite services, the commercial success of composite service providers (CSP) is directly related to their ability to offer services at sharp price/quality ratios.

This raises the need to realize desired client perceived Quality of Service (QoS) levels at minimal cost. The problem of controlling QoS in SOC is complex in that the ownership of the services is decentralized, as a composite service makes use of services offered by third parties. Although a plethora of well–known QoS–control mechanisms exists for "atomic" Web services used for the composition, it remains a challenge how to exploit these mechanisms for QoS–control in SOA in a cost–effective way. The great potential for composite service providers to realize dramatic cost savings and/or revenue improvements by optimizing the QoS–control in SOA has not been exploited much so far. To address this issue, proper modelling of the effects of QoS–control parameters is required. Once the models are specified, analysis of these models to derive the optimal settings of the parameters is a natural next step.

This thesis contributes models and methods to address these QoS–control issues within SOA. We develop the models of the runtime end–to–end QoS–control mechanisms, that are used to satisfy QoS requirements of an *individual* composite service request (e.g. response time) while optimizing some *long–term* goal (e.g. execution cost minimization, expected revenue). These models, based on *per–request, per–task* service selection, facilitate development (using, among others, dynamic programming approach) of simple, yet effective optimal decision–making policies in order to satisfy specified QoS levels. We demonstrate the effectiveness of the developed solutions as well as significant revenue improvements by extensive numerical experiments. The derived policies have negligible overhead with respect to the decision–making process and control actions to be taken by the CSP. Besides, the implementation of these policies is relatively simple, e.g. as a lookup table. The control actions may be automated, and allow for fast reactions to the changes in the volatile service execution environment.

In our view this thesis presents a significant step forward to envisioned autonomous,

economically profitable systems of services and applications of the future. Our approach opens many interesting opportunities for further research in the challenging area of QoS–control of such “system of systems”.

CHAPTER 1

# Introduction

The goal of this chapter is to introduce the research questions addressed in this thesis, and to detail our contributions. To do so, we first position and motivate the work presented in this thesis in Section 1.1. The problem statement is discussed in Section 1.2. In Section 1.3 we formulate our research questions, which are followed by an outline of the main contributions of the thesis in Section 1.4. Finally, in Section 1.5 the structure and organization of the thesis is presented.

## 1.1 Background and motivation

Almost 90% of the data in the world today has been created in the last two years alone [54]. According to the recent study [16], Americans consume on average 3.6 zettabytes ($3.6 \cdot 10^{21}$ bytes) per day. The consumed information is provided by many applications that have blended into our everyday lives. These applications are deployed at versatile platforms and use communication networks that span our world. The emergence of the Internet allows newly deployed applications and services to easily find the way to the millions of users. At the same time, the companies developing applications are engaged in global competition, and we witness a sharp decline in new applications' time–to–market. The ever shorter time–to–market is achieved as more and more applications are built by integrating and composing already existing services offered by different stakeholders.

The composition and integration of services is possible due to the fact that there is an ongoing evolution of (software) systems. In previous decades, software applications and services evolved from monolithic, mainframe–centric and centralized, via client–server systems and applications, to recent data–centric, dynamic and highly distributed. One of the latest "species" evolved during the course of this evolution is *Service–Oriented Computing* (SOC). SOC [77] is a computing paradigm that utilizes services as fundamental elements to support rapid, low–cost development of distributed applications in heterogeneous environments. Rather than building a software system "from scratch", according to the SOC paradigm, the applications are developed by composing autonomous, loosely–coupled, and platform–independent

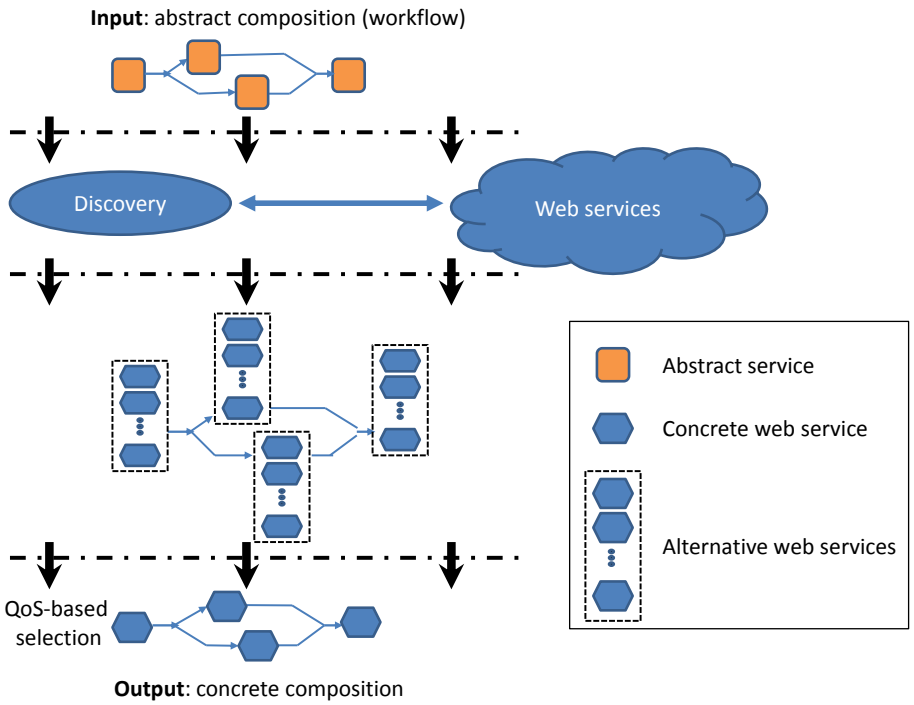


Figure 1.1: Conceptual overview of service composition. Abridged from [2].

networked services [40, 78]. These services can be discovered and invoked by different participants and used for different applications. The architecture that underlies SOC is *Service Oriented Architecture* (SOA) [45, 53, 80]. The process of building service–oriented applications from existing services is known as *service composition* and the result of the composition process is called a composite service [35]. The problem of *service composition*, i.e. how to select services that implement tasks within a given workflow, is one of the most challenging ones within SOC. Service composition should satisfy both functional and non–functional (e.g. Quality of Service) requirements of the designer that performs the service composition, and we denote this composition *QoS–aware service composition*.

In Figure 1.1 a conceptual overview of the QoS–aware service composition problem is given [2]. For a given business description of the service composition (i.e. *workflow*), there are (potentially) many functionally equivalent services implementing a single task within given workflow. These *concrete services* for each task in the workflow are identified by the discovery engine using syntactic/semantic functional matching between the tasks and service descriptions. This results in a list of functionally equivalent alternative services per task. These alternative services differ only with respect to their QoS properties. The goal of QoS–aware service selection is to select one or more concrete services from each list so that the aggregated QoS values

satisfy the client's end–to–end QoS requirements. The problem of optimal QoS–aware service composition, i.e. the selection of services whose composition results in the "best" QoS of composite web services, is known to be a NP–hard problem [2,8,21].

## 1.2 Problem statement

As the services used for composition are usually deployed in a volatile execution environment, performance–related problems could occur relatively often during services' execution processes. The QoS experienced by the end–user of a composite service depends on the QoS levels realized by the individual services; a poorly performing service used for composition may strongly impact the end–to–end QoS of a composite service. This may lead to the client's dissatisfaction and loss of revenue for composite service providers (CSP). To address this problem a number of *re–active* adaptations based on monitoring of services' execution process have been proposed. These solutions usually require human intervention for the root–cause analysis of the problem, and alternation (i.e. adaptation of) the service composition in order to improve on QoS.

There is a need for *pro-active* run–time end–to–end QoS–control that properly responds to *short–term* QoS degradations. The actions (controls) are taken in order to satisfy end–to–end QoS requirements of an *individual* composite service request (e.g. response time) while optimizing some *long–term* goal (e.g. execution cost minimization). The actions (controls) taken for a single composite service request depend on information gathered using real–time monitoring of QoS. In some cases, the QoS–control may require run–time service selection for each task within a given workflow.

The actions taken would certainly impact the revenues of the CSP. For example, one of the alternative services for a single task from Figure 1.1 may provide faster response (i.e. smaller response time) than other services, but this comes with a price, as this service is likely to be more expensive than slower alternatives. The question for the CSP is how much more expensive it is to execute the adapted service composition as compared to the original one. This important aspect of the *cost of control* has been in general neglected so far, with a noticeable exception of [62].

There is an apparent tradeoff between the CSP's objective to realize and retain end–to–end QoS guarantees to its customers and at the same time optimize its revenue. The problem is that the available QoS–control concepts are powerful, but that today little is known about how to cost–efficiently exploit the possibilities for QoS–control in SOA.

Run–time mechanisms are particularly promising (because they allow for per–request state–dependent control decisions), but the proper use of these complex mechanisms in the volatile and heterogeneous SOA environment is highly challenging. In this thesis we develop and analyze quantitative models and methods for the optimal use

of these mechanisms, balancing the complex trade–off between guaranteeing QoS on the one hand and minimizing cost (optimizing revenue) on the other hand.

## 1.3 Main research questions

The problems outlined in Section 1.2 motivate the research conducted as part of this thesis work. More specifically, the following central research questions are addressed in this thesis:

1. How to model the effect of the parameter settings of the QoS–control mechanisms on the end–to–end QoS? The models should capture the dominant factors that influence QoS yet allow for (mathematical) analysis and optimization.
2. How to analyse and use these models to derive optimal settings of the parameters of the control mechanisms? The optimal settings should maximize (long–term) revenue subject to pre–set QoS requirements and related costs and rewards/penalties.

## 1.4 Contributions

Guided by the research questions stated in Section 1.3, the main contributions of this thesis to the state–of–the–art in service composition research can be summarized as follows:

**Contribution I** We develop models and derive rules for the optimal setting of so–called admission control mechanisms for QoS control of composite services. These highly effective mechanisms are aware of the execution state within the composition.

**Contribution II** We develop a model and investigate the performance potential of dynamic service selection based on delayed state information.

**Contribution III** We develop models and methods to identify optimal stopping policies that may be applied during the execution of a workflow. These policies are derived with the objective to maximize revenue of the CSP subject to end–to–end QoS constraints.

**Contribution IV** We develop various models for the analysis of QoS–aware per–request run–time service composition. Using these models, we formulate algorithms to identify optimal policies for dynamic service compositions, subject to end–to–end QoS constraints.

**Contribution V** We develop models and methods to identify optimal setting of conditional retry mechanisms for service composition. A retry (service request)

is generated when a service selected to execute a task within a workflow does not generate the response within a pre–determined time.

## 1.5 Overview of the thesis

The remainder of this thesis is organized as follows:

**Chapter 2** provides background information on relevant concepts and techniques, as well as research related to the remainder of this thesis. We discuss the main relevant ideas of SOA and state–of–the–art QoS–aware service composition and QoS–control mechanisms.

In **Chapter 3** overload control for composite Web services in SOA is studied. Two practical admission control rules are developed resulting in effective mitigation of the severe overload effects for the service composition. The objective of these rules is to keep end–to–end response time and availability at agreed QoS levels. The theoretical background and design of these admission control rules as well as performance evaluation results obtained by both simulation and experiments are discussed. This results of this chapter are the foundation of our first contribution.

In **Chapter 4** the *performance potential* of dynamic (run–time) Web service *selection* is investigated for a single task implemented by a number of alternative concrete services. The response times for the case of static web service selection and for the case where dynamic web service selection is applied, are compared. Simulation results are presented for different run–time selection strategies in scenarios ranging from the "ideal" situation (i.e. up–to–date state information, no background traffic) to more realistic scenarios in which state information is stale and/or background traffic is present. In particular, the effectiveness of a selection strategy based upon the "synthesis" of the Join the Shortest Queue and Round Robin strategies is illustrated. For some specific scenarios we derive and validate insightful (approximate) analytical results for the response times. The results of this chapter represent our second contribution.

In Chapters 5–8 we focus on the problem of *per–request, per–task* service selection for composite web services represented by workflows that contain multiple tasks. The CSP is rewarded by its clients when the achieved end–to–end response time is smaller then the promised deadline. The *long–term* objective of such dynamic, run–time service selection strategies is to maximize the profit of the CSP, taking into account the execution costs incurred by the CSP per service request. We gradually increase the amount of additional choice that is utilized to make decisions. The last three contributions of this thesis are established in chapters 5–8.

In **Chapter 5** the problem of expected profit maximization for orchestrated sequential service composition is investigated. The main question addressed is whether to terminate the execution of a single request within the composite service workflow, and if so, *when* to do it. Depending on the actual response time of executed services,

and taking into account the execution costs, as well as revenue and penalty functions, a dynamic programming based algorithm and a heuristic solution are compared. For both solutions the revenue gains are quantified when compared to the baseline case of static service composition.

In **Chapter 6** the question of run–time dynamic service composition is addressed. For each of the tasks within a given workflow, a number of service alternatives may be available, offering the same functionality at different price/quality levels. We present a fully dynamic service composition approach, where for each individual request and each task in the workflow it is decided at runtime which service is invoked. The decisions are based on observed response times, costs and response time characteristics of the alternatives as well as end–to–end response time objectives and corresponding rewards and penalties. We derive the service selection policy maximizing expected revenue using a dynamic programming approach. Extensive numerical experimentation demonstrates huge potential gain in expected revenues using the dynamic approach compared to other, non–dynamic approaches.

In **Chapter 7** the revenue improvements for the orchestrated composite service are quantified for the case when *service availability* is taken into account next to the non–functional QoS parameters considered in the model in previous chapters. The availability of services is represented by an (a–priori) known probability. Which service alternatives are available for the task that is to be executed is known at the decision moment.

In **Chapter 8** optimal run–time service selection methods based on *conditional request retries* are investigated, and the model specified in Chapter 6 is extended to accommodate for these requests. Here, the actual service requests may be used as probes to assess whether a service used for composition is available or not. We also extend upon the analysis presented in Chapter 7, as this analysis assumes that it is "known" (e.g. due to constant monitoring of the services) which of the service alternatives are available for given task. Instead of constant monitoring of the services by probes, i.e. service requests that are not part of the composite service execution, the actual service requests are used to determine whether the selected service is transiently non–responsive (i.e. not available) and what is the optimal moment to make such a conclusion. When the service that has been selected to serve the request is not responsive, an alternative service implementing the same task may be selected, provided that the benefits of such action (reward for CSP when deadline is met) outweigh the costs of it (penalty and additional execution costs).

**Chapter 9** concludes the thesis and provides an outlook on future research directions opened up by this work.

CHAPTER 2

# Service Composition and Quality Control

In Chapter 1 we scoped this thesis to end–to–end QoS control of Web service compositions within SOA. In this chapter, we will provide background on the techniques, approaches and solutions that are the most relevant for our work.

In Section 2.1 we introduce the most important elements and characteristics of SOA relevant for the problem at hand. In Section 2.2 we discuss the Web services and Web services protocol stack since Web service technology is currently the most prominent realization of SOA. In Section 2.3 two general types of service composition, namely orchestration and choreography are defined. The main concepts of QoS and SLAs are presented in Sections 2.4 and 2.5, respectively. The QoS control mechanisms relevant for this thesis, namely admission control and QoS–aware service composition, are discussed in Section 2.6. Finally, an extensive state–of–the–art overview of these control mechanisms is given in Section 2.7.

## 2.1 Service Oriented Architecture - basic concepts

There are many different definitions of SOA and its main characteristics, especially among practitioners. This resembles a bit the famous poem of John Godfrey Saxe entitled "The Blind Men and the Elephant" [85]. In it, six blind men from Indostan encounter an elephant – each of the men then describes the elephant differently because they are influenced by their individual experiences (see Figure 2.1). SOA was first introduced in 1996 by the Gartner Group [87]. SOA originates from object–oriented and component–based software development, and aims at enabling developers to build collaborative applications, regardless of the platform where the applications run and of the programming language used to develop them. This is achieved by the use of independent software units, called services.

A *service* is a valuable resource offered by a provider usually for a fee. The resource may be physical or a process that is made available for use by others. In general, SOA consists of three classes of entities: the *providers*, *consumers* and *registries*. Services are typically discovered through a service registry, which decouples service provider and service client. In this way, the well–known SOA "triangle" [52] is established,


Figure 2.1: Six blind men and the elephant. Taken from [66].

see Figure 2.2. A service–oriented system can have many consumers and providers that are possibly located anywhere on a computer network.

Don Box summarized the essential common principles of SOA into four tenets [20]:

- *Boundaries are explicit.* A service–oriented application consists of services that are spread over large geographical distances, ownership and trust domains, and operation environments. In order to reduce the cost of cross–boundary communication, explicit message passing is applied for services rather than implicit method invocation.
- *Services are autonomous.* Services are independently deployed and a deployed service does not assume the existence of its consumers. The topology of a service–oriented system is dynamic, i.e. changing with time. New services may be introduced to the system without announcements. The applications consuming a service can leave the system or fail without notification.
- *Services share schema and contract, not class.* Services interact by message passing. Message structures are specified by schemas, and message–exchange behaviours are specified by contracts.
- *Service compatibility is based on policy.* A service has a set of policies that

depict the properties of interaction with a service, e.g. security protocols, transactional properties, and so on.

So far, the most common realisation of SOA is achieved by using *Web services* [11, 102]. Web services provide a set of technologies to support the various elements that include service description (WSDL, [51, 105]), service discovery (UDDI, [93]) and service binding (SOAP, [104]). The Internet is the ubiquitous underlying infrastructure of Web Services, and therefore Web Services inherit both the virtues and vices of the Internet. The virtues include pervasiveness, ubiquity, openness and flexibility. The vices originate in the open nature of the Internet that is a possible cause of many QoS–relating issues. As the web service technology is

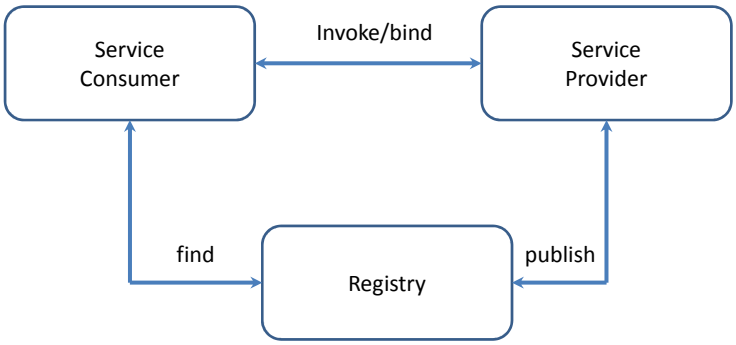


Figure 2.2: Basics of the Service Oriented Architecture.

currently the most prominent realization of SOA, we explain it in some more detail in the next section.

## 2.2 Web Services

The World Wide Web Consortium (W3C) defines two major classes of Web services: arbitrary Web services and REST–compliant Web services. Arbitrary Web services comprise the following core open technologies and specifications (see also Figure 2.3): Extensible Markup Language (XML) and XML Schema Definition Language (XSD), Standard Object Access Protocol (SOAP), Web Services Description Language (WSDL) and Universal Description, Discovery, and Integration (UDDI). Communication between services is message based and specified in standards. SOAP is a XML–compliant specification recommended by W3C as the communication standard for Web services. SOAP messages are transported using the Hypertext Transport Protocol (HTTP). WSDL specifies how to construct a SOAP message to be able to communicate with a service. The WSDL interface of a service describes the operations supported by the service. UDDI is a mechanism to register and locate web service applications on the Internet.

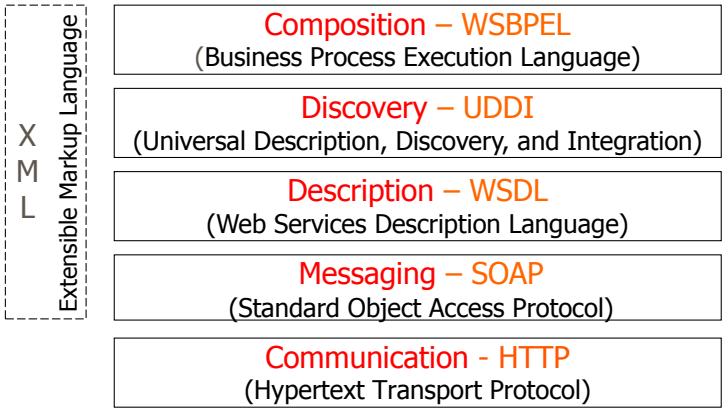


Figure 2.3: Overview of Web Service protocol stack and standards.

A rather new style for designing web services is the Representational State Transfer or REST–style architecture. The term REST was first introduced in 2000 by Roy T. Fielding in his PhD thesis [41]. REST ignores the details of component implementation and protocol syntax and focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements [42]. A RESTful Web service is based on the concept that a service consists of different sources of specific information, each of which is referenced with a unique global identifier. The resources can be manipulated by exchanging documents through a standardized interface using a limited number of methods in HTTP 1.0 or extensions as defined in HTTP 1.1.

We do not make a distinction in this thesis between above–mentioned two classes of Web services. We are mainly focused on non–functional, i.e. QoS properties of these services. The actual implementation and class of Web service are of little importance to us here.

## 2.3 Service composition

One of the basic claims of Web services is their composability [52] into value–added structures, so–called service compositions. Service composition is often cited as one of the key research topics in SOC [78]. Generally, two types of composition (see also Figure 2.4) can be distinguished [62, 79]:

- *Service orchestration* refers to compositions where one central controller "orchestrates" (steers) the execution logics. The execution control is centralized in a single composition engine. The service orchestrations are intra–organizational and reflect business processes within an organization, although

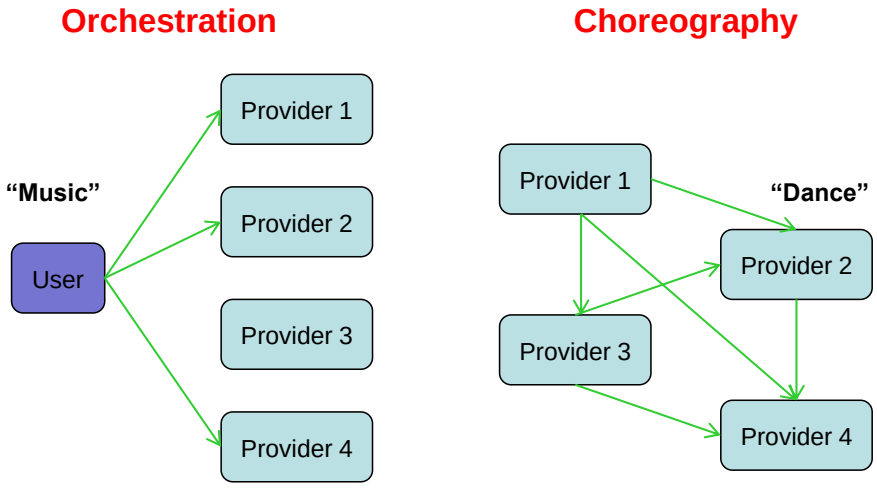


Figure 2.4: Two basic types of composition: orchestration (left) and choreography.

some of the services used in the orchestration may well be external to the organization. Service orchestrations are the compositions most commonly seen in practice, and many tools and languages exist to model orchestrations.

- *Service choreography* is the more complex case of service composition, and no central controller exists in it. Every organization participating in the choreography has its own composition engine, and control over the execution is passed between those engines using well–defined interfaces. There is no entity with knowledge of the whole composition. Each partner only knows its own internal execution flow and the interfaces of the entities it interacts with. Due to this, choreography has proven challenging to implement in practice and has gained little attention so far.

The work described in this thesis deals solely with issues of service orchestration. Hence, in the remainder, the terms composition and orchestration are used interchangeably.

## 2.4 Quality of Service

According to ISO 8402 [55], the quality of software is defined as "The totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs". Those needs can be interpreted in various ways depending on an application domain. Although definitions of the basic QoS metrics in literature [59, 82] may vary to a certain extent, one of the most common ways to define

the most frequently used QoS metrics is the following [36, 67]:

**Availability:** Availability describes the ratio of the time the service is available for accepting requests over the total time.

**Performance:** Performance includes such metrics as latency, service response time and service throughput.

- *Throughput* is defined as the number of requests served per unit of time.
- *Response time* refers to the time required for processing a single request.
- *Latency* is the time elapsed since a request from a client has been sent and a corresponding response was obtained. From a client's perspective, latency includes network delay and response time.

**Reliability:** Reliability refers to the ratio of the successful service invocations over the total number of service invocations. A service invocation is considered successful if it does not result in an exception or failure on the service site.

In this thesis, we have mainly considered response time (latency), availability and reliability QoS metrics. These QoS metrics are usually part of Service Level Objectives (SLOs), which themselves are part of SLAs. Next to the QoS metrics (SLOs), service level agreements usually specify the execution costs, which refer to the amount of money that the service consumers pay to the service provider for the execution (i.e. usage) of its services. Further, SLAs may specify the amount of money that the service provider reimburses to its clients when one or more service–level objectives are not been met, in the form of a *penalty*. This is further described in the next section.

## 2.5 Service Level Agreements

A SLA is a legal contract that specifies the minimum expectations and obligations that exist between a service provider and a service consumer [100]. It has to be mutually agreed by both sides. A typical SLA consists of two main parts. The first part represents the functional aspects of the respective service, e.g. what does the service do, which input parameters are required, in which format, which results the service sends back to the consumer, and so on. The second part specifies the guaranteed aspects which include, among others, SLO, service level evaluation rules, measurements criteria, and ramifications of failing to meet (or indeed exceeding) these SLOs. The refund policies (penalties) for service–level violations can be specified relative to the service cost or in absolute terms.

Even without a SLA, a service can still be invoked. In such a scenario, there is no QoS guarantee, and no–one is responsible in the case of service changes due to poor performance, functional service change, QoS constraint change, and so on [84].

In this thesis we consider two different SLA types: a) the SLA between the CSP and its clients (composite SLA) and b) the SLA between the CSP and third–party domains ("individual" SLA). The composite SLA specifies *nominal* response–time SLO, i.e. the a single value as the end–to–end response–time deadline. Therefore the CSP guarantees response times smaller than a certain value. Besides, this SLA may specify (or omit) the fraction of response time realisations that should be within the deadline. So we consider both *hard* and *soft* response–time SLOs in the composite SLA. Besides, the composite SLA contains possible reward/penalty per composite service request when end–to–end deadline per individual request is met/missed.

The usage of nominal response–time SLOs within individual SLAs lead to pessimistic response–time targets and may be overly inefficient as shown in [83]. Therefore, the SLA agreed between the third–party domains and the CSP specifies *soft* response–time SLO, and it is represented by the response–time PDF. The individual SLA also specifies the execution costs, i.e. how much the CSP pays to the third–party domain for the execution of a single request. From the viewpoint of third–party domain, this value represents reward.

The penalties considered in this thesis are in the form of penalty functions. The *penalty function* specifies the amount of money that the service provider reimburses to its clients when one or more objectives that are part of a SLA have not been met. There are different functions that could be used for this purpose [17,62], but we have mainly considered the case of *constant penalty* for the composite service provider in this thesis. This means that the CSP pays back to its clients a fixed amount of money for each composite service request that missed the response–time deadline promised to the customers.

We also assume that, once agreed, SLAs are "static", i.e., do not change during the execution of the composite service.

## 2.6 QoS control mechanisms

In order to realise the performance guarantees (satisfy QoS levels) given to its clients, the Web service provider may apply different QoS–control mechanisms. These mechanisms are, among others, caching, load balancing, content adaptation, admission control and request scheduling [47]. These QoS–control mechanisms are usually applied to a single Web server/Web service.

In this thesis we consider end–to–end QoS control mechanisms for composite service, and have in particular focused on admission control and dynamic service selection in QoS–aware service composition as the control mechanisms. Therefore, we first briefly describe these two mechanisms, and then focus on state–of–the–art overview of admission control and QoS–aware service composition in the next section. At the end, we briefly describe how we extend current research in this thesis with respect to application of these mechanisms within SOA.

### 2.6.1 Admission control

The performance of a Web server/Web service can be affected due to many reasons, e.g. "flash crowds", sudden execution of background jobs at the server, network/server failure, etc. Some of these reasons are the principal cause of an *overload* of the Web system, which typically means that the performance of this system deteriorates. One way to mitigate the system overload is *admission control*, i.e. a design of admission control policies that will prevent the overload situation. A good admission–control mechanism improves the Web service performance during overload by only admitting a certain limited amount of customers' request at a time to the service. The reasoning behind this is that it may be better to reject some requests in order to complete the other requests and thereby generate some revenue for the provider.

An admission control mechanism typically consists of three parts: a *gate*, a *controller*, and a *monitor*. The monitor measures one or more so–called control variables, and based on the gathered information, the controller decides the rate at which requests can be admitted to the system. The gate rejects requests that cannot be admitted. Optionally, a notification message is sent to a rejected client. The requests that are admitted to the system are served.

There are two basic types of admission control schemes that may be used in Web services: *request–based* and *session–based*. In a request–based scheme there is an upper limit of the number of requests served by the provider. The customers may be therefore rejected in the middle of their sessions. In a session–based admission control scheme, once a customer has been admitted, the the customer is guaranteed that session would be completed.

### 2.6.2 Dynamic service selection as part of QoS–aware service composition

The process of building service–oriented applications from existing services is known as *service composition* and the result of the composition process is called a *composite service* [35]. The term service composition is very broad and can be classified in many different ways. We have seen in Section 2.3 that there are two basic classifications, namely orchestration and choreography. Another possible research domain with respect to the service composition is *QoS–aware service composition*.

QoS–aware service composition refers to the composition that is carried out with the aim to satisfy both functional and non–functional (e.g. QoS) requirements of the developer. The QoS–aware service composition may be either *static* or *dynamic*. Static service composition takes place before a composite service is actually deployed, and implies re–active root cause analysis and adaptation (by humans) of the composite service. Dynamic service composition involves adaptation(s) during the execution of a service composition.

The *QoS–based service selection* is a common QoS–control mechanism for QoS–aware service composition. QoS–based service selection leads to the optimization problem to find a service per workflow task from a list of candidate services with the objective to satisfy certain predefined end–to–end QoS goals. This problem is known to be NP–hard in general, and a number of heuristics have been suggested. In case of dynamic service composition, service selection may happen during runtime, and takes into account the perceived QoS–levels. That requires the optimization problem to be re–solved, and implies delay in the decision–making process. Therefore, state–of–the–art solutions perform the per–task selection of services before a composite–service request is served. In general, the service composition remains the same for a given request.

In this thesis we focus on *dynamic*, *per–request*, *per–task* QoS–aware orchestrated service composition, with a minimal impact on the decision–making process. A short overview of research in the area of QoS–aware service composition is given in the next section.

## 2.7 State–of–the–art

In this section we give a high–level overview of the state–of–the–art for admission control and dynamic, QoS–aware service composition. Specific related works and their relationship to our contributions are discussed in the corresponding chapter of each contribution.

We first present specific related works and conclude each sub–section with a high–level overview of our extension of presented research.

### 2.7.1 Admission control

The problem of Web server/Web service admission control has received a lot of attention so far. The most significant Web admission control approaches are briefly described here.

Kanodia and Knightly [57] develop a multi–class session admission control based on latency targets. The authors propose a server architecture having one request queue per service class. The admission controller attempts to meet latency bounds for the multiple classes using measured request and service rates of each class. Aweya et al. [10] propose a load–balancing scheme in a cluster of Web servers that includes an admission control algorithm based on the CPU utilisation metric, that is retrieved from the Web servers at fixed intervals. The acceptance rate of client requests is adaptively set based on the CPU performance measures. Cherkasova and Phaal [27], consider the rejection of sessions when the Web server is overloaded in order to avoid the consumption of the server resources by a user session that may be interrupted. The metric used to monitor the Web system performance is the CPU utilisation,

which is measured during predefined time intervals and used to compute a predicted utilisation. In case the predicted utilisation exceeds a threshold, the new sessions that arrive for the next interval are rejected, but the service of already accepted sessions continues. Once the observed utilisation drops below the given threshold, the server begins to admit and process new sessions again.

Elnikety et al. [39] develop a session–based admission control and user–level request scheduling for e–commerce Web sites. The admission control algorithm determines if admitting the request will exceed the capacity of the system by using an estimation of the resource usage for that request. The requests are scheduled using their expected processing times in a Shortest–Job First (SJF) queue. The load of the system is measured each second, but the admission control is executed each time a servlet requests a database connection. One of the benefits of this proposal is that it does not require any modification in the operating system nor the Web software (Web server, application server or database).

Kihl and Widell [58] monitor the processing delay of each request and, based on this information, the admission control algorithm considers the admission, or not, of a new session or request on commercial QoS–aware Web sites. Chen and Mohapatra [26] design a scheduling scheme based on the session–level traffic model. They propose a Dynamic Weighted Fair Sharing (DWFS) scheduling algorithm to control overload in Web servers that is based on the probability of of the session that the requests belong to. Bartolini et al. [14] describe a policy that switches between two modes depending on the arrival rate detected. When the system is not overloaded, their approach takes admission control decisions at fixed intervals of time. In case the arrival rate exceeds a limit, the admission control decisions are taken each time a new session arrives to the system.

**Our extension of State–of–the–Art**

Although admission control has received a lot of attention, the majority of the solutions developed till now are applied within the context of a single domain, either a single server/service or, more recently, service farm deployments. These solutions do not include awareness of the execution state of the workflow within a service composition. In this thesis we focus on admission control rules that can be applied on *per–request*, *per–task* basis within a service composition. In order to achieve this, the orchestrator keeps track of the execution state within the considered composition.

### 2.7.2 QoS–aware service composition

QoS–aware service composition has recently been surveyed by Xianglan et al. [108], and Strunk [90]. A popular field of research is QoS–based service selection. QoS–based service selection finds an assignment of services (from a set of selected services) to tasks within the workflow which maximizes a certain QoS–relating utility function. This leads to the multidimensional optimization problem that is NP–hard [113]. Popular techniques in literature to solve this challenge efficiently are

integer programming (Zeng et al. [113]) and genetic algorithms (Canfora et al. [21]).

Zeng et al. [112, 113] present a QoS–aware composition approach based on state diagrams to model a composition. A composition is split into multiple execution paths, each considered to be a directed acyclic graph. For local optimization they use Multiple Criteria Decision Making (MCDM) to choose a service which fulfils all requirements and has the highest score. Global optimization is achieved by using a naive global planning approach (high runtime complexity) and an Integer Programming (IP) solution. The authors also describe an approach to re–plan and re–optimize a composition based on the fact that QoS can change over time. Therefore, a composition is split into regions according to the state of the tasks that allow a re–planning by adding constraints of what has already been accomplished to optimize services that still have to be executed.

Canfora et al. [21] propose an approach to solve the QoS–aware composition problem by applying genetic algorithms. The genome represents the composition problem by using an integer array where the number of items equals the number of distinct abstract services. Each item, in turn, contains an index to the array of the concrete services matching that abstract service. The cross–over operator is a standard two–point cross–over, while the mutation operator randomly selects an abstract service (position in the genome) and randomly replaces the corresponding concrete service with another one from the pool of available concrete services. The selection problem is modeled as a dynamic fitness function with the goal to optimize the QoS attributes. Additionally, the fitness function must penalize individuals that do not meet the QoS constraints. The approach is evaluated by comparing it to well–known integer programming techniques. The authors also describe an approach that allows re–planning of existing service compositions based on slicing [22].

Ardagna, Pernici et al. [7, 8] propose a QoS–aware optimization approach using dynamic service selection where each service in the composition process can be subject to global and local constraints which are fulfilled at runtime through adaptive re–optimization. The authors apply loop–peeling techniques to optimize loop iterations and negotiation techniques to find a feasible solution to the optimization problem. They solve the optimization problem, in particular the fulfilment of global constraints, under more stringent constraints.

An efficient global optimization approach for QoS-aware service composition supporting global constraints on a composition level is proposed by Alrifai and Risse [2]. The authors decompose global QoS constraints into local constraints with conservative upper and lower bounds. These local constraints are resolved by using an efficient distributed local selection strategy. The proposed solution consists of two steps: first, the authors use mixed integer programming (MIP) to find the optimal decomposition of global QoS constraints into local constraints. In the second step, they use distributed local selection to find the best Web services that satisfy these local constraints. Although this approach is highly efficient compared to existing work supporting only hard constraints, it does not allow to specify global soft constraints.

Jaeger et al. [56] present an approach for calculating the QoS of a composite service by using an aggregation approach that is based on the well–known workflow patterns by Van der Aalst et al. [95]. The authors analyze all workflow patterns and then derive a set of abstractions that are well–suited for compositions, so–called *composition patterns*. Additionally, the authors define a simple QoS model consisting of execution time, cost, encryption, throughput, and uptime probability including QoS aggregation formulas for each pattern. The computation of the overall QoS of a composition is then realized by performing a stepwise graph transformation. It identifies a pattern in a graph, calculates the QoS according to pre–defined aggregation functions and replaces the calculated pattern with a single node in the graph. The process is repeated until the graph is completely processed and only one single node remains. For optimizing a composition, the authors analyze two classes of algorithms, namely the 0/1-Knapsack problem and the Resource Constrained Project Scheduling Problem (RCSP). For both algorithms, a number of heuristics are defined to solve the problems more efficiently.

Yu et al. [111] discuss algorithms for Web service selection with end–to–end QoS constraints. Their approach is based on several composition patterns similar to [56] and they group their algorithms according to flows that have a sequential structure and others that solve the composition problem for general flows (i.e., flows with splits, loops etc). Based on this distinction, two models are devised to solve the service selection problem: a combinatorial model that defines the problem as Multidimensional Multi–Choice Knapsack Problem (MMKP) and the graph model that defines the problem as a Multi–Constrained Optimal Path (MCOP) problem. These models allow the specification of user-defined utility functions to optimize some application–specific parameters and to enable the specification of multiple QoS criteria taking global QoS into account. In the case of the combinatorial model, the authors use a MMKP algorithm that is known to be NP–complete, therefore, heuristics are applied to solve the problem in polynomial time. For the general flow structure, the authors use an IP approach (also NP–complete), thus they again apply different heuristics to reduce the time complexity.

**Conditional retry**

Dynamic QoS–aware service composition may be also achieved using *retries*. When a Web service is invoked by a client, it expects a response to be generated. If the reply does not come, either the service is down/overloaded, or some network/service failure occurred. In the latter case, a retry may be issued, and the response may be generated. Otherwise, within a typical SOA environment, a retry may be issued to a different, functionally equivalent service instead of the original one.

The retry as a solution for temporarily unavailable services, have been identified and classified, among others, in [7, 13]. The performance of basic retry mechanisms has been analysed in detail by van Moorsel, Wolter, et al. [97, 98, 103]. Their work has focused on optimal retry mechanisms for a single service with the objective to minimize the expected response time. Okamura et al. [74] analyse the restart policies when response–time deadline is given and develop on–line adaptive algorithms for

estimating the optimal restart time interval via reinforcement learning. The cost of the retries are defined as additional time to re–issue the service request.

Yousefi et al. [110] describe a strategy for QoS–aware service selection which takes advantage of the existing variability in QoS data to provide higher quality services with less cost compared other QoS–aware service selection methods. In their method, *each request* is replicated over multiple independent services to achieve the required QoS, i.e. to limit the response time by a certain pre–assigned value.

**Our extension of State–of–the–Art**

With respect to the dynamic QoS–aware service composition, we focus on *per–request, per–task* QoS–aware service composition, taking into account different price/quality levels, as well as reward/penalty functions. We try to perform the optimal service selection for the task at hand, with the long–term objective such as profit maximization of the CSP. By doing this, we take into account the cost of control (adaptation of service composition) as well. Besides, we consider the *conditional request retries* as one option for dynamic, QoS–aware service composition. We consider conditional retries for service compositions described by a workflow that may have more than one task. The request retries are issued only when the orchestrator estimates that the invoked service (for a given workflow task) is non–responsive. The estimation is performed using the actual request, and following a "watchdog timer" approach. Once the timer expires, a retry is issued to e.g. one of the available alternative services implementing the same task.

CHAPTER 3

# Intelligent Overload Control for Composite Web Services

As discussed in Chapter 2 the overload of Web services lead to reduced availability as well as higher response times, resulting in degraded quality as perceived by end users. Although admission control schemes have been widely accepted and applied, the application of these control schemes for orchestrated Web services poses a number of challenges that need to be addressed. One challenge results from the fact that any provider of the service used for the composition can apply its own admission control rules. There may not be an alignment of admission control between third–party provider and CSP. The other challenge is that orchestrator needs to apply admission control rules that take into account *actual* QoS levels (e.g. response time) before the next task in the workflow is to be executed. The admission control schemes that include awareness of the execution state of the workflow in a composition of web services, has been rarely analysed so far.

In this chapter, we focus on overload control for orchestrated Web services in SOA. Specifically, we investigate how orchestrator can deny service for some of the tasks in order to keep overall web service performance (in terms of end–to–end response time and availability) at agreed QoS levels.

The main contributions of this chapter are as follows:

- Modeling of admission control for orchestrated Web services using queueing theory based models.
- Design of two admission control rules for orchestrated Web services using the proposed models.
- Evaluation of these control rules with respect to both performance and availability conducted by simulations. Besides, experimental validation of one of the rules is conducted using actual composite services.

The rest of the chapter is organized as follows. In Section 3.1, we describe the overload control problem and provide the queueing theory based model of analysed system. In Section 3.2, we provide a brief overview of related literature. In Section 3.3, two algorithms for admission control by the orchestrator are derived from

the model adopted in Section 3.1. In Section 3.4, the simulation setup to investigate our solutions is described as well as two simulation cases. In Section 3.5, the results of an experimental validation are described. In Section 3.6, we conclude the chapter with suggestions for the future work.

This chapter is based on paper [68].

## 3.1 Problem description and modelling

Figure 3.1 shows a simplified orchestrated SOA architecture that illustrates our problem setting. The composite web service comprises of three web services identified by $W_1$ through $W_3$. The orchestrator consists of a scheduler and a controller. The scheduler determines the order of the requests to web services $W_1$ through $W_3$, since it may be different per client. The controller implements web admission control (WAC) mechanisms. It may happen that web services $W_1$ through $W_3$ implement WAC mechanisms themselves.

To illustrate operation without overload, let us suppose that a request from Client 1 (#1) arrives at the orchestrator. The scheduler analyses the request, and determines that the web service $W_1$, $W_2$, and $W_3$ should be invoked in that order. Before delegating a first job to $W_1$, the controller decides that $W_1$ is not in overload and assigns it to $W_1$ (#2). On the response (#3) from $W_1$, the scheduler requests the controller permission to invoke a next job at $W_2$ (#4), and so on until all web services are invoked, and the response (#10) to Client 1 is generated within a given deadline. To demonstrate an overload situation, let us suppose that a request from Client N (#11) arrives at the orchestrator. The scheduler analyses the request, and determines that the web services $W_1$ and $W_2$ should be invoked in that order. Before delegating a first job to web service $W_1$, the controller decides that $W_1$ is not in overload and assigns it to $W_1$ (#12). On the response (#13) from $W_1$, the scheduler requests the controller permission to invoke a next job at $W_2$, which is denied as $W_2$ is in overload. As a result, the orchestrator is able to respond to Client $N$ with a service unavailable message (#14) within given deadline as well as to prevent escalation of the overload situation of $W_2$. We can see that in the described overload situation resources of web service $W_1$ have been wasted. Providers of web services $W_1$ through $W_3$ may apply different state–of–the–art techniques, such as over dimensioning of computing resources, load balancing, and caching, to prevent overload in their own domain. However, such performance–improving measures are beyond the control of the orchestrator as the composite web service typically consists of web services running in different administrative domains.

We derive next a queueing model of a composition of web services, including an orchestrating web service see Figure 3.1. The queueing model forms the mathematical foundation for our admission control rules.

Suppose that the composite web service consists of web services from the set $\mathcal{W} =$

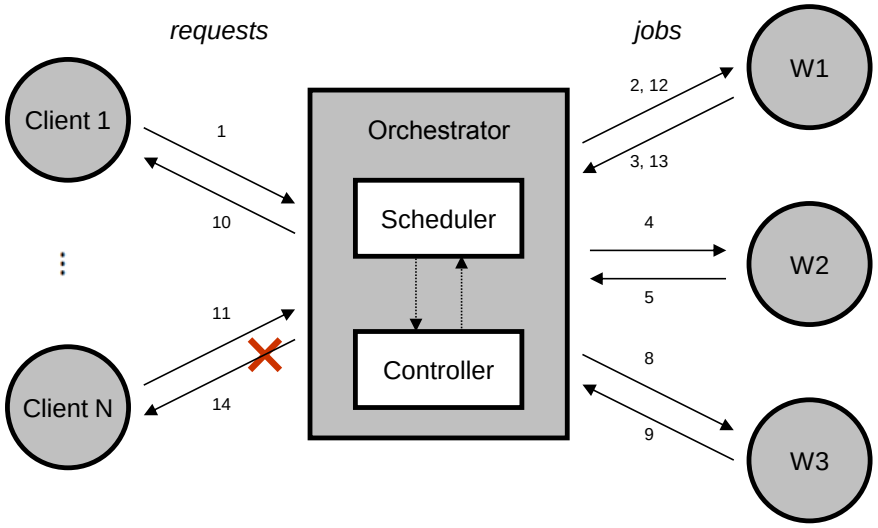


Figure 3.1: Jobs for client requests are routed through a network of web services ($W_1$, $W_2$ and $W_3$) by an orchestrator.

$\{W_1, W_2, \ldots, W_N\}$. In general, the $W_j \in \mathcal{W},\ j = 1, 2, \ldots, N$ may be composite web services themselves. The incoming client requests at the orchestrator are composed of tasks (jobs) to be sequentially executed by a composition of web services from the set $\mathcal{W}$. Thus, each task within the request is served by a single web service. Since the orchestrator may control different composite web services offered by the same provider, the order in which jobs are executed may differ per client request. The orchestrator tracks task execution on a per request basis.

In practice, web services serve jobs using threading, which could be modeled using a round–robin service discipline in which jobs are served for a small period of time ($\delta \to 0$) and are then preempted and returned to the back of the queue. Since $\delta \to 0$, assuming there are $n$ jobs with the same service rate $\mu_w$, the per job service rate is $\mu_w/n$. To simplify analysis, this process is modeled as an (egalitarian) *processor sharing* service discipline.

The service time distribution of web service $W_j,\ j = 1, 2, \ldots, N$ is assumed to be exponential with parameter $\mu_j$. Jobs arrive at web service $W_j$ with arrival rate $\lambda_j$ and the load of web service $W_j$ is defined as $\rho_j = \lambda_j/\mu_j$.

We define the response time $L_i$ of an incoming client request $i$ as the total time it takes for a request to be served. The sojourn time (i.e. time spent in the system) of task $j$ served by the web service $W_j$ from request $i$ is denoted by $S_{ij}$. We assume that the orchestrator is not a single point of failure, i.e. that it can instantly serve

and process all requests. We also ignore possible delay due to network traffic and orchestrator activity, so it holds that

$$L_i = \sum_{j=1}^{N} S_{ij}. \tag{3.1}$$

A client considers request $i$ as successful when its response time $L_i$ is smaller than given maximum $L_{\max}$. We denote by $c_j$ a maximum number of jobs allowed to be served simultaneously by web service $W_j$. When $c_j$ requests are served, the next request that arrives to be serviced by $W_j$ is denied service by the admission control rules at service itself. The admission control rule for web service $W_j$ can be modeled by the blocking probability $p_{cj}$. Since our objective is to serve as many requests as possible (within $L_{\max}$) in an overload situation, our goal is to find the optimal values of the $c_j$.

To further simplify analysis, we assume that the web services $W_j$ have the same values of $c_j, \lambda_j, p_{cj}$, and $\mu_j$, denoted as $c, \lambda, p_c$ and $\mu$, respectively. We address this optimization problem by modeling the web services $W_j \in \mathcal{W}$ as a M/M/1/c Processor Sharing Queue (PSQ). It is generally known that the blocking probability $p_c$ of the $M/M/1/c$ PSQ equals

$$p_c = \frac{\rho^c}{\sum_{k=0}^{c} \rho^k}, \tag{3.2}$$

and that the expected sojourn time at each of the web services equals

$$\mathbb{E}[S] = \frac{\frac{1}{\mu}}{1 - \rho(1 - p_c)}. \tag{3.3}$$

## 3.2 Related work

The use of admission control for Web Servers has been analysed in, for example [39,96,109]. Sharifian et al. [89] propose an approximation–based load–balancing algorithm with admission control for cluster–based web servers. The algorithm classifies requests based on their service times and track numbers of outstanding requests for each class of each web server node and also based on their resource demands to dynamically estimate the loads of each node. Then the estimated available capacity of each web server is used for load balancing and admission control decisions. The use of Web Admission Control (WAC) to prevent overload for Web Services has been discussed in [94, 107]. In the field of composite web services several contributions have been made focusing on web service scheduling, for instance in [37, 38].

Although some of the solutions could be applied to the problem of admission control for orchestrated services, this has not been specifically analysed in above mentioned admission control schemes. Besides these schemes do not include awareness of the execution state of the workflow in a composition of web services.

## 3.3 Dynamic admission control algorithms

In the following sub–sections, two dynamic admission control algorithms, so–called algorithm $S$ and algorithm $D$ are derived from the model discussed in the previous section.

### 3.3.1 Dynamic Admission Control Algorithm S

The basic underlying principle of this algorithm is that the expected sojourn time $\mathbb{E}[S]$ of a job in a Web service should be less than or equal to the average available time for the jobs within the request. Thus, the problem of serving the client request within $L_{\max}$ is split up in consecutive steps. In each step, a limit on the expected sojourn time is calculated in the following way.

The orchestrator divides the maximum response time $L_{\max}$ over all jobs. At the moment $t^*$ when a request enters the orchestrator, the due date for the next task $j^*$ is calculated. First, the total remaining time for this request, i.e. $L_{\max} - \sum_{j=1}^{j=j^*-1} S_{ij}$, is determined. Then, the remaining time to deadline is divided over all remaining jobs in proportion to their service requirements. Let $D_{ij^*}$ be the due date of job $j^*$ from request $i$, let $J_i$ be the total number of jobs from request $i$, let $t^*$ be the time at which the due date for job $j^*$ is calculated, and let $\nu_{ij}$ denote the expected service time of job $j$ from request $i$. Now the following relation holds:

$$D_{ij^*} = t^* + \left( L_{\max} - \sum_{j=1}^{j=j^*-1} S_{ij} \right) \frac{\nu_{ij^*}}{\sum_{j=j^*}^{J_i} \nu_{ij}}. \tag{3.4}$$

As a result, the remaining time for job $j$ from request $i$ at time $t$ is given by $R_{ij}(t) = D_{ij} - t$. When the total remaining time of a request is less than zero, the request is discarded by the orchestrator and the client is notified. Let $\bar{R}$ denote the average of $R_{ij}(t)$.

Dynamic admission control algorithm $S$ is derived using the following constraint: *the expected sojourn time $\mathbb{E}[S]$ of a job in a web service should be less than or equal to the average available time.* Therefore, our optimization problem is defined as follows:

$$\max_c \left\{ c : \mathbb{E}[S] \leq \bar{R} \right\}. \tag{3.5}$$

In 3.5, both $c$ and $\bar{R}$ are time–dependent, but we omit this to simplify our notation. Computation of $\bar{R}$ is straightforward since due times of all jobs within the composite service are known.

Substituting 3.3 in 3.5 yields:

$$\max_{c} \left\{ c : \frac{\frac{1}{\mu}}{1 - \rho(1 - p_c)} \leq \bar{R} \right\}. \tag{3.6}$$

Substituting 3.2 in 3.6 yields:

$$\max_{c} \left\{ c : c \leq \log_{\rho} \left( 1 + \mu \bar{R}(\rho - 1) \right) \right\} \text{ for } \rho > 1. \tag{3.7}$$

Therefore, the admission control algorithm is now defined as:

*Allow arriving jobs service if* $\rho < 1$ *or* $n \leq \log\left(1 + \mu\bar{R}(\rho - 1)\right)$ *still holds after the new job is allowed service.*

There are two major issues concerning the algorithm $S$. First, in order to compute $c$ the value of $\rho$ is needed and thus the values of $\lambda$ and $\mu$ as well. It is assumed that the service requirement rate $\mu$ is known, but the value of $\lambda$ is not. The arrival process (of a web service) will in reality not be known and thus must be estimated. Therefore, the question arises what is the time period to estimate $\lambda$ and how to estimate this value.

The second issue is that the arrival rate is explicitly used to estimate the value of $c$. Intuitively the number of jobs, which can be simultaneously served, does not depend on the number of jobs which arrive at the system. The web service is capable of simultaneously serving $c$ jobs. The blocking probability adjusts for this fact, but further investigation of this issue is required.

In the next sub–section, an alternative dynamic admission control rule is derived, in which the arrival rate $\lambda$ (and hence $\rho$) is not used to determine the maximum value of the number of jobs allowed.

### 3.3.2 Dynamic Admission Control Algorithm D

The goal of algorithm $D$ is to implement an admission control rule that does not require the knowledge of the arrival rate $\lambda$. This algorithm is based on the relaxed constraint that only the jobs of "average" size have to be completed on time. Although in practice jobs may enter the system or depart from the system while such an "average" job is served, we assume that the number of jobs in the system remains the same. Under these conditions/assumptions we investigate whether effective admission control is possible.

When the number of jobs $n$ in the queue is assumed to be constant, the expected sojourn time for a job equals $\frac{n}{\mu}$. When all jobs must be served before their due dates the problem is defined as follows:

$$\max_{c} \left\{ c : \mathbb{E}[S] \leq R_{ij}, \text{for all jobs in service} \right\}. \tag{3.8}$$

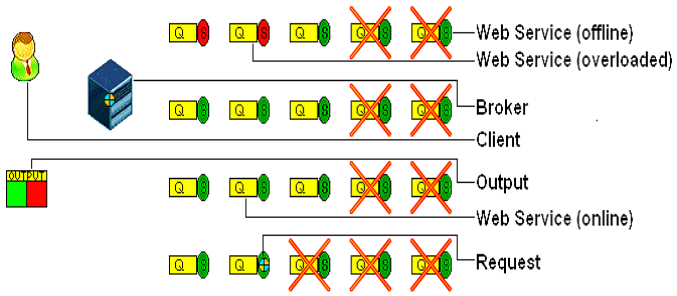


Figure 3.2: Overview of the simulation model.

In our case $\mathbb{E}[S]$ equals $\frac{c}{\mu}$, and $R_{ij}$ is replaced by $\bar{R}$, where $\bar{R}$ determines the average remaining available service time for all jobs in service. These relaxations lead to the following optimization problem:

$$\max_{c} \left\{ c : \frac{c}{\mu} \leq \bar{R} \right\}. \tag{3.9}$$

The solution of this trivial problem yields $c = \mu \cdot \bar{R}$. Hence we define the more practical admission control algorithm $D$ as follows:

*Allow arriving jobs service when for number of jobs in service $n$, inequality $n \leq \mu \cdot \bar{R}$ still holds after the arriving job is allowed service.*

Note that for the calculation of the admission control parameter $c$, the arrival rate (and thus $\rho$) is not needed, which is the major advantage from a practical point of view compared to algorithm $S$.

## 3.4 Numerical experiments

A discrete–event simulation model is constructed to evaluate the proposed admission control rules. The model is implemented using the software package eM–Plant [91]. The simulation model basically consists of four components as illustrated in Figure 3.2. Component 'Client' generates new requests according to a Poisson distribution with rate $\lambda$. Requests are dispatched by component 'Broker', that offers different composite services to its clients. Once a request has been generated a composite service that serves it is randomly assigned. Composite service is described by its workflow that indicates which web services are invoked in order to serve the generated request. Each web service is an instance of component 'WS'. The completed or denied requests arrive at component 'Output', where relevant data is collected. Before a task is executed by one of the web services in the selected composition, the web service checks whether this job is allowed or denied service. In case admission

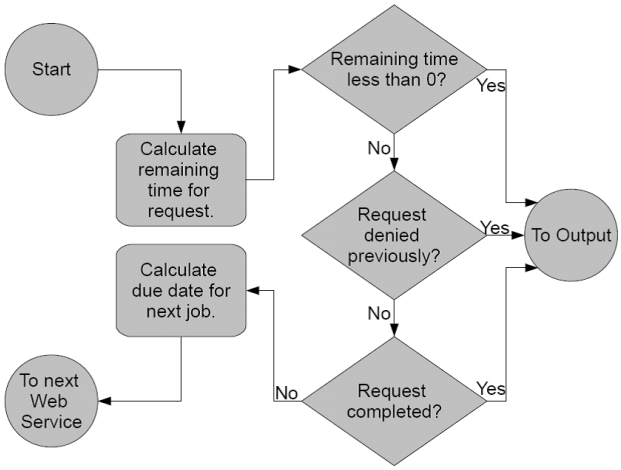


Figure 3.3: Flowchart of the orchestrator in case of admission control.

control is not used, all incoming jobs are allowed. When admission control is used, the web service uses an admission control rule to decide whether the incoming job may be served or not. Figure 3.3 illustrates the flowchart of the orchestrator in case of admission control. When a new request comes in, the orchestrator determines whether the response time of this request has already reached deadline, i.e. whether the remaining time to serve the request is less than zero. If the deadline is reached, the request is denied service and sent to the output component. It may happen that the request has been allowed by the orchestrator, but still the web service itself can not serve the request. That is why, even when the remaining time is greater than zero, the orchestrator determines whether the request has been denied service by a single web service used for the composition. If so, the request is also sent to the output component. If neither the response time deadline has been reached nor the request has been denied service previously, the next web service in a composition needed to complete the request is determined. The orchestrator calculates the due time for the next job, and then invokes the determined web service. For this calculation the total remaining time for the request is divided over all remaining jobs in proportion to their service requirements. When all jobs in the request are served, the response is sent to the output component as well. Two simulation scenarios (ordered and with irregular order) were designed to be used to compare the proposed admission control rules:

- *Ordered simulation scenario:* The web services are placed in a specific order i.e. if web service $X$ is invoked before web service $Y$ for one composite service, the precedence in execution would be the same for every composite service offered by the orchestrator that uses both $X$ and $Y$.

Table 3.1: The workflows of composite services in ordered simulation scenario.

| Composite service | Workflow | Percentage of requests |
|---|---|---|
| 1 | $W_1 \to W_2 \to W_3 \to W_7 \to W_8 \to W_9 \to W_{11}$ | 5 |
| 2 | $W_1 \to W_2 \to W_3 \to W_8 \to W_9 \to W_{10}$ | 20 |
| 3 | $W_1 \to W_2 \to W_7 \to W_8 \to W_9 \to W_{10}$ | 5 |
| 4 | $W_1 \to W_2 \to W_8 \to W_9 \to W_{10}$ | 5 |
| 5 | $W_1 \to W_4 \to W_8 \to W_9 \to W_{10}$ | 5 |
| 6 | $W_1 \to W_4 \to W_8 \to W_9 \to W_{11}$ | 10 |
| 7 | $W_5 \to W_{10}$ | 40 |
| 8 | $W_1 \to W_6 \to W_8 \to W_9 \to W_{11}$ | 5 |
| 9 | $W_2$ | 3 |
| 10 | $W_4$ | 2 |

- *Simulation scenario with irregular order:* There is no specific execution order of web services. We have chosen the most of composite services offered by the orchestrator make use of two specific web services.

Both scenarios are described by

- the (sequential) workflow of each composite service offered by the orchestrator,
- the distribution of requests over the different composite services,
- the (required) service rates of all web services used for the composition.

The arrival rate $\lambda$ and the deadline $L_{\max}$ are parameters specified within a given scenario. There are two performance indicators for the given admission control rules observed:

- the *total* number of successfully served requests,
- *goodput*, defined as the average number of successfully served requests per second.

A bootstrap period (used to estimate $\lambda$) of 15 minutes is chosen as well as a (single) simulation time of 15 minutes. A total of 15 simulations per scenario have been run.

**Ordered simulation scenario**

A total of eleven web services $W_1, W_2, \ldots, W_{11}$ are used for composition of ten different orchestrated services. The orchestrated services are represented by their respective sequential workflows shown in Table 3.1. This table also presents the distribution of requests over composite services, expressed as the percentage of the total of service requests. The required service rates of all web services ($W_1$ to $W_{11}$) used for the compositions are presented in Table 3.2. The response time deadline for each composite web service has been set to the same value, $L_{\max} = 8$ seconds. Using test runs, we found the system gets overloaded for $\lambda \approx 3s^{-1}$. Without WAC,

Table 3.2: The service rates of web services.

| Web service | $W_1$ | $W_2$ | $W_3$ | $W_4$ | $W_5$ | $W_6$ | $W_7$ | $W_8$ | $W_9$ | $W_{10}$ | $W_{11}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Service rate | 5 | $\frac{10}{3}$ | 5 | 5 | $\frac{10}{3}$ | 5 | 5 | 10 | 10 | $\frac{10}{3}$ | 10 |

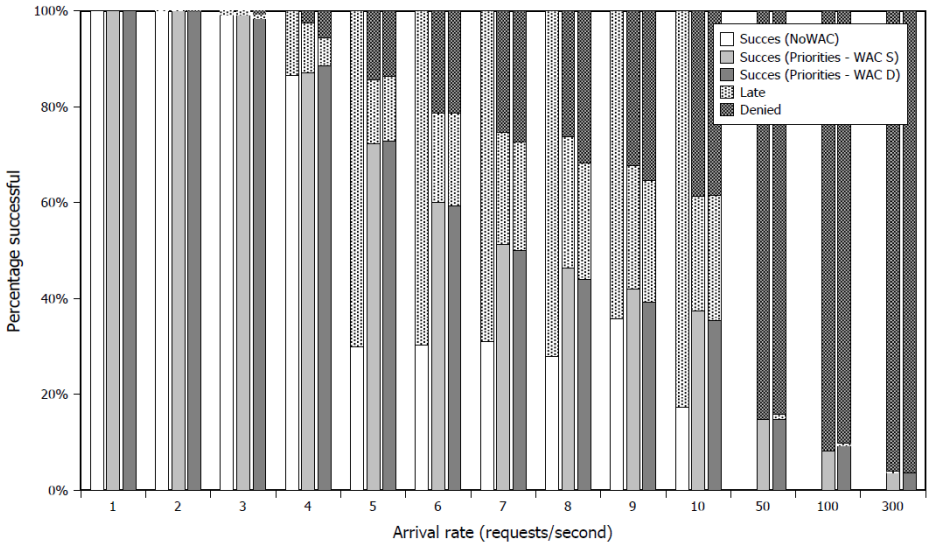


Figure 3.4: Percentage of served requests for ordered simulation scenario.

the simulation runtime rapidly increases as $\lambda$ increases. For $\lambda = 1s^{-1}$ the runtime (without WAC) is about half a minute. For $\lambda = 10s^{-1}$ the runtime has increased to about 45 minutes. To keep simulation runtimes acceptable, the arrival rates $\lambda > 10s^{-1}$ are not investigated for the situation without admission control. It is expected that the total of successfully served request and the goodput both have rapidly decreasing values close to zero when $\lambda > 10s^{-1}$ and without WAC.

Simulation results are summarized in Figures 3.4 and 3.5, including 99.7% individual confidence intervals. Notice that the scale of the horizontal axis changes after $\lambda = 10s^{-1}$. It can be seen that both admission control rules seem to perform equally well and they have a positive effect on goodput. Only at extreme arrival rates, the difference with the theoretical maximum increases. Goodput drops when admission control is not used. However, when admission control is not used, there is a slight increase in goodput between $\lambda = 5s^{-1}$ and $\lambda = 9s^{-1}$. Especially at $\lambda = 9s^{-1}$ the percentage of successful requests is much larger than expected.

Given the confidence intervals it seems unlikely that this is due to the stochastic nature of the experiment results. We call this phenomenon the *arrival paradox* and it is explained by the following example illustrated by Figure 3.6: consider three web services, $W_1$, $W_2$ and $W_3$ each with service rate 5. Requests are first served

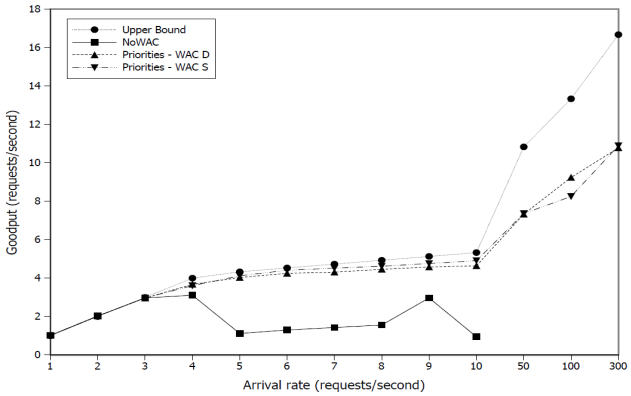


Figure 3.5: Goodput for ordered simulation scenario. The case when no WAC is applied is simulated only for arrival rate(s) $\lambda < 10s^{-1}$ due to the length of simulation, and the fact that goodput would decrease rapidly to the values near zero.

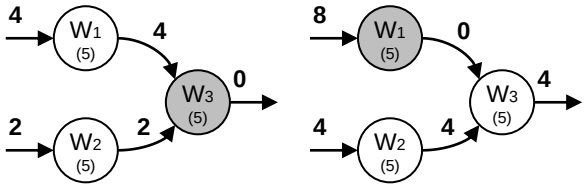


Figure 3.6: Example of the arrival paradox, where web services coloured grey indicate overload.

either by $W_1$ or $W_2$ and consequently by $W_3$. If both $W_1$ and $W_2$ are not overloaded, the goodput from these web services equals the arrival rate of these web services. Therefore the arrival rate at $W_3$ equals the sum of the arrival rates at $W_1$ and $W_2$ and hence $W_3$ is in overload and its goodput drops to zero. When the arrival rates are doubled, one of the web services $W_1$ and $W_2$ may get overloaded. Because admission control is not used, sojourn times at $W_1$ (as illustrated in Figure 3.6) will explode. The composite service requests served by $W_1$ will therefore exceed their respective deadlines. Once served by $W_1$, these requests would be pre–empted by the orchestrator, as the deadline is missed. Therefore the arrival rate at web service $W_3$ decreases due to the higher overall arrival rate and $W_3$ no longer is in overload. Hence, the goodput of $W_3$ increases.

Table 3.3: The workflows of composite services in simulation scenario with irregular order.

| Composite service | Workflow | Percentage of requests |
|---|---|---|
| 1 | $W_8 \rightarrow W_6 \rightarrow W_2$ | 15 |
| 2 | $W_5 \rightarrow W_2 \rightarrow W_3 \rightarrow W_7 \rightarrow W_6 \rightarrow W_1$ | 10 |
| 3 | $W_4 \rightarrow W_3 \rightarrow W_7 \rightarrow W_2 \rightarrow W_6$ | 5 |
| 4 | $W_8 \rightarrow W_7 \rightarrow W_5 \rightarrow W_5 \rightarrow W_9 \rightarrow W_1$ | 10 |
| 5 | $W_7 \rightarrow W_8 \rightarrow W_2 \rightarrow W_5 \rightarrow W_9 \rightarrow W_1 \rightarrow W_6$ | 20 |
| 6 | $W_7 \rightarrow W_4 \rightarrow W_6 \rightarrow W_3 \rightarrow W_5$ | 5 |
| 7 | $W_8 \rightarrow W_9 \rightarrow W_1 \rightarrow W_5$ | 5 |
| 8 | $W_5 \rightarrow W_8 \rightarrow W_3 \rightarrow W_9$ | 10 |
| 9 | $W_6 \rightarrow W_5 \rightarrow W_4$ | 15 |
| 10 | $W_1 \rightarrow W_9 \rightarrow W_8 \rightarrow W_2$ | 5 |

**Simulation scenario with irregular order**

A total of nine web services $W_1, W_2, \ldots, W_9$ are used for composition of ten different orchestrated services. The invocation precedence of web services is different for different composite services, and the most of compositions use either $W_5$ and/or $W_6$. The orchestrated services are represented by their respective sequential workflows shown in Table 3.3. This table also presents the distribution of requests over composite services.

For the ordered simulation scenario it could be argued that some web services would never get overloaded. For simulation scenario with irregular order this cannot be argued. Requests are first served by web services $W_1$, $W_4$, $W_5$, $W_6$, $W_7$ or $W_8$. Any of these web services will get in overload when the arrival rate is high enough. Simulation results are summarized in Figures 3.7 and 3.8. Just as in the previous case, the differences between the two admission control rules seem almost negligible, except for the goodput achieved for relatively high arrival rates. For relatively low arrival rates, $\lambda < 5s^{-1}$, the $D$ rule results with a slightly worse performance than without admission control. In all other cases the admission control rules both perform better than when admission control is not used. The difference between the theoretical maximum for the goodput and the observed goodput is larger compared to ordered simulation scenario, even for small values of $\lambda$. In ordered simulation scenario, the goodput is increasing function, even at high arrival rates. However, in this simulation scenario, the goodput decreases after $\lambda = 12s^{-1}$. This is due to the fact that more different web services used for the compositions would get overloaded for considered simulation scenario.

Table 3.4: The service rates of web services.

| Web service | $W_1$ | $W_2$ | $W_3$ | $W_4$ | $W_5$ | $W_6$ | $W_7$ | $W_8$ | $W_9$ |
|---|---|---|---|---|---|---|---|---|---|
| Service rate | 5 | 5 | 4 | 10 | 4 | 4 | 10 | 5 | 5 |

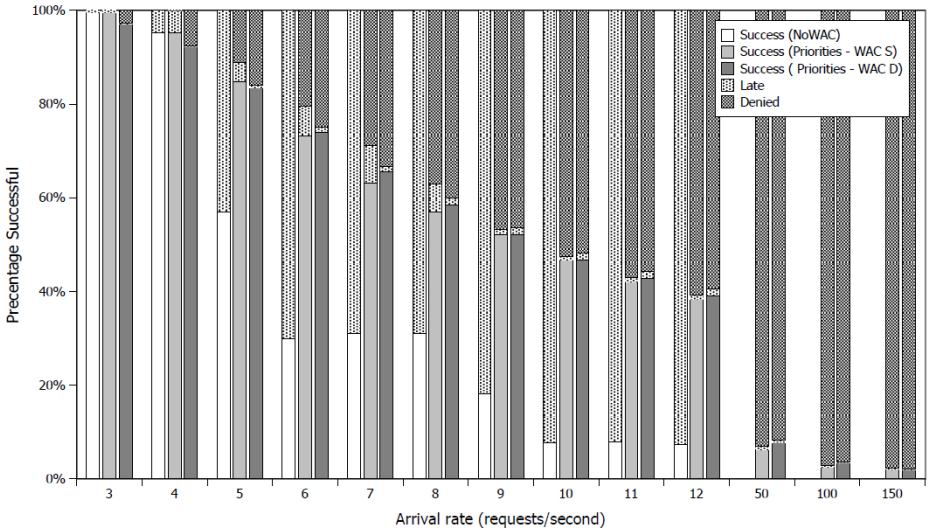


Figure 3.7: Percentage of served requests for simulation scenario with irregular order.

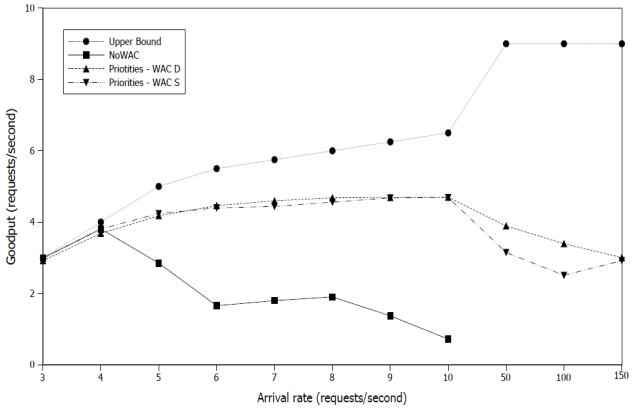


Figure 3.8: Goodput for simulation scenario with irregular order. The case when no WAC is applied is simulated only for arrival rate(s) $\lambda < 10s^{-1}$ due to the length of simulation, and the fact that goodput would decrease rapidly to the values near zero.

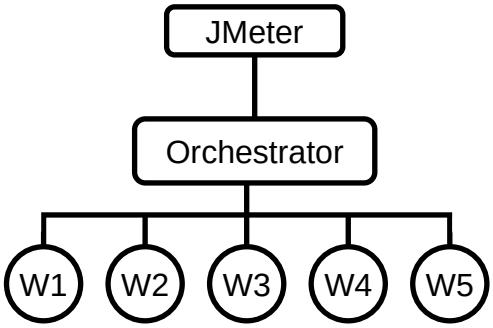


Figure 3.9: System setup for empirical validation of admission control.

## 3.5 Experimental validation

In this Section we describe the empirical experiments used to validate the simulations of admission control rule D. A total of five web services were built and ten different orchestrated services were composed. The goodput of orchestrated services are compared to the simulation results for implemented control rule D.

The functionality of the web services is irrelevant for goodput comparison. In addition, for setting up the tests, it is convenient when the CPU demand of executing a web service can be controlled. We therefore implemented web services that calculate a Fibonacci number and each service gets its own number to calculate. The choice of Fibonacci number influences the CPU consumption of a web service. We compared the experiments and simulations for the ordered simulation scenario with admission rule D (WAC D), as well as for the case when no admission control is applied (NOWAC). A global overview of the experimental setup is given in Figure 3.9. We used the software package JMeter [5] to generate the requests for the composite services. The orchestrator and the individual web services ($W_1$ through $W_5$) are implemented following the design and implementation of the corresponding components in the simulations. All software was written in Java and executed on Tomcat [6] extended with Axis2 [4] for web service functionality.

The orchestrated services and distribution of services' requests are presented in Table 3.1. No values of the service rates of web services are given as all individual web services were configured to calculate the same Fibonacci number. Both the JMeter and the orchestrator run on the system equipped with 2GB RAM and single Pentium IV processors clocked at 3.2GHz. The web services $W_1, \ldots W_5$ run on systems equipped with memory capacity of 0.5GB, 1GB, 1GB, 0.5GB and 0,5GB, respectively, and with Pentium IV processors at 1GHz, 2.4GHz, 2.4Ghz, 1GHz and 1GHz respectively. In each run of JMeter a fixed number of threads (between 1 and 200) were active. Each run used a total time of 30 minutes. An overview of the experimental results is given in Figure 3.10. The empirical and simulation re-

Table 3.5: The workflows of composite services in ordered simulation scenario.

| Composite service | Workflow | Percentage of requests |
|---|---|---|
| 1 | $W_1$ | 10 |
| 2 | $W_4$ | 10 |
| 3 | $W_1 \rightarrow W_2 \rightarrow W_3 \rightarrow W_4 \rightarrow W_5$ | 10 |
| 4 | $W_1 \rightarrow W_2 \rightarrow W_4 \rightarrow W_5$ | 10 |
| 5 | $W_1 \rightarrow W_2$ | 10 |
| 6 | $W_1 \rightarrow W_4 \rightarrow W_5$ | 10 |
| 7 | $W_1 \rightarrow W_3 \rightarrow W_5$ | 10 |
| 8 | $W_1 \rightarrow W_3 \rightarrow W_4$ | 10 |
| 9 | $W_2 \rightarrow W_5$ | 10 |
| 10 | $W_3 \rightarrow W_4$ | 10 |

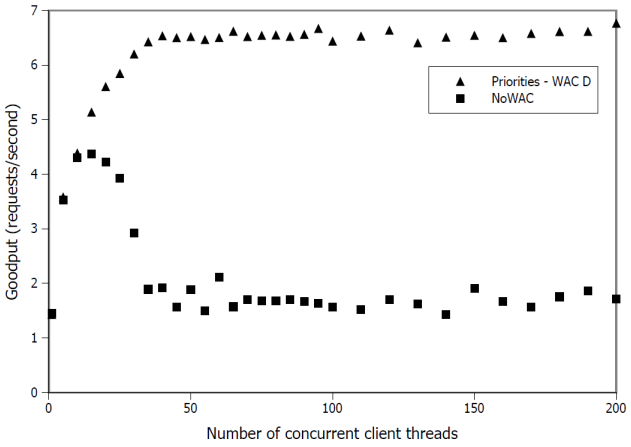


Figure 3.10: Results of the empirical tests.

sults correlate well. Using WAC the overall goodput was noticeably higher than the NOWAC scenario. The NOWAC scenario reaches a maximum goodput when there are a little bit more than 4 requests per second at 15 concurrent threads. The WAC scenario seems to level between 6 to 7 requests per second at 50 concurrent threads.

## 3.6 Conclusions and future work

In this chapter we focused on the use of admission control rules by the orchestrator to prevent the composite web service from becoming generally unavailable in an overload situation. We developed model of the orchestrated web service with admission control in which Web services are modeled using queueing theory (in particular M/M/1/c PSQ) and the admission control rule for web service is modeled by the

blocking probability. The model allows to find the optimal values of the maximum number of requests to be served simultaneously by a web service, with the objective to serve as many requests as possible within specified end–to–end response time. Using this model, two admission control rules ($S$ and $D$) for orchestrated web services were derived.

For a number of different web service compositions, the simulations were conducted with these two rules and a benchmark (in which no admission control rule is used). Based on simulation results, we conclude that in most situations both admission control rules $S$ and $D$ resulted in a considerably higher objective value (measured in goodput) than the benchmark. While the difference is small, rule $S$ does perform better than rule $D$. However, it can be observed that the results are dependent on the case, the workflow and interaction patterns of the used web services.

The admission control rule $S$ requires the estimation of the arrival rate of composite service requests, and it is therefore challenging to implement this rule in actual web services. Given this constraint, and the fact that simulations showed small difference between the two rules, we performed an empirical evaluation of rule $D$. The evaluation results confirmed the outcome of the simulations.

To achieve further improvements, the empirical experiments should be scaled up to evaluate a broader range of different and larger service oriented infrastructures. Such experiments would be primarily focused on obtaining the optimal goodput as well as incorporating other objectives (e.g. costs) in the admission control rules.

CHAPTER 4

# Performance Evaluation of QoS–aware Run–Time Web Service Selection Strategies in Service Oriented Architecture

In the previous chapter we developed relatively simple admission control rules in order to keep end–to–end response time and availability at agreed QoS levels. The considered service composition assumed there is a single service implementing any given task within the workflow. In practice, there may be many different, functionally equivalent services that implement a single task.

In this chapter we consider dynamic, run–time service selection as the QoS control mechanism. This means that the orchestrator representing the composite service provider may apply one of possible service selection strategies per single request (i.e. request dispatching) in order to achieve better QoS control (e.g. response time). We particularly focus on the achievable performance gain of dynamic selection of services implementing a single task. The selection is made from a set of pre–selected concrete services for the task considered, and we evaluate different selection strategies.

The performance improvements of service selection may be impacted by practical conditions such as background traffic or delayed (concrete) service state information. Composite service provider assures certain QoS levels to different clients. When considering guarantees for particular client, CSP has to take into account requests originating from other clients (background traffic) as well. The system state information like the number of requests waiting to be processed by particular concrete service, may be delayed significantly compared to the arrival rate of observed requests. We therefore also analyse the impact of these conditions to the aforementioned performance improvements.

The main contributions of this chapter are as follows:

- Quantification of the achievable performance gain versus the number $K$ of pre–selected concrete services by a fair comparison with respect to the *base case* of static web service selection. The base case corresponds to the situation when there is only one pre–selected concrete service, i.e. when $K = 1$. We establish that even for relatively small values of $K$, significant response–time reductions are feasible.

- Quantification of the achievable gain in terms of reduced response time when background traffic is present at the pre–selected concrete services for different dispatching strategies. We show that the response–time performance of Join the Shortest Queue (JSQ) is quite robust with respect to the presence of background traffic. An insightful approximate formula for the response time under the JSQ dispatching strategy is derived for cases where the background traffic is dominant.
- Quantification of the achievable gain in terms of reduced response time in case of delayed state information. A stateless dispatching algorithm such as Round Robin (RR) always improves upon the base case ($K = 1$). However, stateful dispatching algorithms such as JSQ may result in increased response time compared to the base case. We therefore analyse performance gain of a combination of RR and JSQ strategies, referred to as JSQ–RR. We show that JSQ–RR performs better than RR and hence the base case when delayed state information is used for service selection, even when this delay tends to infinity.

The remainder of this chapter is organized as follows. First, in Section 4.1, we describe the performance model and explain the underlying assumptions that capture the essential system characteristics needed for our study. Next, in Section 4.2, we discuss literature related to our work. In Section 4.3, our simulation results and results obtained by analytical modelling are presented, and we discuss and explain the observations. Finally, in Section 4.4, we draw conclusions and give suggestions for further research.

This chapter is based on paper [121].

## 4.1 Problem description and modelling

We consider one abstract service with $K$ concrete service implementations as given in Figure 4.1. There are two classes of incoming service requests:

- *Foreground service requests* are received by the dispatcher, which decides at runtime to which of the $K$ service instances a particular request is assigned for getting the required service. The foreground requests arrive to the dispatcher according to a Poisson process [65] with rate $\Lambda$ and have exponentially distributed service requirements with mean $\frac{1}{\mu}$. The rate at which foreground traffic requests are offered (by the dispatcher) to service $i$ is denoted by $\lambda_{\mathrm{FTi}},\ i = 1, 2, \ldots, K$.
- *Background service requests* arrive at service instance $i$ according to a Poisson process with rate $\lambda_i,\ i = 1, 2, \ldots, K$, respectively. The background service requests are exponentially distributed with mean $\frac{1}{\mu_i},\ i = 1, \ldots, K$. The background traffic arrival processes are independent from each other and are also independent from the foreground arrival process.

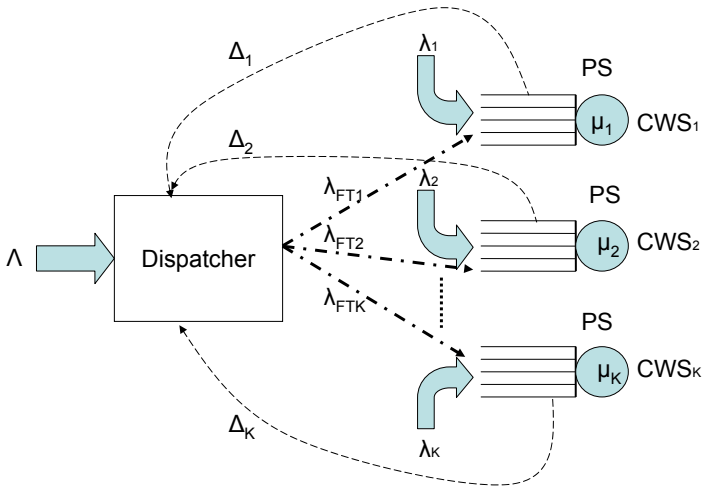


Figure 4.1: Performance model for the case of a single abstract service with $K$ different implementations (concrete services).

Request scheduling at each service instance is modelled by a PS infinite–capacity single server queue. Served requests leave the system.

Obviously, the achievable performance gain of dynamic service selection compared to a pure static approach depends heavily on the nature of the workload fluctuations at the different service instances. It is clear that in the case of slowly and independently varying loads (e.g., due to fluctuations in the service demand over the day) high performance gain can relatively easy be achieved. However, in such cases the runtime character of dynamic service selection may be an "overkill" and the performance gain could largely also be obtained by less flexible service selection approaches. Therefore, this chapter focuses on exploiting workload fluctuations at the services instances that occur at relatively small time scales mainly caused by the random behaviour of individuals in a large population of potential users. In that perspective, and to keep the parameter space manageable, we will assume that the model is symmetric, i.e., $\lambda_i = \lambda_{\mathrm{BT}},\ \mu_i = \mu,\ i = 1, 2, \ldots, K$. The utilization per service $i$ ($i = 1, 2, \ldots, K$ is then defined as $\rho_{\mathrm{tot}} = \frac{\lambda_{\mathrm{tot}}}{\mu}$, where $\lambda_{\mathrm{tot}}$ is the rate of the aggregated (foreground and background) traffic, i.e., $\lambda_{\mathrm{tot}} = \lambda_{\mathrm{FTi}} + \lambda_i = \lambda + \lambda_{\mathrm{BT}}$. The stability condition per service requires that the expected number of requests per service remains finite, i.e., $\rho_{\mathrm{tot}} < 1$.

Ignoring possible delays due to the queueing and processing at the dispatcher, as well as network delay, arriving service requests are instantaneously forwarded to one of the $K$ service instances according to the dispatching strategy. Various strategies can be used for selection of one of the $K$ service instances upon arrival of a new request. The dispatching strategies could be roughly divided into two categories, namely *stateless* and *stateful*. Decision making is independent of the system

state information for the stateless strategies. Conversely, decision making takes into account (stale) system state information for stateful strategies.

The delay in obtaining the information per service instance is represented by parameter $\Delta_i,\ i = 1, 2, \ldots, K$. In this chapter we have adopted the case when system state information (queue length, response time, etc.) is collected periodically with the same period $\Delta > 0$. This information gathering may require sending separate requests ("probes") by the dispatcher to all of the $K$ web services, and collecting information in such a manner introduces an overhead to the system, which influences system performance. This issue as well as using other ways to collect system state information are beyond the scope of the work presented here. The update period $\Delta$ has been related to the intensity of the aggregated traffic as $\Delta = D \cdot \frac{1}{\lambda_{tot}}$. where $D$ is integer. All dispatching decisions between time instances $t_i = i \cdot \Delta$ and $t_{i+1} = (i + 1) \cdot \Delta,\ i = 0, 1, 2, \ldots$ are made based on the system state information obtained at $t_i$.

The dispatching strategies considered are Bernoulli (BL), RR, JSQ and a combination of the latter two, JSQ–RR. BL and RR are typical examples of stateless strategies while JSQ and JSQ–RR are examples of stateful dispatching strategies. In case of the Bernoulli strategy, the requests are randomly distributed over the queues, i.e., a newly arriving request is assigned to queue $i,\ i = 1, \ldots, K$, with probability $\frac{1}{K}$. This case is used as representation of the performance for the static SOA service selection. For RR, the $k$–th request is assigned to queue $(k \bmod K) + 1$. In JSQ, the request is assigned to the queue with the smallest number of requests waiting to be served. Ties are resolved by randomly assigning the request to one of the shortest queues. In case when system state information delay is present in the system, an additional stateful dispatching strategy could be defined, namely JSQ–RR. For JSQ–RR, once the actual system information is obtained, the queues are sorted in non–descending order by the queue lengths. Any request coming to the dispatcher between two state updates is then assigned following the RR scheme, i.e., the first request is assigned to the queue with smallest queue length, the second request is assigned to the queue with smallest queue length from the remaining queues, etc.

## 4.2 Related work

In this section, we give a short overview of papers related to different aspects (e.g., web service selection, composition, performance) of the run–time web service selection in SOA. However, each of these papers treats only a (different) subset of issues relating to run–time web service selection. The analysis of potential performance improvements of different dispatching strategies, based on PS modelling of the request scheduling at web service(s) within SOA, which includes impact of stale system state information and/or background traffic is, to our best knowledge, non–existent.

### 4.2.1 Web service selection and composition

In [63], an overview of common misconceptions about SOA is given. Among others, the issue of dynamic selection of web services is identified, and it is indicated that current SOA solutions lack advanced automatic discovery and composition of web services at runtime.

A lot of attention within SOA community has been dedicated to static QoS–aware composition problem (see e.g., [21, 81, 111] and references therein). The problem of static QoS–aware composition is known to be NP–hard [113] where two service selection approaches for constructing composite services have been proposed: local optimization and global planning.

In [76], several architectures and their respective models that assist in dynamic invocation of web services are discussed. These models allow the *client* to dynamically select the current best web service, based on certain non–functional criteria (availability, reliability, and estimated response time). These clients gather runtime web service information, evaluate the performance of the previously used web services, and may share this information with other clients. The selection decision is let to the clients, which contain intelligent agents and therefore the complexity of the clients increases. The inherent problem is that different clients may decide to use the same web service, which would eventually result in worsened performance e.g., due to the overload of the targeted web service.

The framework proposed in [64] enables quality–driven web service selection, based upon evaluation of the QoS of a vast number of web services. The fair computation and enforcing of QoS of web services takes place when making the web service selection. In order to provide fair computation the feedback from clients is gathered.

### 4.2.2 Performance of dispatching strategies

Performance of dispatching strategies in multi–server systems has been a topic that received a lot of attention within the queueing theory research community. Specifically, a lot of work has been done for systems with First Come First Served (FCFS) scheduling at the queues, [19, 29]. In the most of the papers written for JSQ/FCFS, explicit results for response times are given only for the case $K = 2$ servers, an exponential job size distribution and the mean response time metric [43]. The performance of the JSQ/FCFS strategy for $K > 2$ servers has been analysed in [71] where the approximation of the mean response time for $K$ homogeneous servers is given. In [72], an extension to this approximation has been given, however, the approximation is less accurate as the requests' size variability increases.

Opposite to the JSQ/FCFS systems, JSQ/PS systems have not received so much attention. The notable exceptions are [18], and, more recently, [48] where approximate analysis of JSQ in the PS server farm model for general job size distributions is presented. The queue length of each queue in the system is approximated by a

one–dimensional Markov chain, and based on this approximation the distribution of the queue length at each queue is determined. In [3], the authors investigate optimal dispatching strategies for a multi–class multi–server PS systems with a Poisson input stream, heterogeneous service rates, and a server–dependent holding cost per unit time.

### 4.2.3 Performance of dispatching strategies with stale system information or background traffic

In [69], the problem of dispatching with stale system status information (server load) is analysed in case of FCFS. Servers' status information is periodically updated and three strategies are compared: random selection, selection of the server with the least load (based on the stale system information), and random selection of a small subset of servers and then selecting the least loaded of the chosen servers (based on up–to–date information about their loads). It is shown that the latter strategy mostly outperforms the other ones, even for a small randomly chosen subset of e.g., two servers, while the overhead (due to processing and information retrieval) remains limited.

In [24], the authors present a strategy that routes the jobs to the server with expected shortest FCFS queue. The decisions are made based on stale information and elapsed time since the last state update. This strategy works well, but does not always minimise the average response time. In [50], a dispatching policy based on splitting foreground traffic according to a predefined rule described by a certain parameter vector is analysed while background traffic is modelled as independent Poisson processes with different rates. Due to the assumptions made each of the $N$ servers in isolation can be represented as a two–class M/G/1 PS queue. The approximation of the response times is deduced for the case of light foreground traffic and an optimal parameter vector is found.

## 4.3 Numerical experiments

In this section, we present and discuss simulation results for the run–time service selection strategies described in Section 4.1 in order to investigate their performance potential. For some special scenarios we also present numerical results obtained from analytical modelling.

The simulations were performed using the simulation tool implemented in Java programming language, and using the Java library for stochastic simulation (SSJ) [32]. In order to make the simulations less sensitive to the startup transient, the number of foreground traffic arrivals per simulation has been set to at least $0.5 \cdot 10^6$. Besides, in order to improve the accuracy, we have trimmed simulation results for certain number of foreground traffic arrivals at the end of the arrival process.

We have considered four main categories of simulation scenarios:

- *Baseline scenarios* – these simulations were performed for the system without background traffic and with up-to-date system state information. The simulation results are given in subsection 4.3.1.
- *Scenarios with stale system state information* – these simulations were performed for the system without background traffic in which system state information is only periodically updated i.e., the dispatching process does not (always) use up-to-date information. The results are presented in subsection 4.3.2.
- *Scenarios with background traffic* – these simulations were performed for the system with up-to-date system state information and different intensities of background traffic. In addition to the simulations we derived an analytical approach to study the performance for these scenarios. The results are presented in subsection 4.3.3.
- *Scenarios with background traffic and stale system state information.* The simulation results are presented in subsection 4.3.4.

### 4.3.1 Baseline scenarios

The goal of these simulation scenarios was to establish the performance results in case when there is no background traffic and system state information is up-to-date at the dispatcher.

In Figure 4.2, we show mean response times for different dispatching strategies (JSQ, RR, BL) as a function of the number of concrete services, $K$ and for different values of utilization per service, $\rho_{\text{tot}}$. The utilization per service is kept constant in order to have a fair assessment of the impact of $K$; otherwise, an increase of $K$ would simply be interpreted as capacity add-on to the system. Since JSQ–RR is identical to JSQ when up-to-date system state information is available at the dispatcher, the results for JSQ–RR are not shown. For $\rho_{\text{tot}} = 0.8$, the mean response time for JSQ strategy with $K = 4$ services is around 66% of mean response time for the same strategy when $K = 2$. Similarly, in case of JSQ with $K = 8$ and $K = 16$ services, response times are 49% and 40% of the response time for $K = 2$, respectively. In case when one of the $K$ services becomes unavailable, the performance of the system (response time) does not deteriorate dramatically, as long as the utilization per queue remains (approximately) the same. The utilization per queue can be kept the same when, e.g., $K + 1$ services are pre–selected, of which given (fixed choice) $K$ services are used for dispatching. The remaining $(K+1$–th) service is placed "on hold" and when one of the chosen $K$ services becomes unavailable, it is immediately replaced.

Figure 4.3 shows relative comparisons between JSQ and BL (with BL as the baseline) and JSQ and RR strategies (with RR as the baseline), respectively. Stateful strategy (JSQ) is superior to either of the stateless strategies (BL, RR), which confirms that

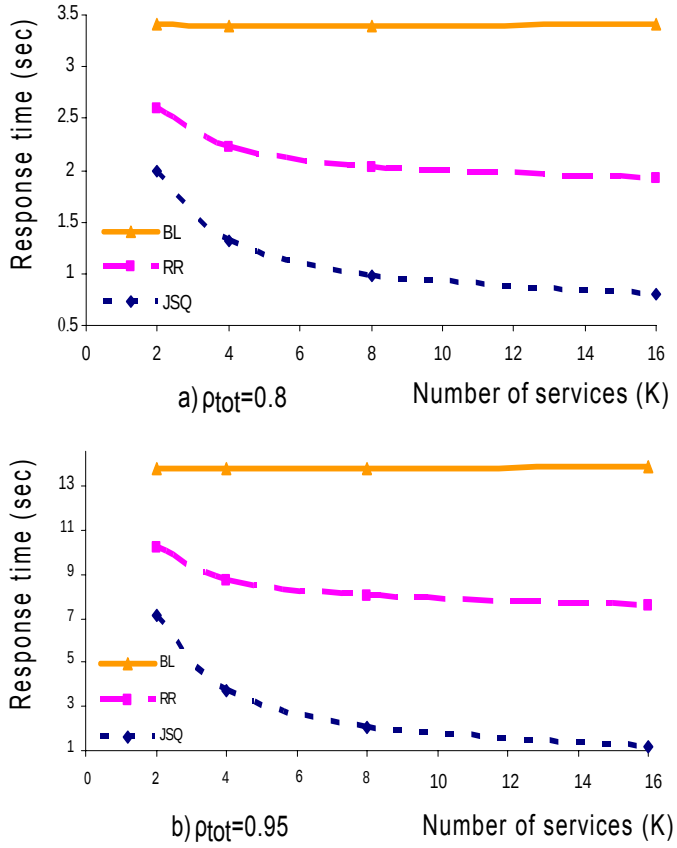


Figure 4.2: Comparison of mean response times for JSQ, RR and BL strategies for different number of services $K$ and different values of $\rho_{\text{tot}}$. There is no system state information delay and only foreground traffic is present in the system.

more (and accurate) state information made available to the dispatcher leads to better decision making.

The potential performance improvements in the first case range from 28% to 46% for $K = 2$, depending upon the utilization per queue, $\rho_{\text{tot}}$ and are in the range from 49% to 86% for $K = 16$. What is also of interest is when do the gradient of the performance improvement is highest, taking into account the increase of the number of services. From Figure 4.3 we see that this is the case when the number of services increases from 2 to 4. The gradient of the gains is (significantly) smaller when the number of services increases from 4 to 8 or 8 to 16, respectively.

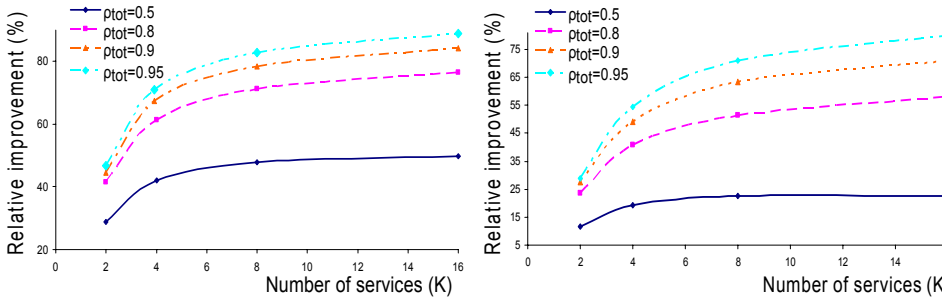


Figure 4.3: Relative comparison of mean response times between JSQ and BL (left) and JSQ and RR (right) strategies for different number of services $K$ and different values of $\rho_{\text{tot}}$. The system state information is up-to-date and only foreground traffic is present in the system.

Based on these simulation results, we can draw the following conclusions:

- Large performance improvements compared to the static service selection are possible with relatively small values of $K$.
- The largest relative improvements of response time for number of services, $K > 1$, are obtained when we increase the number of services that are used from 2 to 4.

### 4.3.2 Scenarios with stale system state information

The goal of these simulations was to analyse the impact of the stale system state information to the response time of the system for different dispatching strategies. No background traffic has been assumed. The simulations were performed only for the stateful strategies, i.e., JSQ (see Figure 4.4) and JSQ–RR, see Figure 4.5. The response times for stateless strategies, RR and BL, are not affected by (stale) system state information, and are shown for comparison as well.

From Figure 4.4 we see that, for relatively small values of parameter $D$ that determines the update interval, JSQ still performs better than RR or BL dispatching strategy. However, as expected, when parameter $D$ increases performance of JSQ deteriorates e.g., for $D = 20$ response time for JSQ is worse than either RR or BL for almost complete range of parameter $\rho_{\text{tot}}$. When $D \to \infty$, the system state information is obtained just once, and then all arrivals are "blindly" assigned to the queue which had the smallest queue length when system state information was obtained. In that case, the service composition in this case is static, and the system model reduces to a M/M/1/PS queue with arrival rate $\Lambda$ and mean service time $\mu$.

We have also investigated the behaviour of the JSQ–RR strategy for systems with

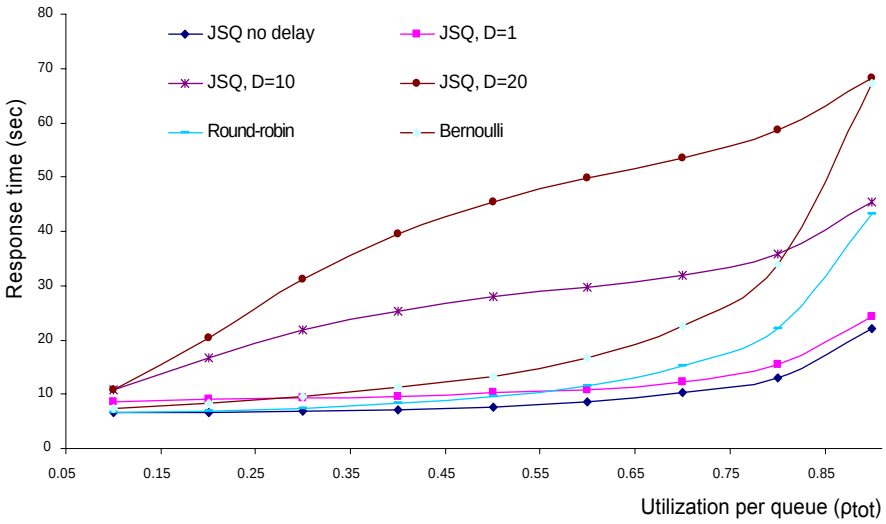


Figure 4.4: Comparison of mean response times for the following dispatching strategies: BL, RR, JSQ with up-to-date system state information, and JSQ when system state information (queue lengths) is obtained with period $\Delta = D \cdot \frac{1}{\lambda_{\text{tot}}}$, where $D \in \{1, 10, 20\}$. Background traffic is not present and the number of services $K = 4$.

stale state information. Figure 4.5 shows that, as expected, JSQ–RR strategy is less sensitive to stale information than "blind" JSQ strategy. For example, when $D = 10$ and $\rho_{\text{tot}} = 0.7$, response times for BL, RR, JSQ and JSQ–RR are 2250 ms, 1515 ms, 3200 ms (!) and 1350 ms, respectively. For comparison, the response time for JSQ without stale information and the same $\rho_{\text{tot}}$ is approximately 1020 ms.

Based on the simulations which results are presented at Figure 4.4 and Figure 4.5 we can draw the following conclusions:

- When $D \to 0$, JSQ–RR is identical to JSQ and when $D \to \infty$, JSQ–RR is identical to the common RR strategy.
- With respect to the response time, the JSQ–RR strategy is never worse than RR, regardless of the delay within the system. This makes JSQ–RR appealing strategy for systems with delay without background traffic.

### 4.3.3 Scenarios with background traffic

In the previous simulation scenarios we have assumed that the concrete services were used by the foreground traffic clients only. In what follows we look into the situation when background traffic is present as well, and the dispatcher has up-to-date system state information.

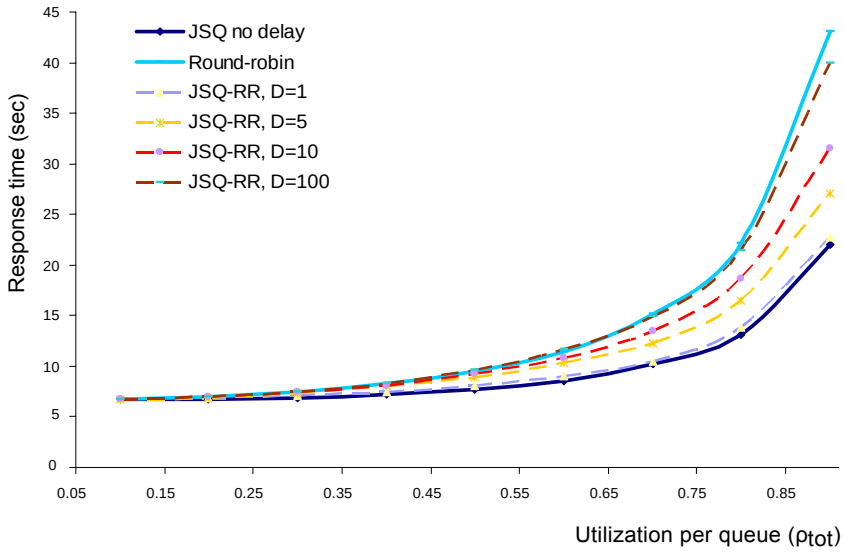


Figure 4.5: Comparison of mean response times for the following dispatching strategies: RR, JSQ with up-to-date system state information and JSQ–RR when system state information (queue lengths) is obtained with period $\Delta = D \cdot \frac{1}{\lambda_{\text{tot}}}$, where $D \in \{1, 5, 10, 100\}$. Background traffic is not present and the number of services, $K = 4$.

Our simulations and analysis are directed to answering the question of the impact of the background traffic to the response times. The simulations results are shown in Figure 4.6 for $K = 4$ services and BL, RR, and JSQ strategies. Since the system state information is assumed to be instantaneously available, JSQ–RR is identical to JSQ, and therefore not shown. We have recorded the response times of the foreground requests only. For given utilization per queue $\rho_{\text{tot}}$, and dispatching strategy, foreground traffic percentage of $\rho_{\text{tot}}$ has been varied from as little as 10% (i.e., 90% background traffic) to 99% (i.e., 1% background traffic). Apart from the case of the BL dispatching strategy when response times are constant, as expected, from Figure 4.6 it follows:

- In case of the RR strategy response times decrease as the percentage of foreground traffic increases. It seems that response times dependency from the given percentage is linear.
- In case of the JSQ strategy response times show linear non-increasing dependency from the given percentage of foreground traffic. The decrease of the response time is limited by 15% for the considered cases. It seems that the JSQ strategy is rather insensitive to the background traffic.

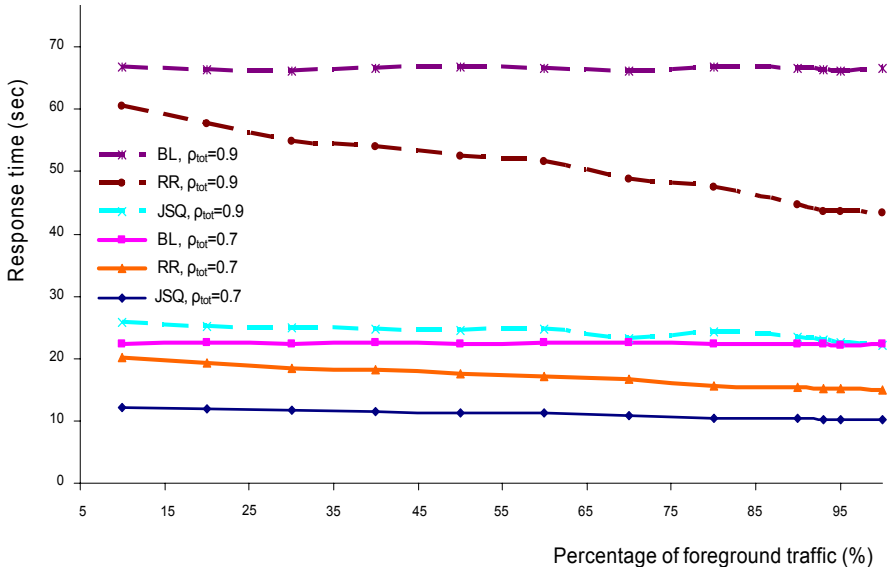


Figure 4.6: Response times for BL, RR and JSQ strategies for scenario with background traffic and no information delay. Utilization per queue $\rho_{\text{tot}}$ is 0.7 or 0.9, and number of services $K = 4$.

The intuitive explanation for the decreasing nature of response times in case of RR and JSQ strategies may be given as the following – the response time in the case of these two strategies is biggest for the smallest percentage of foreground traffic, due to the fact that only foreground traffic is "intelligently" assigned to one of the queues.

**Response time for JSQ with low foreground traffic load**

Let us now consider the situation where the foreground traffic constitutes only a small percentage of the total traffic. We will analyse the mean response time of a tagged foreground traffic arrival. According to the JSQ policy this arrival will be dispatched to the queue (out of $K$ queues) with the smallest length. Since the foreground traffic is negligible and the background traffic arrival processes are mutually i.i.d., the random processes representing the queue lengths are also independent from each other and behave as the queue length of an M/M/1 PS queueing model with load $\rho_{\text{tot}}$. The queue length distribution for this model is geometric with parameter $\rho_{\text{tot}}$ [92]. As shown in [30, 60], the probability distribution of a random variable that represents the minimum of $K$ mutually i.i.d. geometric random variables with parameter $\rho_{\text{tot}}$ is also geometric, with parameter $\rho_{\text{tot}}^K$. Hence, the probability that the queue selected for the tagged foreground job contains $n$ (background) jobs is given by:

$$\text{Prob}\{n \,\text{jobs in selected queue}\} = \left(1 - \rho_{\text{tot}}^K\right)\left(\rho_{\text{tot}}^K\right)^n,\ n = 0, 1, \ldots$$

Once the tagged arrival is placed to a particular queue, that queue further behaves as an "ordinary" M/M/1 PS queue with utilization $\rho_{\text{tot}} = \frac{\lambda_{\text{BT}}}{\mu}$, as the foreground traffic is negligibly small.

Now, let us denote by $X_n(\tau)$ the random variable whose distribution is that of the "delay" experienced by the tagged arrival if it would have service requirement $\tau$ and arrives when there are $n$ background jobs in the queue. The total time spent in the system for the tagged arrival (i.e., response time) is then $X_n(\tau) + \tau$.

From the detailed analysis of the M/M/1 PS queue in [30], it follows that (cf. Eq. (33) in [30]):

$$\mathbb{E}[X_n(\tau)] = \frac{\rho_{\text{tot}}\tau}{1-\rho_{\text{tot}}} + [n(1-\rho_{\text{tot}}) - \rho_{\text{tot}}] \cdot \frac{1 - e^{-(1-\rho_{\text{tot}})\mu\tau}}{\mu(1-\rho_{\text{tot}})^2},\ \tau \geq 0,\ n = 0, 1, \ldots.$$

Since the r.v. $X_n(\tau)$ is conditioned by $n$, $\mathbb{E}[X(\tau)]$ is given by the following equation:

$$\mathbb{E}[X(\tau)] = \sum_{n=0}^{\infty} \left(1 - \rho_{\text{tot}}^K\right) \left(\rho_{\text{tot}}^K\right)^n \cdot \mathbb{E}[X_n(\tau)],\ \tau \geq 0,$$

which leads to

$$\mathbb{E}[X(\tau)] = \frac{\rho_{\text{tot}}\tau}{1-\rho_{\text{tot}}} + \frac{\rho_{\text{tot}}^K - \rho_{\text{tot}}}{1-\rho_{\text{tot}}^K} \cdot \frac{1 - e^{-(1-\rho_{\text{tot}})\mu\tau}}{\mu(1-\rho_{\text{tot}})^2},\ \tau \geq 0.$$

The overall mean response time for the tagged arrival is given by

$$\text{RT} = \frac{1}{\mu} + \int_0^{\infty} \mathbb{E}[X(\tau)] d(1 - e^{-\mu\tau}),$$

which finally gives

$$\text{RT} = \frac{1}{\mu} + \frac{\rho_{\text{tot}}}{\mu(1-\rho_{\text{tot}})} + \frac{\rho_{\text{tot}}^K - \rho_{\text{tot}}}{1-\rho_{\text{tot}}^K} \cdot \frac{1}{\mu(1-\rho_{\text{tot}})} \cdot \frac{1}{2-\rho_{\text{tot}}}. \tag{4.1}$$

The equation 4.1 gives a surprisingly simple relationship between the response time for the foreground traffic, the number of services $K$, the utilization per queue $\rho_{\text{tot}}$ and the mean of the foreground job sizes $\frac{1}{\mu}$. When number of services $K \to \infty$, the response time equation 4.1 can be further simplified to

$$\text{RT} = \frac{2}{\mu(2-\rho_{\text{tot}})}.$$

These formulae have been deduced under the assumption that foreground traffic intensity is negligible compared to background traffic. Inspired by the numerical results in Figure 4.6 we investigated whether this response time formula could be used as an approximation for larger values of the percentage of the foreground traffic. A first comparison between our approximate formula and simulations, taking the simulations as the baseline, is given in Table 4.1. The comparison indicates that:

- As expected, for a fixed number $K$ of concrete services, the difference between our analytical results and simulation increases when the percentage of foreground traffic becomes larger. This is because our formula has been deduced under the assumption that there is only one foreground traffic arrival.
- Roughly speaking, the error of our approximate formula increases as the number of services $K$ increases (and all other parameters remain the same).
- The relative difference between our formula and simulation increases when $\rho_{\text{tot}}$ increases and all other parameters remain the same

Table 4.1: Relative comparison between the response times obtained by simulations and response times calculated using the formula.

| | $K=2$ | | $K=4$ | | $K=8$ | |
|---|---|---|---|---|---|---|
| FG traffic (%) $\rightarrow$ | 5 | 10 | 5 | 10 | 5 | 10 |
| $\rho_{\text{tot}}=0.5$ | 0.1% | 0.19% | 0.5% | 1.6% | 1.8% | 1.5% |
| $\rho_{\text{tot}}=0.7$ | 1.1% | 1.7% | 1% | 1.3% | 2.4% | 8.2% |
| $\rho_{\text{tot}}=0.9$ | 1.7% | 4.6% | 5.2% | 8.6% | 5.0% | 13.5% |

### 4.3.4 Scenarios with background traffic and stale system state information

For these scenarios we have conducted simulations in order to investigate which factor has more impact to the response time: delayed system state information or background traffic.

The simulation results presented in Figure 4.7 for the JSQ–RR strategy, apply to the case when $\rho_{\text{tot}}$ is fixed at 0.7 and the number of services $K = 4$. Results are shown for four different values of the parameter $D$ representing the system state information delay: $D \in \{1, 2, 5, 10\}$. As for the case with up-to-date system state information ($D = 0$) considered in the previous subsection, we see that the response time as function of the percentage of foreground traffic has a decreasing trend. Obviously, when background traffic diminishes, the response time approaches the values obtained for the scenarios without background traffic considered in subsection 4.3.2. However, all together, it is hard to determine from this figure which of the two factors has predominant influence on the response time.

In order to investigate whether delayed system state information or intensity of the background traffic has more impact to the system performance, we compare results from Figure 4.7 ($RT_{\text{BG}+\Delta}$) to results when only stale information is present ($RT_{\Delta}$). The comparison is presented at Figure 4.8 and represents the ratio $r = \frac{RT_{\text{BG}+\Delta}}{RT_{\Delta}}$ for different values of the system state information delay parameter $D$. The ratio $r$ is lower bounded by 1, and when $r \rightarrow 1$ delay has more influence on $RT_{\text{BG}+\Delta}$ than background traffic.

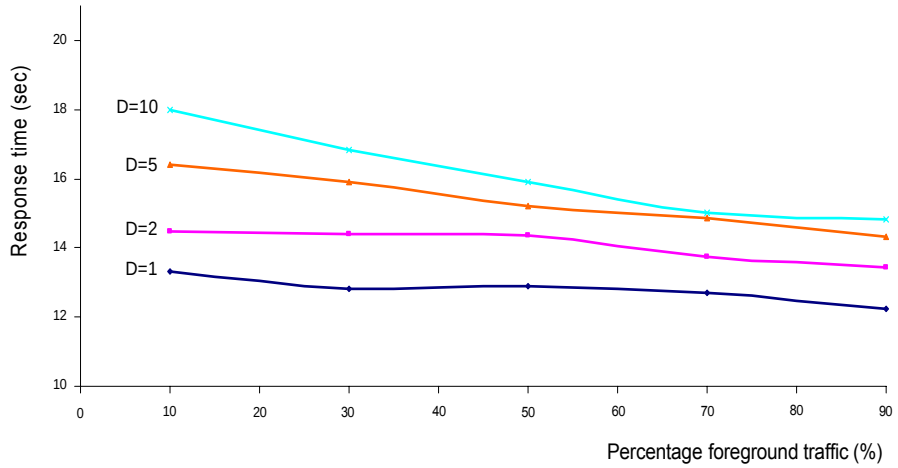


Figure 4.7: Response times for JSQ–RR strategy in case of the scenario with background traffic and stale information, with parameter $D \in \{1, 2, 5, 10\}$. Utilization per queue is $\rho_{\text{tot}} = 0.7$ and the number of services is $K = 4$.

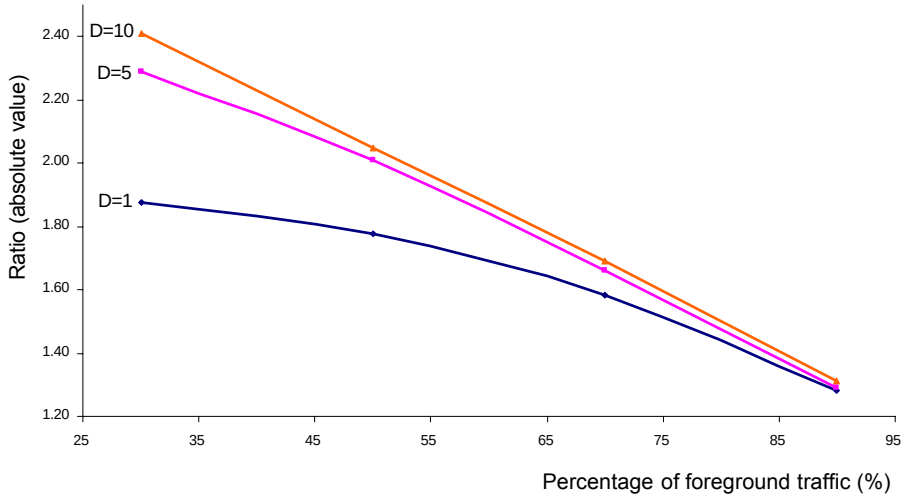


Figure 4.8: Ratio of response times between the system with background traffic and stale information and the system with stale information only. The dispatching strategy is JSQ–RR. Utilization per queue is $\rho_{\text{tot}} = 0.7$ and the number of services is $K = 4$.

The following conclusions can be made from Figure 4.8:

- The larger $D$, the more influence the background traffic has on $RT_{\mathrm{BG}+\Delta}$. Suppose that the percentage of the foreground (background) traffic in the system is fixed. As $D$ increases, the interval when state information is collected becomes larger. The larger the interval, the more background traffic arrivals to a queue between two state information updates. The response time of the tagged foreground arrival will therefore be influenced by more background arrivals.
- For smaller values of parameter $D$, the relative change of ratio $r$ is smaller. For example, when $D = 1$ the ratio changes from 1.88 (30% foreground traffic) to 1.28 (90% foreground traffic), compared to change from 2.41 to 1.31 when $D = 10$, respectively. This means that absolute influence of background traffic is smaller for smaller values of $D$. The smaller the period of the system state information update, less background traffic arrivals are probable within one such period.

## 4.4 Conclusions

In this chapter, we have investigated the performance potential of dynamic, run–time web service selection within SOA, under various assumptions regarding the available system state information and/or presence of background traffic. Using the model of request scheduling at individual web services as PS queueing system, simulation results are presented for different run–time selection strategies in scenarios ranging from the "ideal" situation (up–to–date state information, no background traffic) to more realistic scenarios in which state information is stale and/or background traffic is present. In particular, we show the effectiveness of a selection strategy based upon the "synthesis" of JSQ and RR strategies. For some specific scenarios we derive and validate insightful approximate formulae for the resulting response times. The observed performance improvements result from exploiting workload fluctuations that occur at relatively small time scales mainly caused by the random behaviour of potential clients.

The promising results raise several directions for further research. One of possible directions is the performance analysis under more general assumptions regarding the requests' arrival processes and their service requirements. Also, the impact of the resulting overhead due to making the required system state information available needs to be addressed as well. Consequently, the performance of alternative dispatching strategies without additional overhead, e.g. strategies based on response times from previously assigned jobs instead of explicit (stale) system information may be an interesting field for future work.

The analysis for a single task presented in this chapter may be used as a starting point and indication of the potential performance gain for a composite service which workflow consists of more than one task. The service selection presented in this chap-

ter uses (stale) information about the status/performance of considered services. In the following chapters, we will consider models that are based on statistical information of considered services, and will extend the analysis to the worfklows with more than one task, with the objective to maximize the profit of the composite service provider.

CHAPTER 5

# Optimal Stopping Policies for Dynamic Service Composition

In the previous chapter, the performance potential of dynamic web service selection for a single task has been studied. This service selection uses (stale) information about the status/performance of considered services. As the individual services used for service composition are owned and controlled by independent third–party domain owners, gathering of detailed service usage and state information may be challenging for the CSP. In this context, it may be useful to model the services using black–box models. For example, the response time of a service can be modeled based on its specification (e.g. average, variance, percentile) in the SLO.

Next to modelling QoS attributes in an alternative way, for the rest of the thesis we focus on the problem of *per–request, per–task* optimal service selection for composite web services represented by workflows that contain multiple tasks. The *long–term* objective of this dynamic, run–time service selection approach is to maximize the profit of the CSP, taking into account execution costs incurred by CSP per web service request. When the CSP responds to a service request within a pre–specified deadline, it receives a reward, otherwise it pays a penalty to its clients. In our approach we gradually increase the amount of additional choice that the orchestrator has for service selection decisions in the consecutive chapters. A key motivation for such approach comes from the idea that taking advantage of even a small amount of additional choice [70] for an orchestrator can lead to significant performance (profit) improvements.

In this chapter we investigate the problem of profit maximization for composite web services by allowing the orchestrator to block requests before executing the next task in the workflow. The orchestrator may terminate the execution of a single request e.g. in the case when the end–to–end deadline would be almost certainly missed, and there are still some tasks to be executed. We focus on the scenario of composite service represented by *sequential* workflow. In fact, we derive decision logic for request control that uses actual time–to–deadline as well as parameters in SLAs among different parties to optimize profit for the composite web service provider. The main question we investigate is whether the orchestrator should ever terminate the execution of a single request within the composite service workflow,

and if so, when to do it.

The main contributions of this chapter include:

- A model for sequential decision–making for the composite service.
- Estimation of the profit(s) for different cost structures when two types of decision–making algorithms are used (a) the optimal, dynamic programming backward recursion and (b) heuristic, "forward–looking" algorithms. These profits are also compared with the basic scenario when no decision–making is applied.
- Analysis of the errors made during the decision process for both algorithms.

The chapter's structure is as follows: after an overview of related work in Section 5.1, we develop the model of the considered system in Section 5.2. In Section 5.3, we specify two algorithms to derive decision logic for request control. Following the results from experiments on the algorithms presented in Section 5.4, we conclude the chapter in Section 5.5.

This chapter is based on paper [118].

## 5.1 Related work

The general sequential decision problem and optimization approaches are discussed, among others, in [9, 46]. However, none of the many papers dedicated to it describe its usage within the context of the composite web services. Cardellini et al. [25] discuss the solution to dynamically adapt at runtime the composite service configuration. The solution is based on a service selection scheme that minimizes the cost while guaranteeing the negotiated QoS specified within the SLAs. However, this paper does not consider the rewards and penalties when QoS targets for e.g. response time are met/missed. Shaaban and Hillston [88] propose a cost–based admission control approach with main goal to preserve QoS in Internet commerce systems. Different from rejecting customer requests in a high–load situation, a discount–charge model is used, that depends on system load and internal service structure, in order to encourage customers postponing their requests. They apply a scheduling mechanism based on load forecasting in order to schedule user requests in more lightly loaded time periods. However, this has not been applied within the composite service environment, and the main goal is to preserve the QoS of the system, not to maximize the profit of the CSP when admitting requests to the system.

## 5.2 Model description

In this section we describe our mathematical model for analysis of the problem at hand, using the representation of service composition in Figure 5.1, in which the

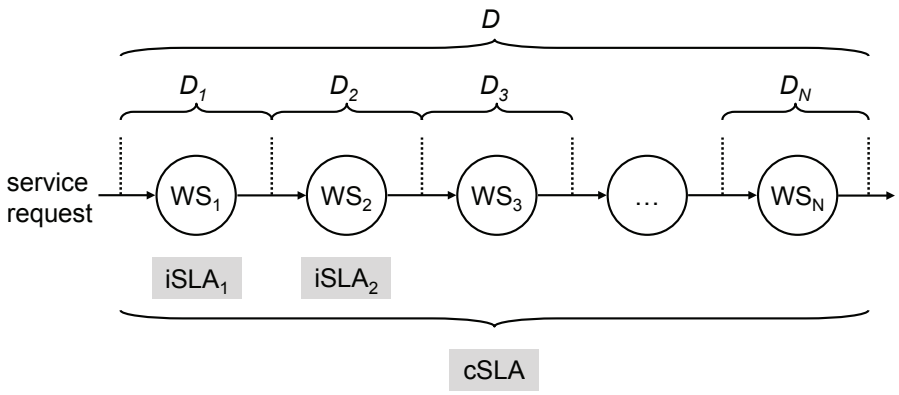


Figure 5.1: Sequential composite service orchestration.

(individual) Web Services are executed sequentially. There are in total $N$ individual services in the orchestration. The response time $D_i \geq 0$ of service $i$ is a random variable for which (an estimate of) the PDF is given. In practice, this PDF may either be estimated from measurements carried out by the CSP, or the third–party domains may publish, or otherwise make available, such information. In our model these PDFs are not time–dependent, but in practice their estimates can be dynamically adjusted. The PDFs are general (i.e. not assumed to belong to any particular class of distribution) and we denote these with $f_i(x)$ for service $i$. The realisation $d_i$ of the response time is a single value drawn from the given PDF. Let $p_i := \mathbb{P}\{D_i \leq \delta_{ip}\}$, i.e. the probability $p_i$ that $D_i$ is smaller than the value $\delta_{ip}$. We assume that all probabilities within observed system are equal, i.e. $p_i = p$ and that the response times of individual web services are mutually independent.

The random variable representing the end–to–end response time, $D$, is given as $D = D_1 + \cdots + D_N$. Since the response time PDFs of individual web services are known, the response time distribution $f^*(x)$ for the composite web service, can be computed by convolving the response time PDFs of the third party domains. The convolution(s) of the probability density functions can be done offline, i.e. before the composite service is deployed, but must be updated if dynamic estimates for the PDFs are used. The realisation of $D$ is represented by $d$, and let $p_{e2e} := \mathbb{P}\{D \leq \delta_p\}$ be the probability that $D$ is smaller than the given target $\delta_p$.

We refer to the client–CSP SLA as cSLA while the SLA between CSP and third–party domains $i, i = \{1, 2, \ldots, N\}$ is referred to as iSLA$_i$. The iSLA$_i$ $(i = 1, 2, \ldots, N)$ contains the following elements:

- The response–time probability density function.
- The response–time penalty deadline $\delta_{ip}$ [time unit].
- The probability that the penalty deadline is met, $p_i$ $(0 < p_i \leq 1)$.

- The reward that the service provider gets for executing a single request, $c_i$ [money unit]; from the CSP viewpoint, this value represents cost.

The cSLA contains the following elements:

- The end–to–end response time penalty deadline $\delta_p$ [time unit].
- The probability that the end–to–end penalty deadline is met $p_{e2e}$ ($0 < p_{e2e} \leq 1$).
- The reward $R$ [money unit] that the CSP gets for executing a single request within penalty deadline $\delta_p$.
- The penalty that the CSP pays to the end customer when the request has not been completed, or the agreed end–to–end deadline for given request is not met, $V$ [money unit].

The SLAs, once agreed, do not change, i.e. above mentioned parameters do not change either with time. The CSP obtains the reward when the response time of the request is below the promised value $\delta_p$. Otherwise, the CSP pays the penalty to the clients. The promised deadline $\delta_p$ is met with probability $p_{e2e}$. Suppose that the request has been served by the composite service within agreed time. The profit in that case is $R - Nc$. The biggest loss for the CSP occurs when the complete chain is traversed and the end–to–end deadline is not met. This loss amounts to $V + Nc$. Taking into account probabilities specified in SLAs, in order for the CSP to make profit, the following should hold

$$p_{e2e} \cdot (R - Nc) - (1 - p_{e2e}) \cdot (V + Nc) > 0.$$

However, it may happen that the orchestrator decides to terminate the execution of the composite request after service $k$ ($1 < k < N$), for example because the promised response time $\delta_p$ is already breached, and continuation of the service execution would only increase the loss of the composite service provider. Also, depending on the response times and costs involved, the orchestrator may decide to terminate the further execution because the risk of not reaching the promised response time is significantly high.

## 5.3 Sequential decision processes

In this section we describe two decision algorithms. The first algorithm, called the Dynamic Programming Algorithm (DPA), is based on the backward–recursion principle [15]. The second algorithm, called the Forward Looking Algorithm (FLA), which is based on ideas expressed in [9].

The algorithms considered facilitate decision–making process of the orchestrator i.e. whether to further traverse the workflow after each task within the workflow has been executed. The actions that the orchestrator could take are:

- `stop` the execution. The CSP pays the costs to the third party providers of used services. Besides, the CSP pays the penalty to its client(s),
- `continue` the execution, i.e. execute the next task.

If the last service is executed the CSP pays the actual costs to the third-party providers. When the end–to–end deadline is met the CSP obtains reward from the client(s); otherwise, the CSP pays the penalty to the client(s).

The profit gains for both the DPA and the FLA are calculated in absolute terms, and compared to the "worst case scenario" (WCS) when all tasks within workflow are always executed. For all algorithms, the first task within the workflow is always executed. The DPA algorithm results in pre–calculated *decision policy*, i.e. actions to be taken by the orchestrator after task $k$ ($k = 1, 2, \ldots, N-1$) has been executed, and for any given value of remaining time to deadline $\delta^* \leq \delta_p$. Due to the backward–recursion principle, the calculated policy would specify *optimal* actions for considered task $k$, taking into account all possible actions (`stop/continue`) for tasks $k{+}1, \ldots, N$ and given $\delta^*$.

On the other hand, the FLA is heuristic that determines actions to be taken by the orchestrator after task $k$ ($k = 1, 2, \ldots, N-1$) has been executed by comparison of decisions to stop the execution or to continue and execute all of the remaining $N-k$ tasks.

### 5.3.1 The Dynamic Programming Algorithm

The DPA is based on a one-dimensional recursion that calculates a set of optimal decisions, namely `continue` or `stop`, given the decision is taken before service $k$ and $\delta^*$ units of time budget are left before the agreed deadline is exceeded. Using the notation introduced in previous sections, the backward recursion is formulated as follows: For $1 < k < N$,

$$\mathbb{E}[R_N \mid \delta^*] = \max\Big\{ -V, -c - \mathbb{P}(D_N > \delta^*)V + \mathbb{P}(D_N \leq \delta^*)R \Big\}, \tag{5.1}$$

$$\mathbb{E}[R_k \mid \delta^*] = \max\Big\{ -V, -c - \mathbb{P}(D_k > \delta^*)V + \int_0^{\delta^*} f_k(t)\mathbb{E}[R_{k+1} \mid \delta^* - t]\mathrm{d}t \Big\},$$

$$\mathbb{E}[R_1 \mid \delta_p] = \max\Big\{ -V, -c - \mathbb{P}(D_1 > \delta_p)V + \int_0^{\delta_p} f_1(t)\mathbb{E}[R_2 \mid \delta_p - t]\mathrm{d}t \Big\}.$$

Here $\mathbb{E}[R_k \mid \delta^*]$ ($1 < k < N$) is the expected reward in case the decision made is `continue` and $\mathbb{E}[R_1 \mid \delta_p]$ is the total expected reward given an overall deadline of $\delta_p$ time units. In practice, the recursive equations (5.1),can be solved efficiently by discretizing the response–time distributions.

### 5.3.2 Forward–looking decision process

The FLA decision mechanism is based on evaluation of the expected reward once the first $k$ ($k = 1, \ldots, N-1$) tasks within workflow are executed. The orchestrator decides whether the remaining $N-k$ tasks will be executed. At the decision–taking moment, the orchestrator knows:

- the response times of the services already executed, i.e. the realisations $d_1, d_2, \ldots, d_k$ of $D_1, D_2, \ldots, D_k$, respectively.
- the response time probability–density functions for each of the $N-k$ services to be executed.

Based on this information, the orchestrator calculates the following:

- The remaining "response–time budget", $\delta^* = \delta_p - \sum_{i=1}^{k} d_i$.
- $\mathbb{P}_0$, the probability that $\delta^*$ will be met. The reward of the CSP in this case is $R - Nc$.
- Probability that $\delta^*$ will not be met, $\mathbb{P}_{\text{LOSS}} = 1 - \mathbb{P}_0$. The loss in this case is $-V - Nc$.

After the service $k$ ($k = 1, 2, \ldots, N$) has been executed, the algorithm calculates the expected reward, *as if all the remaining tasks within workflow would be executed*, and compares it to the loss in case the execution is terminated after service $k$. The expected reward is

$$\mathbb{E}[R_k] = \mathbb{P}_0 \cdot (R - Nc) - \mathbb{P}_{\text{LOSS}} \cdot (V + Nc). \tag{5.2}$$

The loss when the execution terminates (after execution of the service $k$) is

$$\text{TH}_k = -V - \sum_{i=1}^{k} c_i = -V - kc. \tag{5.3}$$

The FLA is then based on the following decision rule:

$$\mathbb{E}[R_k] > \text{TH}_k = \begin{cases} \text{true}, & \texttt{continue} \\ \text{false}, & \texttt{stop}. \end{cases} \tag{5.4}$$

## 5.4 Numerical experiments

The framework and the two algorithms described in this chapter can be used to numerically determine the corresponding decision strategies. In order to test a wide range of (heuristic) strategies, including the WCS without decision control, we also developed a simulation tool. In this section we give an overview results that were obtained with our simulation tool, to illustrate the effectiveness of our algorithms

and demonstrate the practical usability of our algorithms. Here, we limit ourselves to comparison of our two algorithms with the WCS. Further experimentation for a wider set of strategies is subject of current research.

In the experiments presented here, we have chosen the following parameter settings:

- Requests consist of a chain of $N = 6$ services.
- The response–time distributions used for experiments were normal (with truncation of negative values) for all services.
- The probabilities that (individual or composite) service would meet its deadline were identical and set to one of the values 0.75, 0.8, 0.85, 0.9 or 0.95.
- The reward parameter is fixed to a value of 2.5.
- The cost and penalty parameters have been chosen in such a way that the expected profit per request $\mathbb{E}[R_{WCS}]$ made by the CSP is positive for the WCS and remains the same. The relationship between $V$ and $c$ is determined by

$$c = \frac{Rp - V(1-p) - \mathbb{E}[R_{WCS}]}{N}.$$

The two main experimental setups are

**Setup A:** In this setup response time PDFs of the services were identical, and represented by (truncated) normal distributions with the same parameters for expectation and variance. The values selected were $\mathcal{N}(\mu, \sigma^2) = \mathcal{N}(15, 4)$.

**Setup B:** In this setup the response time PDFs of the services were not identical, although all considered distributions were (truncated) normal. The expectation and variance of the first two services within the chain are chosen to be significantly higher than expectation and variance of the services within the rest of the chain. The values selected for the experiments were $\mathcal{N}_1 = \mathcal{N}(150, 25)$, $\mathcal{N}_2 = \mathcal{N}(100, 16)$, and $\mathcal{N}_{3-6} = \mathcal{N}(15, 4)$.

The fact that distributions are identical in setup A allows that delays (i.e. deadlines) that are not met "early" in the chain could be compensated by better than expected performance as the request traverses the chain. The chance of prematurely terminating the execution of the composite request therefore increases. Good algorithms would show quantifiable improvements even for these scenarios.

The setup B has been used to verify the accuracy of the algorithms observed, since delays introduced by services at the beginning of the chain cannot be made up for by the services close(r) to the end of the chain. It may be expected that the algorithms would defer requests at one of the first two services more often than at one of the last four services.

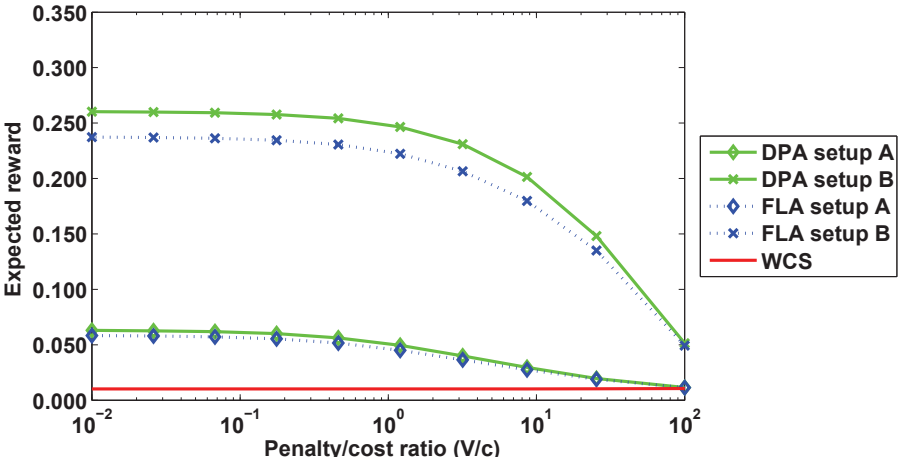


Figure 5.2: Comparison of expected rewards per request using different algorithms for setup A and B. The promised percentage of requests within end–to–end deadline $p = 90\%$.

Our results emphasise two performance aspects of the algorithms:

- Increase in the reward (absolute and/or relative) that algorithms yield when compared to the WCS, i.e. the "do nothing" scenario.
- Amount of erroneous decisions made.

### 5.4.1 Reward improvements

For each value of the probabilities, cost parameters selection made, and both setup A and B, we have recorded the performance of the DPA and the FLA. Some results are summarized in Figure 5.2 for setup A and B, and Figure 5.3 for setup B.

The following conclusions can be made from the experiments:

- Setup A shows less improvement than setup B.
- If the penalty becomes much higher than the cost parameter, the expected reward drops to the WCS value.
- Performance results are better in case when deadline probabilities chosen are smaller. Put simply, the higher probability deadline is met, the less requests could be terminated by the algorithms. The less requests that are terminated, the less profit is gained by the algorithms when compared to WCS.

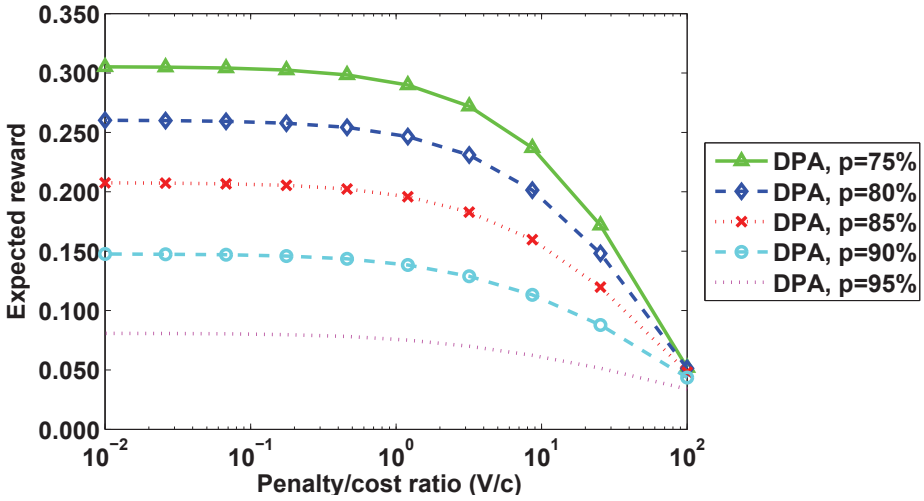


Figure 5.3: Comparison of expected rewards per request with different $p$ values for setup B.

### 5.4.2 Decision errors

We define two type of decision–making errors: Type I and Type II. **Type I** errors are made within the decision process when execution of the request is terminated before the end of the chain is reached, while the optimal decision is to traverse the whole chain. **Type II** errors are made within the decision process when the request traverses the complete chain, while the optimal decision is to terminate the request.

Suppose that a Type I error has been made by terminating the request once the service $k$ has been executed. The composite service provider is left without reward and also has to pay the penalty. The net loss is therefore $-R-V+(N-k)c$. Similarly, Type II errors reduce the algorithm performance for that request to the worst case scenario, and the net loss reduces to the costs made by redundant invocation of services within the chain. Therefore, in general, reduction of Type I errors leads to larger profit increase than the reduction of Type II errors.

The experimental results are summarized in Figure 5.4, which shows percentage of errors made for different values of the penalty/cost ratio $V/c$. We observe that, for both DPA and FLA, the percentage of Type I errors decreases as the ratio $V/c$ increases. This is due to the larger impact of the penalty, as stopping becomes extremely expensive compared to the additional invocation cost of the next sub–service. Similar reasoning explains percentage increase of Type II errors as ratio $V/c$ increases. Besides, DPA is not optimal when it comes to Type II errors, and the percentage of Type II errors for both DPA and FLA is below 2.5% for the observed ratio interval.

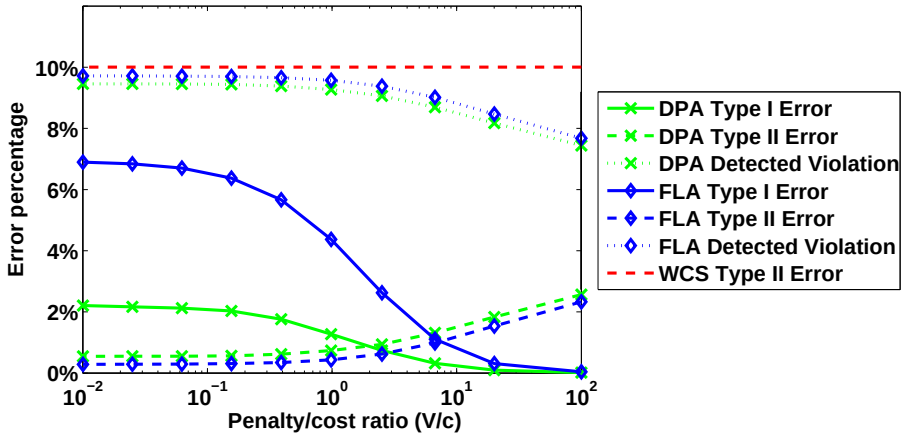


Figure 5.4: Comparison of errors made by different algorithms for setup B. The promised percentage of requests within end–to–end deadline $p = 90\%$.

## 5.5 Conclusions

In this chapter we have discussed decision–making mechanisms for the composite web services within service oriented architecture. First, we have considered the model of the composite web service in which the CSP pays a certain amount per request to each third party provider. In addition, the CSP pays a penalty to the customer whenever the execution of the composite service request is prematurely terminated or the agreed end–to–end response time is above the threshold promised by the CSP. Further, a stochastic model has been formulated in which the sub–service response times are modeled as stochastic variables with probability distributions known *a priori*.

Based on the derived model, two different decision–making approaches have been discussed, the DPA and the FLA. These two approaches were compared to the WCS approach (without decision–making) and to each other. In the performed simulations, two main scenarios have been examined. Firstly (symmetric scenario) we considered a composite web service consisting of the chain of six sub–services with identical response time distributions. In the second, asymmetric scenario the first two sub–services have a significantly larger expected response time and variance. We have shown that huge profit gains with respect to WCS can be made using any of the two algorithms. As expected, the profit gains in asymmetric scenario(s) were much higher than in the case of symmetric scenarios. The DPA is optimal when it comes to the CSP profit, whereas the FLA performs better when it comes to errors made when the request traverses the complete chain, while the optimal decision is to terminate the request.

We presented numerical results for rather small model instances, with six sub–

services, whereas in practice, services chains may be of considerably larger size. This raises the need to address the scalability of the solution approaches presented (in terms of computational complexity), and about the cost reductions that can be obtained in those cases. In this context, also notice that for large number of tasks in the considered workflow the classical Central Limit Theorem suggests that the optimal policy becomes less sensitive to the response–time performance of the individual sub–services, which opens up the way for CSPs to negotiate much less strict, and hence much cheaper, SLAs with respect to the response times.

CHAPTER 6

# Optimal Service Selection Policies with Response–Time Constraints

In this chapter we extend the problem considered in Chapter 5 and investigate more sophisticated decision mechanisms for composite web services. In particular, the actions that can be taken by the orchestrator include the service selection among a number of alternatives instead of just stop/continue. In this setting, a composite web service is represented by a (sequential) workflow, and for each of the tasks within this workflow, a number of service alternatives may be available, offering the same functionality at different price/quality levels. We present a fully dynamic service composition approach, where for *each individual* request at *each* task in the workflow it is decided at runtime which alternative of the corresponding sub–service is invoked. The decisions are based on observed response times, costs and response–time characteristics of the alternatives as well as end–to–end response–time objectives and corresponding rewards and penalties.

We derive the service selection policy maximizing expected revenue by exploitation of dynamic programming. The corresponding decision rules can be computed offline and stored in the form of a simple lookup table. Extensive numerical examples demonstrate huge potential gain in expected revenues using the dynamic approach compared to other, non–dynamic, approaches. We explore the impact of reward, penalty and service execution cost parameters, in relation to the services' response time characteristics (PDF, mean, variance). In addition, the impact of the number of available concrete services (alternatives) and their position in the workflow is investigated. Overall, the numerical results show huge potential for the run time service selection approach.

In contrast to most of the state–of–the–art solutions discussed in Chapter 2 the response times of the services are not assumed to be just a fixed value (e.g. the empirically found mean value), but are represented by a PDF. One of the challenges is to take these uncertainties regarding the response times of the sub–services appropriately into account in the decisions.

The main contributions of this chapter are as follows:

- Modeling dynamic service composition as a dynamic programming problem.

Using this approach an optimal service selection policy is derived which can be expressed in terms of response–time thresholds that are calculated before the service is deployed. In such a way, fast service selection at runtime is possible since service selection can be implemented as a simple lookup function, e.g. lookup table or database query.

- Numerical investigation of our full dynamic approach as well as a comparison to the optimal a–priori static selection, which shows significant potential gain in expected revenue.
- Analysis of the impact of the number of available alternatives (and their QoS properties) for each task within the sequential workflow. The results show nice monotonicity properties of the expected revenue and combination of the position of the task within the workflow and the number of alternatives available for the given task.

The rest of this chapter is organized as follows. After further motivation and illustration of our work in Section 6.1 and related work in Section 6.2, we describe the model of the system under consideration in Section 6.3. In Section 6.4 we formulate the run–time service composition problem as a DP problem and provide the corresponding backward recursion solution method used to determine the optimal decision policy. Next, in Section 6.5, the results of extensive numerical experiments are presented and discussed in order to show the performance of our run–time service composition approach and provide insight into the impact of different system parameters. We conclude the chapter in Section 6.6, and give directions for further research.

This chapter is based on papers [116] and [117].

## 6.1 Motivating example

In order to illustrate our dynamic service selection approach, we observe the case of sequential workflows. Fig. 6.1 depicts a sequential workflow consisting of four abstract services, and each abstract service maps to a number of concrete services (alternatives). Per position, the number of alternatives is three, four, one and two, respectively. The response time of concrete web services are modelled by probability distributions and the execution costs per concrete service are known in advance as well. In case when the workflow is not sequential, it could be reduced/aggregated to the sequential one. This can be done by the calculations of the aggregated services for the most frequently used workflow patterns in which the probabilistic models are used as explained in [12, 115]. In case of arbitrary workflow patterns, an efficient numerical method could be used, as presented in [28]. After the execution of a single task within the workflow, the orchestrator decides on the next concrete service to be executed. The decision taken is based upon comparison among (1) the remaining time to meet the end–to–end deadline, and (2) the pre–calculated response–time

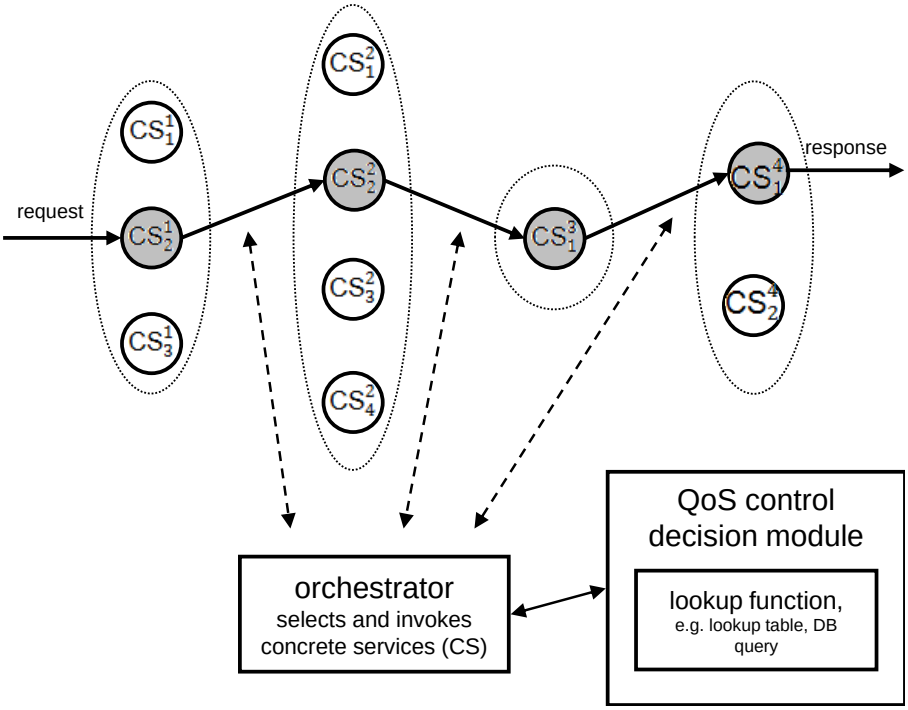


Figure 6.1: Composite web service depicted by a sequential workflow. Dynamic service composition is based on pre–calculated response–time thresholds using dynamic programming.

thresholds for each service. In the most extreme example, after the execution of, e.g. the first task within the workflow, the promised end–to–end deadline may already been violated. In such a case, the dynamic selection algorithm should choose the cheapest alternatives for all remaining tasks in the workflow. Similarly, when there is just the last task to be executed, and the promised deadline is jeopardized, it may be sensible to select the more expensive service with better deadline–meeting promise than the cheaper service with the much smaller chance of meeting the deadline.

When the client's request meets the agreed end–to–end deadline, the composite service provider is rewarded by the client. Otherwise, when the deadline is not met (i.e. an event of an SLA violation) the provider pays a penalty to the client. The exact relation between the metric of percentile conformance and the monetary penalties charged on the provider due to non–conformance (hereafter referred to as the "penalty function") is an important parameter of the SLA [17]. There are multiple ways in relating the two and thus a variety of penalty functions can be chosen. We have adopted the step–wise penalty function in this case with the desired conforming percentile of 100% [17].

The response–time thresholds are calculated before the first request is admitted to the system, where the dynamic programming takes the penalty, the reward, the concrete SLO and the execution costs as input. These thresholds are represented by e.g. a lookup table. Depending upon the actual response times and the thresholds, it is possible that every client request may be served by a different chain of alternatives.

## 6.2 Related work

A lot of attention in the literature has been paid to the problem of optimal, QoS–aware service composition of SOA services [21,111,113]. The main problem addressed in these papers is how to select one concrete service per abstract service for a given workflow. This selection is made with the goal to guarantee the end–to–end quality of the composite service, while at the same time achieving certain objectives like cost minimization. For the same QoS parameter, different SLAs are considered in literature, ranging from those based on a single value (e.g. expected response time) to SLAs specifying probabilistic guarantees [115] and [12].

In static service composition solutions, the composition would remain unchanged during the entire life–cycle of the composite web service. Due to the high variability of the service environment and due to the fact the variability is not well–reflected in static service composition solutions, the SLA violations could occur relatively often, leading to providers' losses and customer dissatisfaction. One way to address the problem of SLA violations is the request replication approach, as presented in [110]. Although showing relative potential, the request replication may not work for any set of alternatives given, and may be cost–inefficient. It is suggested in [23, 61, 114] that, based on observations of the actually realised performance, re–composition of the service may be triggered, in order to mitigate the issue of SLA violations. These "reactive" solutions may select new concrete service(s) for the given workflow during the re–composition phase. Once the re–composition phase is over, the (new) composition is used as long as there are no further SLA violations. In particular, the authors of [23, 61, 114] describe *when* to trigger such (re-composition) events, and *which* adaptation actions may be used to improve overall performance.

A highly promising means for further improvement of service quality and efficiency, which is also addressed in this chapter, is to adapt the service composition *dynamically at run time* by taking advantage of this enhanced flexibility [1,25,44,99]. In [44], the authors analyse a problem of dynamic web service composition for different composition patterns, e.g. sequential, loop, conditional and parallel. QoS parameters considered include reliability, availability, as well as cost and expected value of the response time of individual web services. The authors propose a solution based on MDPs to minimize the expected response time, taking into account the availability and reliability of the respective services and the invocation costs. In [99], a similar dynamic service composition problem is studied. Besides deriving an MDP–based solution, the authors of [99] also develop and investigate a reinforcement learning

approach for the case that not all of the used system state information is available. Cardellini et al. [25] consider dynamic service composition in the context of admission control with various service classes. The (different) composite service configurations for the service classes can be dynamically adapted according to variations in the operating environment due to the admission or departure of users generating requests for the composite service. The authors derive an optimal admission and re–composition policy (by formulating the problem as a linear optimization problem) that maximizes the profit while guaranteeing QoS for the admitted users. In [1], for a somewhat different setting, a forward looking MDP–based control policy is derived.

Different from the presented works, we focus on *per–request, per–task* QoS–aware service composition, taking into account different price/quality levels, as well as reward/penalty functions. We perform the optimal service selection with long–term objectives such as profit maximization of the CSP. By doing this, we take into account the cost of control (adaptation of service composition) as well.

## 6.3 Model description

A composite service consists of $N$ of *abstract services*, denoted $AS_1, \ldots, AS_N$, that are executed in sequential order. For each abstract service $i$ there are $M_i$ alternative implementations available, which are called *concrete services*, denoted $\mathrm{CS}_j^i$, for $j = 1, \ldots, M_i$. The composite service handles incoming composite service requests. To this end, each request is assigned a unique service alternative for each abstract service in the workflow. In other words, each composite service request is served by a *composition*

$$\mathrm{CS}_{w_1}^1 \rightarrow \cdots \rightarrow \mathrm{CS}_{w_N}^N \tag{6.1}$$

that maps each abstract service $AS_i$ to a service alternative $\mathrm{CS}_{w_i}^i$. Each composition is represented by a *workflow vector* $\boldsymbol{W}$ that is defined as:

$$\boldsymbol{W} := (w_1, \ldots, w_N). \tag{6.2}$$

The position of $w_i$ in the vector corresponds to the position in the workflow and the values $w_i = 1, \ldots, M_i$ correspond to the concrete service alternative at position $i$.

**Remark [non sequential workflows]** Our solution is applicable to any workflow that could be aggregated and mapped into a sequential one. It is shown in [116] how to aggregate an example workflow into a sequential one. The aggregation is illustrated for some of basic workflow patterns in case of stochastic (probabilistic) models of services' QoS parameters. The end result is a sequential workflow pattern, in which each service (aggregated or not) has a number of alternatives. However, the aggregation leads to coarser control, since decisions could not be taken for a single service within the aggregated workflow, but rather for the aggregated workflow patterns themselves.

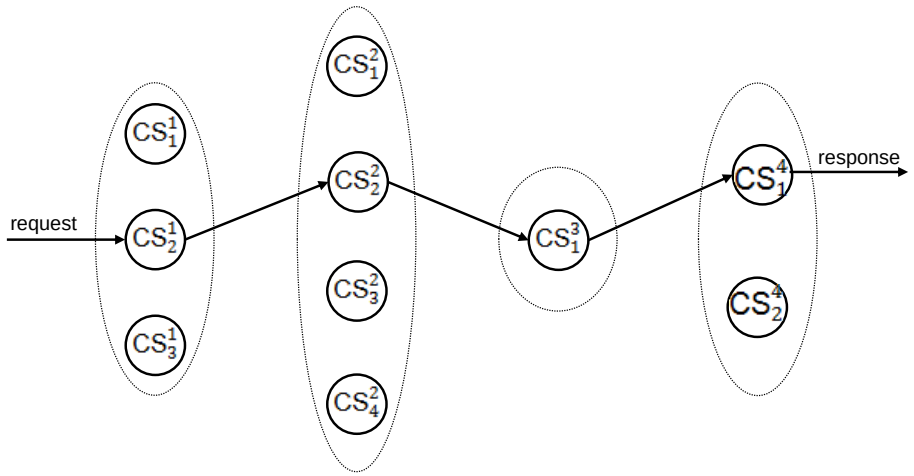


Figure 6.2: Model with example composition $\mathrm{CS}_2^1 \rightarrow \mathrm{CS}_2^2 \rightarrow \mathrm{CS}_1^3 \rightarrow \mathrm{CS}_1^4$.

Figure 6.2 illustrates an example for the case $N = 4$, $M_1 = 3$, $M_2 = 4$, $M_3 = 1$, $M_4 = 2$ and $\boldsymbol{W} = (2, 2, 1, 1)$. Per single composite service request, the orchestrator (illustrated in Figure 6.1) executes services one–by–one as indicated by the workflow. Before abstract service $AS_i$ is executed, the orchestrator makes a decision which one of the $M_i$ service alternatives to choose. Decisions from the orchestrator are based on information about response times from the concrete services. The response–time SLOs of the concrete services are specified as "soft" ones, and in general, "soft" SLOs are expressed as a response–time PDF [83], or alternatively, as CDF.

**Remark [probability density functions]** In practice, probability density functions may either be estimated from the measurements carried out by the composite service provider, or the third–party domains may publish, or otherwise make available, such information. Since we investigate the potential of our approach, we assume time–invariant SLAs and the PDFs (which are part of the SLAs) do not change either. In case of time–variant PDFs, a recalculation of the response–time thresholds may be occasionally necessary. The recalculation may be triggered when there is a long–term SLA violation, i.e. when the observed PDF differs "significantly" from the initial one. This is not discussed further in this chapter.

Each service $\mathrm{CS}_j^i$ (the $j$–th alternative for abstract service $i$) has a response time distribution $D_j^i \geq 0$ for $i = 1, \ldots, N$ and $j = 1, \ldots, M_i$. From the perspective of the response time, we model each concrete service as a black box, which means that $D_j^i$ is a random variable for which respective PDF (or CDF) is given. The PDFs and CDFs for concrete services are denoted by $f_j^i(t)$ and $F_j^i(t)$, respectively. If a service $\mathrm{CS}_j^i$ is invoked a response time realisation $d_j^i$ $(i = 1, \ldots, N,\ j = 1, \ldots, M_i)$ is generated. Because response time of $\mathrm{CS}_j^i$ has distribution $f_j^i$, $d_j^i$ is a sample from this distribution. We assume that response times of concrete service alternatives

are mutually independent. Using the mutual independence, for a given composition $\mathrm{CS}^1_{w_1} \to \cdots \to \mathrm{CS}^N_{w_N}$ the response time distribution can be determined by taking the convolution of the respective distributions:

$$D_{\boldsymbol{W}} = D^1_{w_1} \star D^2_{w_2} \star \ldots \star D^N_{w_N}. \tag{6.3}$$

The realisation of a end–to–end response time, resulting from composition represented by $\boldsymbol{W}$, is denoted $D_{\boldsymbol{W}}$. The overall deadline for a request handled by the orchestrator is denoted by $\delta_p$. For each workflow $\boldsymbol{W}$ the probability for a successful response within $\delta_p$ is defined by:

$$p_{\boldsymbol{W}} := \mathbb{P}(D_{\boldsymbol{W}} \leq \delta_p). \tag{6.4}$$

The workflow invocation cost $c_{\boldsymbol{W}}$ for workflow $\boldsymbol{W} = (w_1, \ldots, w_N)$ is defined by:

$$c_{\boldsymbol{W}} := \sum_{i=1}^{N} c^i_{w_i}. \tag{6.5}$$

As for a fixed workflow $\boldsymbol{W}$ the composition is exactly known, an explicit expression for the expected revenue per request can be formulated. We define $R^*_{\boldsymbol{W}}$ as the expected revenue per request for workflow $\boldsymbol{W}$:

$$R^*_{\boldsymbol{W}} := Rp_{\boldsymbol{W}} - V(1 - p_{\boldsymbol{W}}) - c_{\boldsymbol{W}}. \tag{6.6}$$

When dynamic decisions are made, concrete service workflow $\boldsymbol{W}$ is not fixed but depends on the response time realisations of concrete services. Our approach is solving the optimization problem using dynamic programming. The key idea behind dynamic programming is that the optimization problem can be separated into smaller subsets that are easier to solve. Solving the subsets will lead to the solution of the optimization problem that optimized expected revenue. Dynamic programming consists of the following components:

- **State space** $\mathcal{S}$, in our case the state space is defined by $\mathcal{S} = \{1, \cdots, N\} \times \mathbb{R}^+$. Each state is a tuple $(i, \Delta) \in \mathcal{S},\ (i \in \{1, \cdots, N\},\ \Delta \in \mathbb{R}^+)$ combining position $i$ in the workflow and remaining time until deadline violation $\Delta$.
- **Decision epochs**, before execution of the next task we have to select a concrete service alternative based on $\Delta$.
- **Action space**, the set of possible actions for given state $(i, \Delta)$. At each decision moment at abstract service $AS_i$ we have a set of concrete service alternatives $\{CS^i_1, \ldots, CS^i_{M_i}\}\ i = 1, \ldots, N$.
- **Cost function**, the cost function defines cost for each action or state that is reached. For concrete service $\mathrm{CS}^i_j$ the invocation cost $c^i_j$ has to be paid. At the end of the workflow a reward $R$ is obtained by the CSP when the deadline is not violated. Otherwise a penalty $V$ has to be paid.

- **Value function** $F(s)$. The value function represents the expected revenue for state $s \in \mathcal{S}$. We have a value function $F(i, \Delta)$ where $i \in \{1, \cdots, N\},\ \Delta \in \mathbb{R}^+$.
- **Policy** $A$ represents the set of decisions or actions over all states $(i, \Delta)$. The policy can be implemented as a lookup table where for given $(i, \Delta)$ the optimal pre–calculated decision is selected. An example is depicted in Figure 6.3. We refer to Section 6.4 for more details.

The combination of state space, decision epochs, action space, cost function and value function defines a dynamic programming problem where the solution results in a optimal policy. An optimal policy can be determined by defining a backward recursion on value function $F(s)$. The recursion is defined and implemented in Section 6.4. We denote the dynamic workflow that implements lookup table $A$ as $\boldsymbol{W}^d(A)$.

Due to the dynamic policy there is no tractable explicit expression for the end–to–end response time distribution when the orchestrator chooses compositions dynamically like for the static workflow. End–to–end response time distribution is denoted by $D$ and has corresponding realisation $d$. The fraction of requests within the deadline is

$$p := \mathbb{P}(D \leq \delta_p). \tag{6.7}$$

There are two types of SLAs that we consider:

First, the SLA agreed between the ISP that provides $\mathrm{CS}_j^i$ and the CSP. This SLA consist of the following elements:

- The response–time probability distribution function, $f_j^i$.
- The cost $c_j^i$ [money unit] that the CSP pays to ISP for the execution of a single request. No penalties are imposed. From the ISP viewpoint, this value represents reward.

Second, the SLA agreed between the CSP and its clients, that contains the following elements:

- The end–to–end response time penalty deadline $\delta_p$ [time unit].
- The fraction of response time realisations $p_{e2e}$ that should be within the deadline $\delta_p$.
- The reward $R$ [money unit] that the CSP gets for executing a single request within penalty deadline $\delta_p$.
- The penalty $V$ [money unit] that the CSP pays to the end customer when the agreed end–to–end deadline is not met.

The orchestrator selects services for composition such that it meets the required fraction $p_{e2e}$ of requests within deadline $\delta_p$:

$$\mathbb{P}(D \leq \delta_p) \geq p_{e2e}. \tag{6.8}$$

## 6.4 Algorithm description

In this section we describe how to optimise expected CSP revenue by formulating the dynamic service selection as a DP problem [15]. Before a request is executed by the CSP a response time budget $\delta_p$ is available until response time deadline $\delta_p$ is violated. Each execution of concrete service $\mathrm{CS}_j^i$ in the CSP workflow will result in a response time realisation $d_j^i$ and a remaining response time budget $\Delta$. After execution of $\mathrm{CS}_j^i$ the remaining time budget $\Delta$ will reduce with $d_j^i$. Dynamic refers to the fact that workflows are selected on–line based on $\Delta$, the remaining response time budget until the deadline $\delta_p$ is violated. In other words before the execution of each abstract service, a concrete service must be selected, based on $\Delta$. The DP optimises CSP revenue by taking in account the effect of future (optimal) decisions. Since the decisions within DP take in account (possible) effects subsequent decisions, a "backward recursion" method is needed for finding the optimal solution. The application of DP will result in a decision policy. The policy indicates, for given response time realisations, which concrete service alternatives should be chosen in order to optimize the CSP's expected revenue per composite service request. The policy is determined by the current position within the sequential workflow $i$ and the remaining time $\Delta$ ("time budget") till the overall deadline $\delta_p$ will be violated.

The recursion for a set of expected rewards $\mathbb{E}[R^i \mid \Delta = \delta^*]$ is given by:

$$\mathbb{E}[R^i \mid \Delta = \delta^*] = \max_j \Big\{ -c_j^i + \mathbb{E}[R_j^i \mid \Delta = \delta^*] - \mathbb{E}[V_j^i \mid \Delta = \delta^*] \Big\},$$

where $i = 1, \ldots, N$, $j = 1, \ldots, M_i$.

For $j = 1, \ldots, M_i$, we have

$$\mathbb{E}[R_j^i \mid \Delta = \delta^*] = \begin{cases} \mathbb{P}(D_j^N \leq \delta^*)R, & i = N, \\ \int_0^{\delta^*} f_j^i(t)\mathbb{E}[R^{i+1} \mid \Delta = \delta^* - t]\mathrm{d}t, & \text{for } i = 1, \ldots, N-1. \end{cases} \tag{6.9}$$

$$\mathbb{E}[V_j^i \mid \Delta = \delta^*] = \begin{cases} \mathbb{P}(D_j^N > \delta^*)V, & i = N, \\ \mathbb{P}(D_j^i > \delta^*)\mathbb{E}[R^{i+1} \mid \Delta = 0], & \text{for } i = 1, \ldots, N-1. \end{cases} \tag{6.10}$$

Here $f_j^i(t)$ represents the response time PDF of service $\mathrm{CS}_j^i$, while the term $\mathbb{E}[R_j^i \mid \Delta = \delta^*]$ represents the expected reward, when $\mathrm{CS}_j^i$ is invoked for the given time budget value $\delta^*$. The term $\mathbb{E}[V_j^i \mid \Delta = \delta^*]$ represents the expected penalty for exceeding the overall deadline while executing $\mathrm{CS}_j^i$ for the given time budget value $\delta^*$. The expected reward and penalty functions take into account the impact of future decisions as represented by terms relating to $R^{i+1}$ in equations (6.9) and (6.10). Once the end of the workflow is reached ($i = N$) we stop.

The integrals in equation (6.9) can become rather complicated and will generally not result in tractable expressions. By discretizing the distributions the problem can be solved numerically. For the discretization we split the time interval over which the response–time PDF is defined in segments of the same size $h$. The number of segments is $m^*$ and the size of $h$ corresponds to the accuracy of the discretization. The discretized versions of the PDF ($p^i_{j,k}$) and CDF ($P^i_{j,k}$) are therefore defined as the following:

$$m^* = \left\lceil \frac{\delta^*}{h} \right\rceil,\ p^i_{j,k} = \mathbb{P}\big(D^i_j \le h[k+0.5]\big) - \mathbb{P}\big(D^i_j \le h[k-0.5]\big),\ P^i_{j,k} = \sum_{l=0}^{k} p^i_{j,l},$$

where $i = 1, \ldots, N$, $j = 1, \ldots, M_i$, $k = 0, \ldots, m^*$.

Using the discretization, the backward recursion can be transformed into a scheme that can be evaluated numerically. For given number of segments $m^*$, let terms $R^i_{m^*}$, $R^i_{j,m^*}$, and $V^i_{j,m^*}$ represent discretized versions of $\mathbb{E}[R^i \mid \Delta = \delta^*]$, $\mathbb{E}[R^i_j \mid \Delta = \delta^*]$, and $\mathbb{E}[V^i_j \mid \Delta = \delta^*]$, respectively. The backward recursion formulae are then as follows:

$$R^i_{m^*} = \max_j \Big\{ -c^i_j + R^i_{j,m^*} - V^i_{j,m^*} \Big\}, \tag{6.11}$$

where $i = 1, \ldots, N,\ j = 1, \ldots, M_i$,

$$R^i_{j,m^*} = \begin{cases} P^N_{j,m^*} R, & i = N, \\ \sum\limits_{k=0}^{m^*} p^i_{j,k} R^{i+1}_{m^*-k}, & i = 1, \ldots, N-1, \end{cases} \tag{6.12}$$

and

$$V^i_{j,m^*} = \begin{cases} \left(1 - P^N_{j,m^*}\right) V, & i = N, \\ \left(1 - P^i_{j,m^*}\right) R^{i+1}_0, & i = 1, \ldots, N-1, \end{cases} \tag{6.13}$$

where $j = 1, \ldots, M_i,\ k = 0, \ldots, m^*$.

While applying formulae (6.11)–(6.13), the corresponding decisions (actions) $A^*$ can be obtained by storing the maximum arguments evaluated as

$$A^i_{m^*} = \operatorname*{argmax}_{j=1,\ldots,M_i} \Big\{ -c^i_j + R^i_{j,m^*} - V^i_{j,m^*} \Big\},\ i = 1, \ldots, N.$$

The optimal decisions could be represented by a lookup–table, and a graphical example of a lookup–table for the sequential workflow with $N = 4$ tasks is shown in Figure 6.3. The horizontal axis corresponds to the time budget left until the overall deadline is breached, while the vertical axis corresponds to the position of the abstract service within the chain. The colour corresponds to the decision that has to be taken, e.g. proceed with the alternative $j$. We illustrate the lookup table with the following examples:

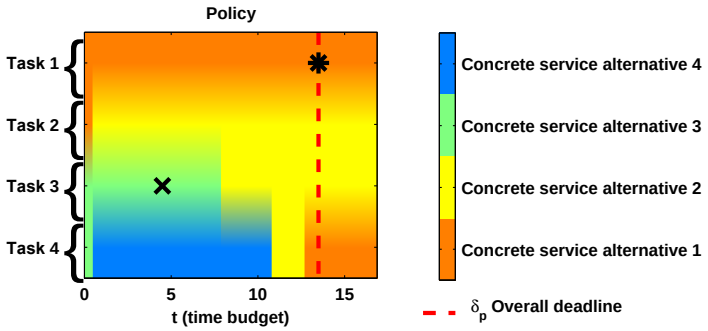


Figure 6.3: Graphical example of a decision table. The vertical dashed line represents the overall deadline $\delta_p$. Symbols $*$ and $\times$ represent decisions for different values of remaining time–to–deadline.

1. We start handling a new request and the overall deadline equals $\delta_p = 13.5$. The decision is marked by an asterisk $*$ at the lookup table shown in Figure 6.3, and alternative 1 is to be selected.
2. We have 4.9 time units remaining the decision is made for the abstract service at position 3. The decision is marked by a cross $\times$, and points that concrete service alternative 3 should be selected.

## 6.5 Numerical experiments

In this section we investigate the influence of the system parameters on the potential gain that can be obtained by applying dynamic service composition as compared to static service composition. To this end, we have performed a wide variety of numerical experiments. The results of these experiments are outlined below.

The parameters of the models discussed in Sections 3 and 4, and their corresponding value ranges, are listed in Table 6.1. The parameter space is large and prohibits an exhaustive analysis of model instances. Therefore, we have defined a number of specific model instances that represent certain characteristics of service compositions. First, we study the impact of the input parameters reward, penalty and cost on the potential gain in expected revenue obtained by dynamic versus static service composition. Second, we investigate the impact of the length of the service chain on the potential gain. Third, we consider the impact of the number of available alternatives for abstract services, and their corresponding positions in the service chain, on the expected gain. For convenience, we assume that the response–time distributions for the $j$–th alternative of abstract service $i$ can be approximated by a log–normal distribution with mean $\mu_j^i$ and variance ${\sigma_j^i}^2$, see Table 6.1. We emphasize that this assumption is not restrictive, because our approach supports *all*

Table 6.1: Overview of model parameters

| Parameter | Definition |
|---|---|
| $N$ | Number of abstract services |
| $M_i$ | Number of concrete alternatives for abstract service $i$ |
| $f_j^i$ | PDF of response–time distribution $\mathrm{CS}_j^i$ |
| $\mu_j^i$ | Mean response time of $\mathrm{CS}_j^i$ |
| ${\sigma_j^i}^2$ | Variance of response time of $\mathrm{CS}_j^i$ |
| $c_j^i$ | Cost of invocation of $\mathrm{CS}_j^i$ |
| $\delta_p$ | End–to–end deadline |
| $p_{e2e}$ | Required fraction of responses within the deadline |
| $R$ | Reward per successful request within deadline $\delta_p$ |
| $V$ | Penalty per request not completed within deadline |
| $K_R$ | Scaled reward |
| $K_V$ | Scaled penalty |

non–negative probability distributions.

The *gain* $G$ in expected revenue obtained by using optimal dynamic composition compared to the optimal static composition is defined as follows:

$$G := \frac{R^*_{\text{dynamic}} - R^*_{\text{static}}}{R^*_{\text{static}}} \times 100\%, \tag{6.14}$$

where $R^*_{\text{dynamic}}$ is the expected revenue per request under the optimal dynamic policy, obtained by applying the algorithm described in 6.4. Moreover, $R^*_{\text{static}}$ is the expected revenue per request under the optimal static policy, obtained by an exhaustive search of all possible "paths" traversing the chain of abstract services (see for example Figure 6.2, where there are 24 possible paths).

For our parameter study we need to choose a deadline $\delta_p$. This has to be done such that experiments generate sensible results. Therefore we introduce the reference workflow $\boldsymbol{W}_{\text{ref}}$. Using the reference workflow we can relate a sensible deadline to a given end–to–end objective $p_{e2e}$. The reference workflow $\boldsymbol{W}_{\text{ref}}$ is determined by taking

$$\begin{aligned} \boldsymbol{W}_{\text{ref}} &= \operatorname*{argmax}_{\boldsymbol{W}} \left[ R p_{\boldsymbol{W}} - V(1 - p_{\boldsymbol{W}}) - c_{\boldsymbol{W}} \right], \\ c_{\boldsymbol{W}} &:= \sum_{i=1}^{N} c^i_{w_i}, \\ p_{\boldsymbol{W}} &:= \mathbb{P}(D_{\boldsymbol{W}} \leq \delta_p). \end{aligned} \tag{6.15}$$

For given $p_{e2e}$, we denote by $\delta_p$ the deadline such that for the reference composition $\boldsymbol{W}_{\text{ref}}$, consisting of $\mathrm{CS}_1^i,\ i = 1, \ldots, N$, holds that: $\mathbb{P}(D_{\text{ref}} < \delta_p) = p_{e2e}$.

Table 6.2: Parameter settings for exploratory example

| Abstract service i → | Service 1 | | | Service 2 | | |
|---|---|---|---|---|---|---|
| Concrete service j ↓ | $c_j^i$ | $\mu_j^i$ | ${\sigma_j^i}^2$ | $c_j^i$ | $\mu_j^i$ | ${\sigma_j^i}^2$ |
| Alternative 1 | 1 | 5 | 4 | 1 | 5 | 4 |
| Alternative 2 | N.A. | N.A. | N.A. | 5 | 2.5 | 4 |

### 6.5.1 Impact of reward, penalty and cost parameters

In this subsection we study the impact of concrete service cost $c_j^i$, the reward $R$ and the penalty $V$ on the potential gain $G$, defined in (6.14). To this end, we define the overall mean service cost by

$$\overline{c} = \sum_{i=1}^{N} \overline{c}_i, \text{ where } \overline{c}_i = \frac{1}{M_i} \sum_{j=1}^{M_i} c_j^i. \tag{6.16}$$

Note that $R$, $V$ and $\overline{c}$ are scale–invariant, in the sense that if these parameters are scaled to $\alpha R$, $\alpha V$ and $\alpha \overline{c}$ for some $\alpha > 0$, then the optimal dynamic and static policies, and hence also the gain $G$, remain the same. Therefore, it is convenient to define the following two scaled parameters, the scaled reward $K_R$ and the scaled penalty $K_V$ parameters:

$$K_R := \frac{R}{\overline{c}}, \qquad K_V := \frac{V}{\overline{c}}. \tag{6.17}$$

Consider the following exploratory example with $N = 2$, $M_1 = 1$ and $M_2 = 2$, depicted in Figure 6.4, and where the cost parameters $c_j^i$ and the distribution parameters $\mu_j^i$ and ${\sigma_j^i}^2$ are listed in Table 6.2. Figure 6.5 shows the contour lines

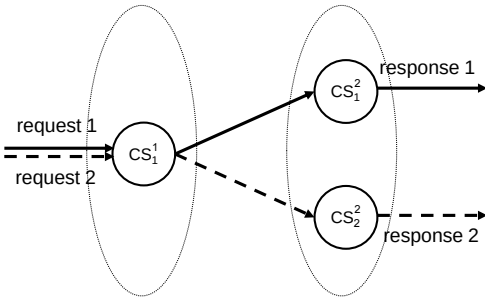


Figure 6.4: Exploratory example.

of combinations $(K_R, K_V)$ that lead to the same expected revenue, subject to the constraint that the reference *static* workflow (i.e., the workflow $(1, 1)$) leads to an end–to–end response time less than the deadline with probability $p_{e2e} = 80\%$. The

values plotted on the dotted lined indicate the expected revenue. The grey area is the set of combinations for which the static workflow is $(1,1)$, indicated by the solid line in Figure 6.4. The white area represents the combinations for which the static workflow $(1,2)$ is optimal, indicated by the dashed line in Figure 6.4. Note that in these cases $p_{e2e} > 80\%$, because one chooses a faster alternative while maintaining the same deadline. Note that all iso–curves in Figure 6.5 are piecewise linear. More

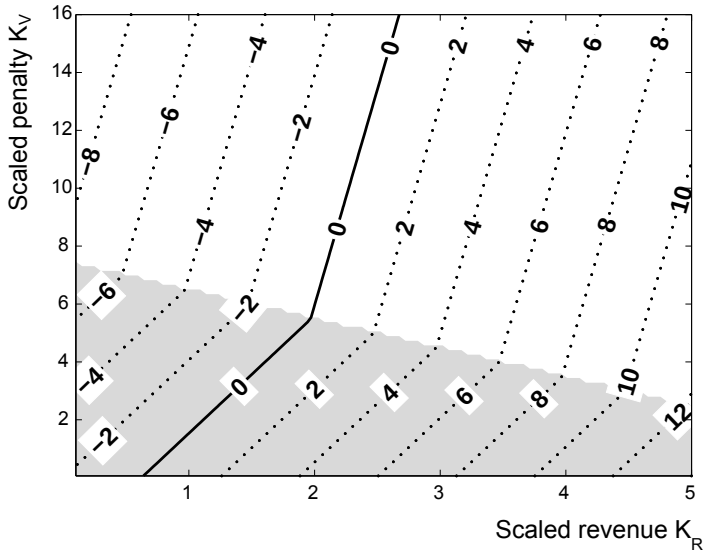


Figure 6.5: Expected revenue in static setup with $p_{e2e} = 80\%$. All contour lines represent equal expected revenue for the static optimal composition. The grey and white area correspond to the different optimal static compositions.

precisely, one may verify using (6.6) that for a given optimal static workflow the iso–curves are of the form $K_V = aK_R + b$, with

$$a = \frac{p_{\boldsymbol{W}}}{1 - p_{\boldsymbol{W}}}, \quad \text{and} \quad b = \frac{R^*_{\text{static}} + c_{\boldsymbol{W}}}{\overline{c}(p_{\boldsymbol{W}} - 1)}, \tag{6.18}$$

where $R^*_{\text{static}}$ is defined in (6.14), and where $c_{\boldsymbol{W}}$ is the cost for the optimal static workflow $\boldsymbol{W}$, $p_{\boldsymbol{W}}$ is probability a request using $\boldsymbol{W}$ will be within deadline $\delta_p$ and $\boldsymbol{W}$ is the optimal static workflow for which holds that $\boldsymbol{W} = (1,1)$ in the grey area and $\boldsymbol{W} = (1,2)$ in the white area. It is readily verified from (6.6) that the switching curve of the optimal workflow (i.e., the border separating the grey and the white area) is given by the following equation

$$K_V + K_R = \frac{c_{(1,1)} - c_{(1,2)}}{\overline{c}(p_{(1,1)} - p_{(1,2)})}, \tag{6.19}$$

where $c_{(1,k)}$ is the cost for workflow $(1,k)$ $(k=1,2)$ as defined in ((6.5)), $\overline{c}$ is defined in (6.16) and $p_{(1,k)}$ is the probability that the response time meets the deadline if the static workflow $(1,k)$ is optimal. Figure 6.6 shows the results for the dynamic

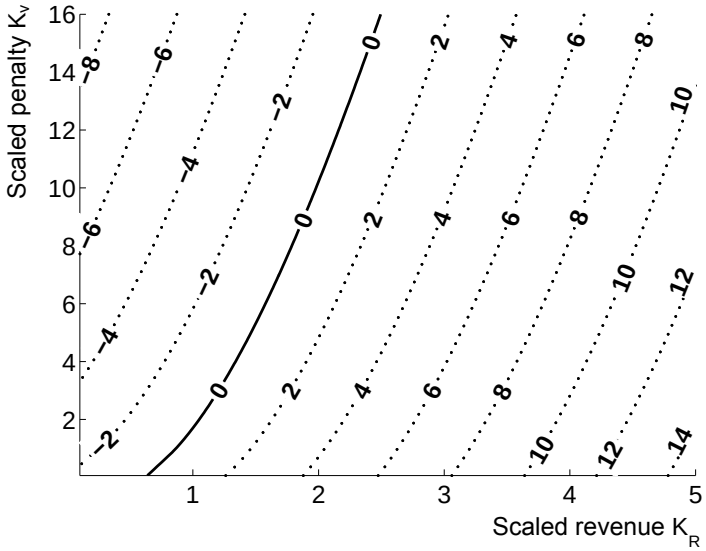


Figure 6.6: Expected revenue in dynamic setup with $p_{e2e} = 80\%$. All contour lines represent equal expected revenue for the optimal dynamic composition.

counterpart of Figure 6.5, and Figure 6.7 shows the iso–curves for the relative gain $G$, defined in (6.14). Most remarkably, the iso–curves depicted in Figure 6.7 are wedge–shaped with a sharp angle at the switching curve (6.19). This suggests that the relative gain $G$ has the highest values at the switching curve. In other words, the information about the response times at the first abstract service is particularly crucial to decide about the optimal path.

The solid line is the contour line over the combinations $(K_V, K_R)$ where $R^*_{\text{dynamic}} = 0$, and the dashed line show the combinations for which $R^*_{\text{static}} = 0$. We observe that the distance between these two lines is maximal on the switching curve, as expected.

Figures 6.8 to 6.10 show the same results as in Figures 6.5 to 6.7, but with $p_{e2e} = 90\%$. The results show similar behaviour to those in Figures 6.5 to 6.7.

Table 6.3: Cost and distribution parameters

| Abstract service i → | Service $i \notin \{2, N\}$ | | | Service $i \in \{2, N\}$ | | |
|---|---|---|---|---|---|---|
| Concrete service j ↓ | $c_j^i$ | $\mu_j^i$ | ${\sigma_j^i}^2$ | $c_j^i$ | $\mu_j^i$ | ${\sigma_j^i}^2$ |
| Alternative 1 | 1 | 5 | 16 | 1 | 5 | 16 |
| Alternative 2 | N.A. | N.A. | N.A. | 3 | 2.5 | 4 |
| Alternative 3 | N.A. | N.A. | N.A. | 9 | 1.25 | 1 |
| Alternative 4 | N.A. | N.A. | N.A. | 27 | 0.675 | 0.25 |

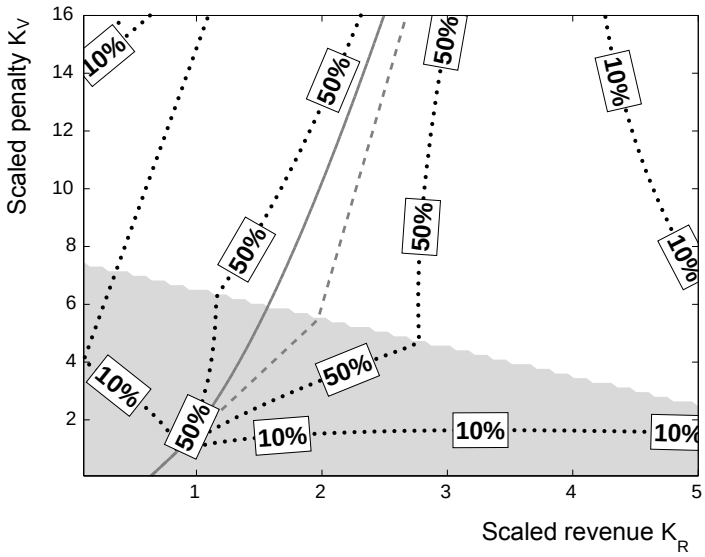


Figure 6.7: Relative gain $G$ of expected revenue with $p_{e2e} = 80\%$. The contour lines represent equal relative gain. The grey and white area correspond to the different optimal static compositions. The solid contour line is where $R^*_{\text{dynamic}} = 0$, the dashed contour line is where $R^*_{\text{static}} = 0$.

### 6.5.2 Impact of number of alternatives

We will now investigate the impact of the number of alternatives for each of the executed tasks on the obtained gain $G$ when using dynamic composition. To this end, we consider a workflow of length $N \geq 3$, with $M_i = 1$ for $i \neq 2, N$, and where $M_2$ and $M_N$ are varied as $\{1, 2, 3, 4\}$. The cost and distribution parameters are listed in Table 6.3, and moreover, we take $R = 26$, and $V = 130$. Note that it is readily verified from (6.16) that $\overline{c} = 26$, and from (6.17) that $K_R = 1$ and $K_V = 5$.

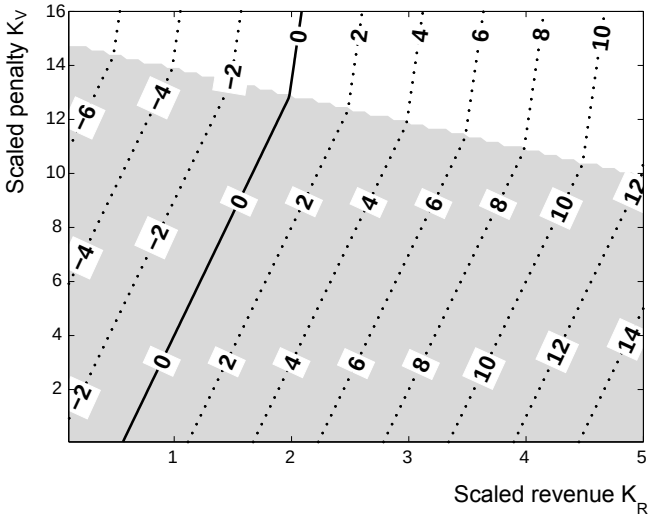


Figure 6.8: Expected revenue in static setup with $p_{e2e} = 90\%$. All contour lines represent equal expected revenue for the static optimal composition. The grey and white area correspond to the different optimal static compositions.

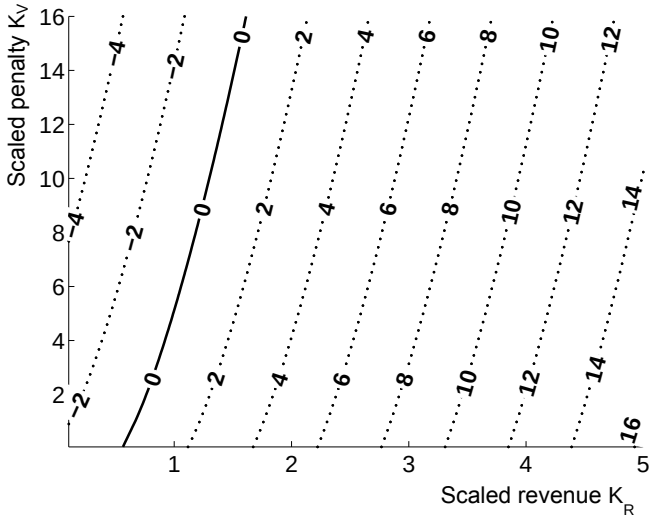


Figure 6.9: Expected revenue in dynamic setup with $p_{e2e} = 90\%$. All contour lines represent equal expected revenue for the optimal dynamic composition.

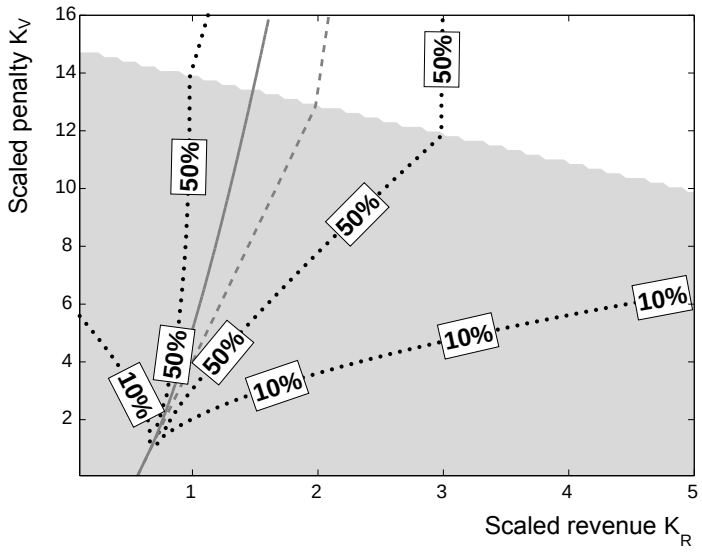


Figure 6.10: Relative gain $G$ of expected revenue with $p_{e2e} = 90\%$. The contour lines represent equal relative gain. The grey and white area correspond to the different optimal static compositions. The solid contour line is where $R^*_{\text{dynamic}} = 0$, the dashed contour line is where $R^*_{\text{static}} = 0$.

Figure 6.11 illustrates an example with $M_2 = 2$ and $M_N = 3$. For each of the model

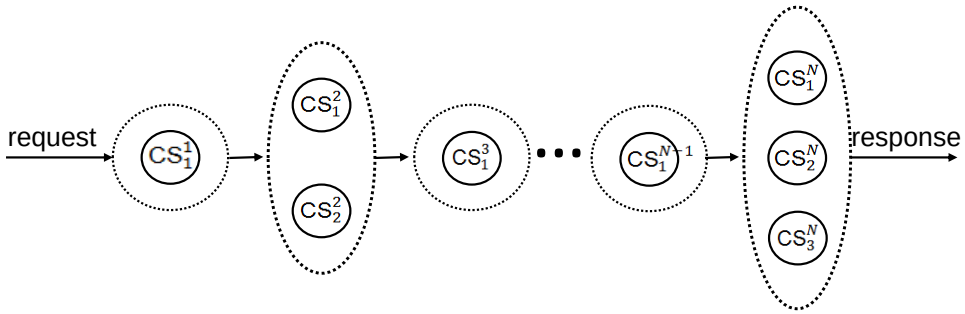


Figure 6.11: Configuration for analysis on the impact of alternatives' position for the second and the last task in the workflow.

instances, we have calculated (1) the optimal static composition and its corresponding expected revenue $R^*_{\text{static}}$, (2) the optimal dynamic composition and its corresponding expected revenue $R^*_{\text{dynamic}}$, and (3) the relative gain $G$, defined in (6.14).

In particular we expect that, due to increasing uncertainty when traversing a service composition workflow, more alternatives are desired at the end of the workflow. For our analyses we define a sequential workflow of length $N$ where the number of service alternatives is varied for the second and the last task in the workflow. Using this setup we can compare the relative value of alternatives at the beginning and at the end of the workflow. The values that have been chosen are presented in Table 6.3.

Table 6.4: Impact on availability of alternatives

| Gain | | $N_2$ | | | |
|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 |
| $N_8$ | 1 | 0.0% | 0.0% | 0.4% | 0.4% |
| | 2 | 16.6% | 14.6% | 14.8% | 14.8% |
| | 3 | 37.1% | 25.1% | 25.3% | 25.3% |
| | 4 | 42.1% | 27.6% | 27.8% | 27.8% |

| dynamic revenue | | $N_2$ | | | |
|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 |
| $N_8$ | 1 | 2.44 | 5.43 | 5.45 | 5.45 |
| | 2 | 6.33 | 8.21 | 8.23 | 8.23 |
| | 3 | 7.44 | 8.97 | 8.98 | 8.98 |
| | 4 | 7.71 | 9.15 | 9.16 | 9.16 |

| Static revenue | | $N_2$ | | | |
|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 |
| $N_8$ | 1 | 2.44 | 5.43 | 5.43 | 5.43 |
| | 2 | 5.43 | 7.17 | 7.17 | 7.17 |
| | 3 | 5.43 | 7.17 | 7.17 | 7.17 |
| | 4 | 5.43 | 7.17 | 7.17 | 7.17 |

Results of the case where $N = 8$ and $p_{e2e} = 90\%$ for the reference composition are represented in Table 6.4. Increasing the number of alternatives for the second task in the workflow results in a negligible relative gain increase. However, increasing the number of alternatives for the last task results in an increase of gain up to 42.1%. If we take a closer look to static expected revenue in Table 6.4 we observe that the third and fourth alternative are never used as the expected revenue does not increase when these alternatives are available.

Furthermore we observe that the table is symmetric for the number of alternatives at positions two and eight. This is caused by the fact that workflows in the static scenario are fixed and therefore, if the alternatives at both positions are "similar", the position where the number alternatives is increased has no effect on expected revenue. However, for the dynamic composition the position where the number of alternatives is increases has a dramatic effect.

For the case with dynamic service composition, we see from the results presented in Table 6.4 that for the last task the availability of alternatives three and four has a significant impact on revenue. For the second task the availability of mentioned alternatives has hardly any effect as the advantage of optimal dynamic selection is averaged out over the successive workflow response times. For the last task, the dynamic service composition enables the orchestrator to take the advantage of the third and the fourth alternative where the static service composition does not optimally use the second alternative and never uses the third and the fourth alternative.

### 6.5.3 Length of the composition

In subsection 6.5.2 we observed that the effect of alternatives at the start of the workflow can vanish due to averaging of response times over the succeeding services. In this subsection we investigate the effect of the composition length. The setup used for the experiments is depicted in Figure 6.12 and the workflow has $N$ sequential tasks. There is one alternative for the first task ($M_1 = 1$) while there are two alternatives for tasks $i \geq 2$ ($M_i = 2$ for $i = 2, \ldots, N$). The parameter space is large, therefore we limit us to compositions where orchestrator selects among two alternatives for each task in the workflow. Concrete service alternative parameters are defined in Table 6.5. Reward parameter $R$ and penalty parameter $V$ are related to overall mean service cost $\bar{c}$ using scaled reward $K_R$ and scaled penalty $K_V$ as defined in (6.17). Deadline $\delta_p$ is chosen using (6.15).

The set of service alternatives is identical for all tasks in the workflow. Therefore, there are $N$ different static compositions as for such composition the actual position of alternatives does not matter. We define a workflow for a system with $N$ tasks as $\boldsymbol{W}_k$, $k = 0, \ldots, N-1$ where $k$ represents the number of faster alternatives used in the composition. In our case reference composition becomes $\boldsymbol{W}_{\text{ref}} = \boldsymbol{W}_0 = (1, 1, \ldots, 1)$, i.e. a composition that consists of regular (slow) alternatives only. We choose to vary parameters $K_R$ and $K_V$ in the ranges $K_R \in (0; 5]$ and $K_V \in (0; 50]$. Figures

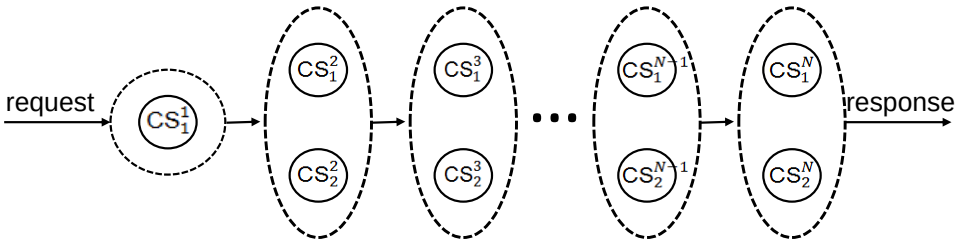


Figure 6.12: Configuration for analysis of the impact of workflow length.

6.13–6.14 contain results for workflow lengths 3 and 20, respectively. The end–to–end probability for the reference composition is $p_{e2e} = 90\%$. In these figures different

Table 6.5: Cost and distribution parameters for impact of chain length.

| Abstract service i → | Service $i \notin \{2, N\}$ | | | Service $i \in \{2, N\}$ | | |
|---|---|---|---|---|---|---|
| Concrete service j ↓ | $c_j^i$ | $\mu_j^i$ | ${\sigma_j^i}^2$ | $c_j^i$ | $\mu_j^i$ | ${\sigma_j^i}^2$ |
| Alternative 1 | 1 | 5 | 16 | 1 | 5 | 16 |
| Alternative 2 | N.A. | N.A. | N.A. | 3 | 2.5 | 4 |

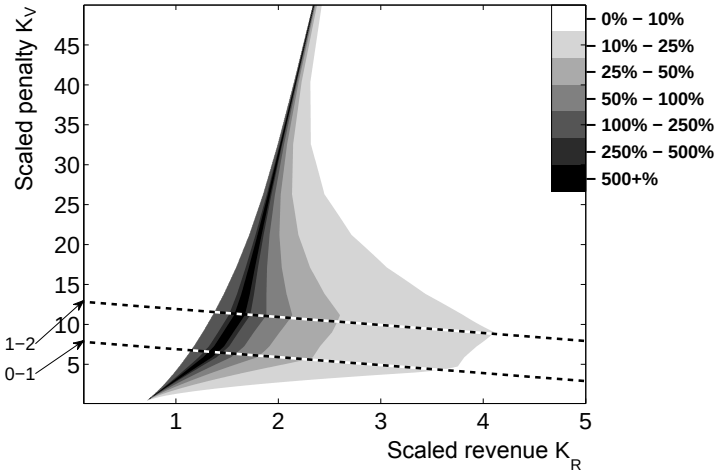


Figure 6.13: Contour plot for relative gain $G$ with $N = 3$ and $p_{e2e} = 90\%$. The grey–scale areas correspond to a specific range of relative gain $G$ as specified in the colour bar. The arrows identify the (dashed) lines that separate different optimal static policies, i.e. the number of faster alternatives used for the composition.

grey–scale levels of the contour areas identify different revenue gain $G$. We observe wedge shapes at the edges of the grey–scale contour areas. These wedges correspond to the switching curves where the static policy adds another faster alternative to the workflow. Dashed lines identify the switching curves between different optimal static workflows. Arrows identify what static optimal paths are separated. Switching curves could be expressed as the following:

$$K_V + K_R = \frac{c_{\boldsymbol{W}_k} - c_{\boldsymbol{W}_{k+1}}}{\overline{c}(p_{\boldsymbol{W}_k} - p_{\boldsymbol{W}_{k+1}})}, \tag{6.20}$$

which is similar to (6.19). Probability $p_{\boldsymbol{W}_k}$ is the probability that a static composition with $k$ faster alternatives will generate a response within deadline $\delta_p$. When the workflow length $N$ increases the grey–scale contour areas become wider. This implies that longer workflows potentially have a higher revenue gain $G$ when comparing dynamic and static workflows.

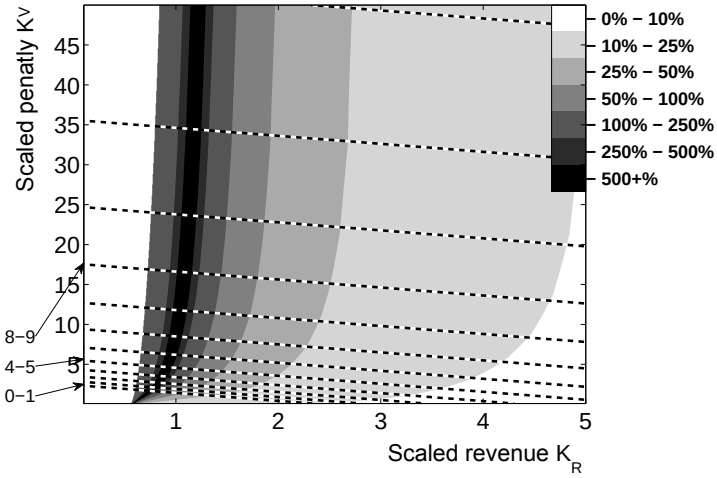


Figure 6.14: Contour plot for relative gain $G$ with $N = 20$ and $p_{e2e} = 90\%$. The grey–scale areas correspond to a specific range of relative gain $G$ as specified in the colour bar. The arrows identify the (dashed) lines that separate different optimal static policies i.e. the number of faster alternatives used for the composition.

### 6.5.4 Arbitrary alternatives

We consider a service workflow illustrated at Figure 6.15 for our experiments. This sequential workflow consists of $N = 4$ tasks. For each task $i$, there are $M_i$ service alternatives, where $M_i$ could take one of the values $\{1, 2, 3, 4\}$, and $M_i \neq M_j$ whenever $i \neq j$. The notation $(M_1, M_2, M_3, M_4)$ depicts the particular experiment setup, and represents one of the possible permutations of the set $\{1, 2, 3, 4\}$. Therefore, there are in total 24 different compositions (experimental setups) to be considered. Let $\boldsymbol{W}_{SW} = (w_1, w_2, w_3, w_4)$ represent the service composition for the Static Workflow (SW) strategy, where $1 \leq w_j \leq M_j,\ j = 1, 2, 3, 4$. We want to compare the SW and Dynamic Programming (DP) strategies with fixed scaled parameters $K_V$ and $K_R$. We choose to dimension the deadline $\delta_p$ such that the optimal static composition has exactly a fraction of requests served within the end–to–end deadline $\delta_p$ that equals to $p_{e2e}$. We can determine the optimal static composition "path" and so determine the reward and penalty parameters such that the expected revenue for the static composition equals a predefined value. The expected reward per request for the SW strategy $R^*_{SW}$ is given by

$$R^*_{SW} = R \cdot p_{e2e} - V \cdot (1 - p_{e2e}) - c_{\boldsymbol{W}_{SW}}. \tag{6.21}$$

This expression is similar to (6.6) with workflow cost $c_{\boldsymbol{W}_{SW}}$ as defined in (6.5). We use the SW strategy for the benchmarking and we set the value $R^*_{SW} = 0.01$.

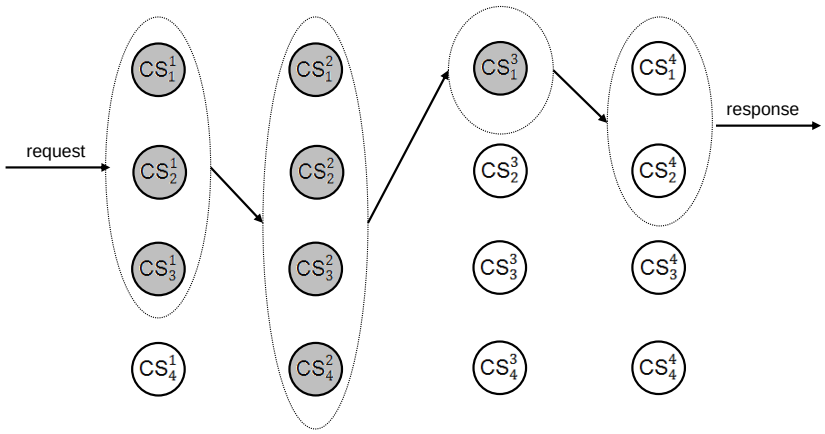


Figure 6.15: Example workflow where the number of alternative services are represented by the permutation $(M_1, M_2, M_3, M_4) = (3, 4, 1, 2)$.

Taking into account (6.21) the values for parameters $R$ and $V$ satisfy the following equations:

$$R = \frac{R^*_{SW} + c_{\boldsymbol{W}_{SW}} + K_V \cdot (1 - p_{e2e}) \cdot \overline{c}}{p_{e2e}}, \tag{6.22}$$

$$V = \frac{R^*_{SW} + c_{\boldsymbol{W}_{SW}} - K_R \cdot p_{e2e} \cdot \overline{c}}{p_{e2e} - 1}. \tag{6.23}$$

We have conducted simulations for two general scenarios, symmetric and asymmetric. These scenarios are defined based on the selection of cost parameters, as well as $\mu$ and $\sigma^2$ for a concrete service.

The goal of symmetric scenario simulations is to illustrate the importance of the position of, and number of, service alternatives within the workflow. The choice of parameters for symmetric scenario is given in Table 6.6; here the parameters of the concrete service alternatives are the same for all abstract services. When the number of alternatives for abstract service is e.g. 2, we always consider alternative 1 and alternative 2 in our experiments.

In the asymmetric scenario which corresponds to parameter Table 6.7, the concrete services have different mean response times for abstract services at different positions in the service composition chain. For example, service $\mathrm{CS}_1^2$ has execution cost $c_{2,1} = 5$, mean $\mu_{2,1} = 10$ and variance $\sigma^2_{2,1} = 16$ while alternative $\mathrm{CS}_2^2$ is cheaper, i.e. $c_{2,2} = 1$, has a lower mean $\mu_{2,2} = 9.5$ but higher variance $\sigma^2_{2,2} = 64$. Furthermore, for the third task, an expensive service alternative with zero variance is added. The

Table 6.6: Concrete service alternatives symmetric scenario

| Abstract service i → | Service 1, Service 2, Service 3, Service 4 | | |
|---|---|---|---|
| Concrete service j ↓ | $c_j^i$ | $\mu_j^i$ | ${\sigma_j^i}^2$ |
| Alternative 1 | 1 | 5 | 4 |
| Alternative 2 | 5 | 2.5 | 4 |
| Alternative 3 | 10 | 1.25 | 16 |
| Alternative 4 | 50 | 0.5 | 0.0009 |

Table 6.7: Concrete service alternatives asymmetric scenario

| Abstract service i → | Service 1 | | | Service 2 | | |
|---|---|---|---|---|---|---|
| Concrete service j ↓ | $c_j^i$ | $\mu_j^i$ | ${\sigma_j^i}^2$ | $c_j^i$ | $\mu_j^i$ | ${\sigma_j^i}^2$ |
| Alternative 1 | 1 | 5 | 4 | 5 | 10 | 16 |
| Alternative 2 | 5 | 2.5 | 4 | 1 | 9.5 | 64 |
| Alternative 3 | 10 | 1.25 | 16 | 10 | 1.5 | 0.25 |
| Alternative 4 | 50 | 0.5 | 0.0009 | 50 | 1 | 0.0025 |
| Abstract service i → | Service 3 | | | Service 4 | | |
| Concrete service j ↓ | $c_j^i$ | $\mu_j^i$ | ${\sigma_j^i}^2$ | $c_j^i$ | $\mu_j^i$ | ${\sigma_j^i}^2$ |
| Alternative 1 | 1 | 0.5 | 0.04 | 1 | 2.5 | 1 |
| Alternative 2 | 5 | 0.4 | 0.04 | 5 | 2.45 | 2.25 |
| Alternative 3 | 10 | 0.3 | 0.0025 | 10 | 1 | 4 |
| Alternative 4 | 100 | 0.05 | 0 | 50 | 0.25 | 0.0004 |

goal of the asymmetric scenario is to illustrate the importance of the variance in addition to mean and cost for the service selection, which is usually neglected in state–of–the–art service composition solutions.

The calculated values for reward and penalty parameters (and given $p_{e2e} = 90\%$) are then used for the simulations of the DP strategy, for both symmetric and asymmetric scenarios.

### 6.5.5 Results

For all possible permutations of number of concrete service alternatives (see Table 6.8) the expected revenue $\mathbb{E}[R]$ is calculated for both symmetric and asymmetric scenarios and DP and SW strategies. The results are summarized in Figure 6.16 (symmetric scenario) and Figure 6.17 (asymmetric scenario). For both figures, the alternative configurations are ranked upon the expected revenue for the DP strategy. In Figure 6.16 we observe that the DP solution achieves the lowest revenue for the permutation $(M_1, M_2, M_3, M_4) = (4, 3, 2, 1)$. In this case there are many

Table 6.8: Indices of alternative configurations

| label | a | b | c | d | e | f | g | h | i | j | k | l |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| position 1 | 4 | 3 | 4 | 2 | 3 | 2 | 4 | 3 | 4 | 1 | 3 | 1 |
| position 2 | 3 | 4 | 2 | 4 | 2 | 3 | 3 | 4 | 1 | 4 | 1 | 3 |
| position 3 | 2 | 2 | 3 | 3 | 4 | 4 | 1 | 1 | 3 | 3 | 4 | 4 |
| position 4 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| label | m | n | o | p | q | r | s | t | u | v | w | x |
| position 1 | 4 | 2 | 4 | 1 | 2 | 1 | 3 | 2 | 3 | 1 | 2 | 1 |
| position 2 | 2 | 4 | 1 | 4 | 1 | 2 | 2 | 3 | 1 | 3 | 1 | 2 |
| position 3 | 1 | 1 | 2 | 2 | 4 | 4 | 1 | 1 | 2 | 2 | 3 | 3 |
| position 4 | 3 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 |

alternatives for the first task and no alternatives for the last task which results in no possibility to recover from (possible) large service response time accumulated at the beginning of the workflow. Further, DP–based solution makes always the same service selection for the first task in the workflow, not taking any advantage of available alternatives for this service. On the other hand, the highest revenue for the DP solution is achieved for the permutation $(M_1, M_2, M_3, M_4) = (1, 2, 3, 4)$. The most alternatives are then available for the last workflow task, thus increasing the possibility to recover from (possible) large response times accumulated at the beginning of the workflow.

From Figure 6.16 seven regions can be identified labelled by capital letters $A, B, C, D, E, F, G$. Each region is separated by line where the configuration change resulted in a significant increase in expected revenue. The changes that correspond to the most significant increase in revenue are listed in Table 6.9. In this Table labels $M_3$ and $M_4$ correspond to the number of alternatives available for the last two workflow tasks. We observe from Figure 6.16 that six configurations in regions $F$ and $G$ with the highest revenue are those where the number of alternatives is the highest for the last, fourth workflow task. The number of alternatives is in given example four.

Configurations in region $G$ perform better than configurations in region $F$ as the number of alternatives for the third task is three in region $G$, and either one or two in region $F$. Additionally, the revenue "jump" between regions $E$ and $F$ is due to the number of alternatives (3 and 4, respectively) for the fourth task. The "jump" between regions $D$ and $E$ is caused by the fact that in region $D$ for there are only alternatives 1 and 2 available for the third task, while for region $E$ four service alternatives are available for the same, third task.

The largest revenue improvement for DP strategy is achieved when for the last task, service alternative four is considered, see Table 6.6. When this alternative with low mean and variance is considered, enough certainty to proceed with the workflow execution exists and the high price of this service will be compensated

by the increase in expected revenue. Another (smaller) revenue increase can be observed when service alternative 3 becomes available for the third task. This can be explained by the fact that the 90–th percentile for alternative 3 is still lower than for alternatives 1 and 2, despite its higher variance, see Table 6.10. In Figure 6.17

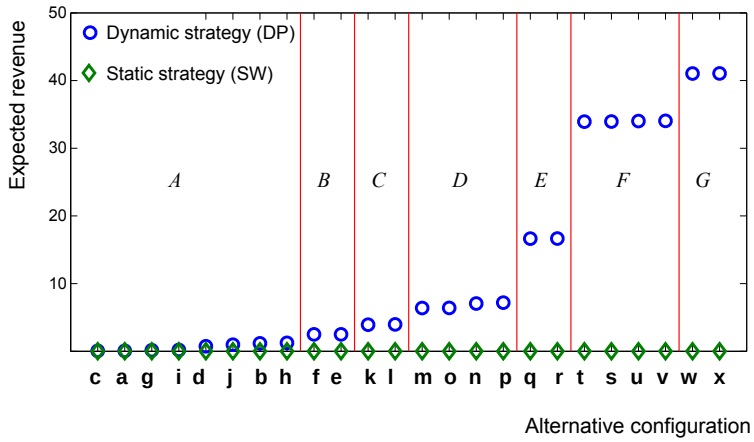


Figure 6.16: Expected revenues per request in case of the symmetric scenario. The results are presented for both static (SW) and dynamic (DP) strategies. See Table 6.9 for the characteristics of the regions $D, E, F, G$.

Table 6.9: Symmetric scenario regions from Figure 6.16.

| Region | $D$ | $E$ | $F$ | $G$ |
|---|---|---|---|---|
| $M_3$ | $< 3$ | 4 | $< 3$ | 3 |
| $M_4$ | 3 | 3 | 4 | 4 |

Table 6.10: The 90th percentile of service alternatives.

| Alternative | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Percentile | 7.61 | 4.81 | 2.74 | 0.54 |
| $\mu$ | 5 | 2.5 | 1.25 | 0.5 |
| $\sigma$ | 2 | 2 | 4 | 0.03 |

the results are given for the asymmetric scenario. For the asymmetric scenario it is harder to observe any structure, because the different alternatives have different impact on the response time and the DP takes advantage of the properties of all service alternatives. The lowest expected revenue is achieved for the configuration $(M_1, M_2, M_3, M_4) = (4, 3, 2, 1)$ while the highest expected revenue is achieved for

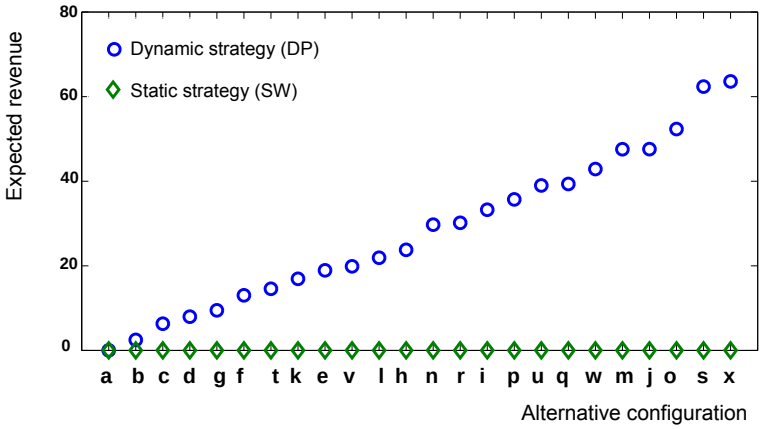


Figure 6.17: Expected revenues per request in case of the asymmetric scenario. The results are presented for both static (SW) and dynamic (DP) strategies.

the configuration $(M_1, M_2, M_3, M_4) = (1, 2, 3, 4)$. The largest revenue increase is achieved when the near–zero variance service alternative is considered for the last task and when the cheaper second alternative for the second task is considered for selection despite its higher variance. In Figure 6.18 we show the comparison of the

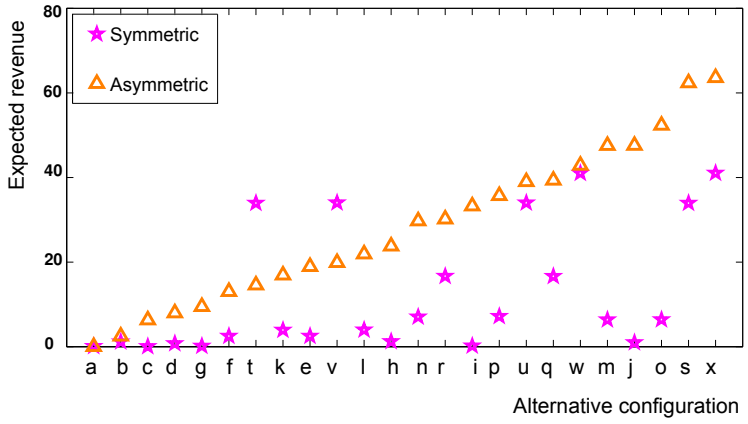


Figure 6.18: Revenue comparison between symmetric and asymmetric scenarios.

rewards for the symmetric and asymmetric scenarios. The comparison is done when the same service configuration is used for both scenarios. The indices of service

configuration are shown in Table 6.8.

The main conclusions and some "rules of a thumb" that could be drawn from the Figures 6.16 – 6.18 are as the following:

- Dynamic, on–the–fly service composition results in higher revenues for the CSP, compared to optimal "static" service composition (Figures 6.16, 6.17). While the expected reward per request for the SW strategy is 0.01, for the symmetric and asymmetric DP scenarios, the expected rewards per request, depending of the configuration, may be greater than 40 or greater than 60, respectively.
- It is more beneficial to have higher number of concrete service alternatives closer to the end of the sequential workflow (Figures 6.16, 6.17).
- The variability of the response times may have significant impact to the revenues achieved. When the end–to–end deadline is in jeopardy, it may be better to have more expensive service with small response–time variability (and smaller mean) than less expensive one with large response–time variability (Figure 6.17).
- It is in general better to have more response–time versatility with respect to mean and variance, (Figure 6.18). However, this needs to be investigated further.

## 6.6 Conclusions

In this chapter we have developed a model to maximize revenue for composite services by on–the–fly dynamic service selection. The selection decision for a specific task in the workflow is based on the response–time characteristics of the service alternatives for the current and subsequent tasks, the remaining time to end–to–end deadline, services' execution costs, and the reward and penalty parameters. The results do not only indicate *that* there is enormous potential gain compared to other, non–dynamic approaches, but also show *how* one can realize such gain. In particular, we have shown that the optimal service selection policy can be implemented as a lookup table, which also means negligible decision–making time. We believe that the work presented in this chapter is a significant step in realizing cost–efficient provisioning of complex composite services.

An interesting and practically useful extension of the model is to include the possibility of reattempts when the response time of a given individual service exceeds some threshold value. Such reattempts may be particularly beneficial when the response–time distribution has a decreasing hazard rate of failure. Investigation of the potential for cost reduction and the cost/benefit trade–off of reattempts has been addressed in this thesis in Chapter 8.

CHAPTER 7

# Optimal Selection Policies under Partial Service Availability

In this chapter *quality assurance* of composite services within SOA is addressed in the same context as in the previous chapter, i.e. *run–time* service composition and revenue maximization by the composite service provider. The quality assurance is defined as the probability that end–to–end deadline will be met. Motivated by practical applications, the important feature of the model considered in this chapter, compared to the model considered in Chapter 6, is that it takes into account the partial availability of services, next to the response time performance and service costs. In order to derive the selection policy that will be used at runtime, we extend the DP approach from Chapter 6 by the inclusion of the services' availability.

The main contributions of this chapter are as follows:

- We analyse the relation between the selection policy and quality assurance, and explain in particular how to derive the optimal selection policy for given target assurance. The developed service selection policy is optimal with respect to profit maximization of the composite service provider.
- We analyse the resulting end–to–end response–time distribution of the composite service when the optimal service selection policy is used. We further explain why our method results in a system, the response time of which has reduced variance with respect to the commitments of composite service provider.

The chapter is organized as follows: after overview of related work in Section 7.1, we briefly describe in Section 7.2 the model of the system under consideration. In Section 7.3 we formulate the run–time service composition problem as a dynamic programming problem and provide the corresponding backward recursion solution method used to determine the optimal decision policy. Next, in Section 7.4, we explain how to determine the end–to–end assurance and show strategies to increase it. In Section 7.5 we give the formulae for calculation of the response time distributions taking the partial availability of the services into account as well. In Section 7.6, the results of extensive numerical experiments are presented and discussed. We conclude the chapter and give directions for further research in Section 7.7.

This chapter is based on paper [106].

## 7.1 Related work

A number of solutions have been proposed for the problem of dynamic, run–time QoS–aware service selection and composition within SOA [25, 33, 44, 116]. These (proactive) solutions aim to adapt the service composition dynamically at run time. In [44] the authors analyse a problem of dynamic web service composition for different composition patterns. QoS parameters considered include reliability, availability, as well as cost and *expected value* of the response time of individual web services. The authors propose a solution based on Markov Decision Processes to minimize the expected response time, taking into account the availability and reliability of the respective services and the invocation costs. However, the authors do not consider the stochastic nature of response time, but the expected value of it. Besides, they do not consider the cost structure, revenue and penalty model assumed in this chapter.

Doshi et al. [33] present a policy–based approach for dynamically choreographing web services. The goal of the decision policy is to minimize the execution cost, while the impact of the reward and penalty on per request base are not analysed. Besides, the impact of the service availability on the revenue of the composite service provider is not addressed in this paper. Cardellini et al. [25] consider dynamic service composition in the context of admission control with various service classes. The (different) composite service configurations for the service classes can be dynamically adapted according to variations in the operating environment due to the admission or departure of users generating requests for the composite service. The authors derive an optimal admission and re–composition policy (by formulating the problem as a linear optimization problem) that maximizes the profit while guaranteeing QoS for the admitted users.

Despite the fact that each of these papers provide useful results on improving the response time performance, neither [25] nor the solution described at Chapter 6 take into account the availability of services. This observation is the main motivation for this study, as we take into account both availability and response time performance aspects. The service (un)availability may have significant impact for the end–to–end service composition and response–time commitments made by the composite service providers, as illustrated further in the chapter.

## 7.2 Sequential workflow decision model

In this section we briefly describe our model based on a composite web service represented by a sequential workflow with a simple illustration of an example workflow shown in Figure 7.1. The workflow in Figure 7.1 consists of four abstract services, and each abstract service maps to a number of concrete services (alternatives), which a deployed by (independent) third–party service providers. After the execution of a single task within the workflow, the orchestrator decides on the next concrete service

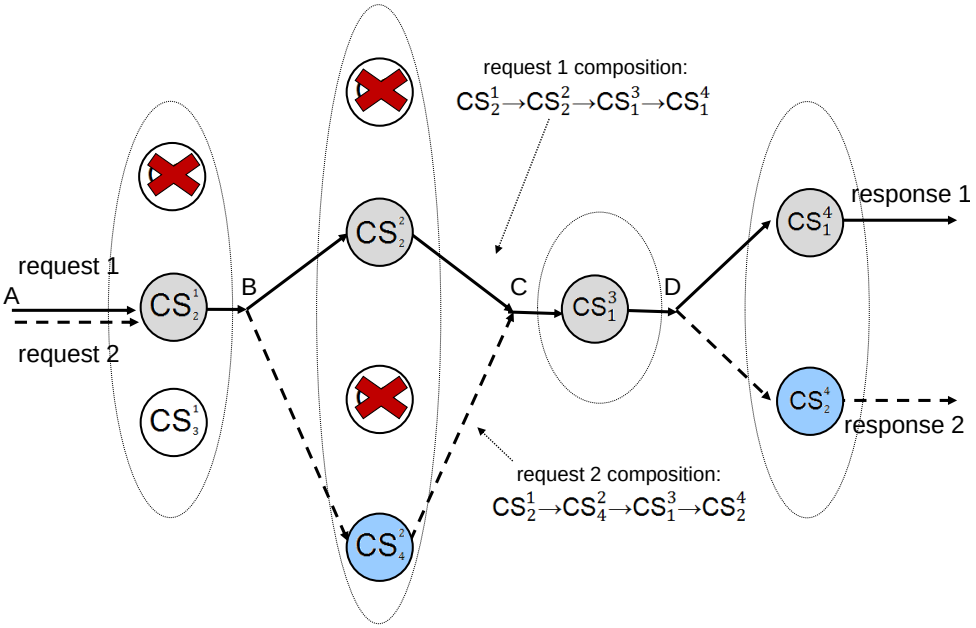


Figure 7.1: Orchestrated composite web service depicted by a sequential workflow. Dynamic run–time service composition is based on pre–calculated policy. Decisions are taken at points A, B, C and D, and e.g. it is known to the orchestrator at B which services implementing task 2 are available at the decision instance.

to be executed. The decision points for given tasks are illustrated at Figure 7.1 by A, B, C and D, respectively.

Our solution is applicable to any workflow that could be aggregated and mapped into a sequential one. Some basic rules for aggregation of a non–sequential into the sequential workflow have been illustrated in [28,115,116]. However, the aggregation leads to coarser control, since decisions could not be taken for a single service within the aggregated workflow, but rather for the aggregated workflow patterns themselves. Per single composite service request, the orchestrator executes tasks one–by–one as indicated by the sequential workflow. There are in total $N$ tasks in the workflow and the task position within workflow is indexed by $i$, $i = 1, 2, \ldots, N$. Each task $i$ maps onto $M_i \geq 1$ concrete services (alternatives), where service $j$ implementing task $i$ is denoted by $\mathrm{CS}_j^i$, see also Figure 7.1.

We denote by $c_j^i$ the cost that the composite provider pays for the single invocation of service $\mathrm{CS}_j^i$. To stochastic nature of response time for service $\mathrm{CS}_j^i$ is modelled by PDF or respective CDF. The PDF and CDF for service $\mathrm{CS}_j^i$ are denoted by $f_j^i(t)$ and $F_j^i(t)$, respectively.

We define the composite service *assurance* $T$ as the probability that a single request will be executed within end–to–end penalty deadline $\delta_p$. Composite service provider is rewarded by $R$ when single request is completed within the deadline, otherwise, composite service provider pays penalty $V$ to the end customer when this deadline is not met. This penalty is also enforced when none of the concrete services is available for task $i$ and therefore the composite service provider cannot generate the response at all.

We assume that the orchestrator knows (due to constant monitoring of the services) which of the service alternatives are available for task $i$, as some of the alternatives may not be available at the decision moment. The availability of services is expressed by the probability $p_j^i$ that the concrete service $\mathrm{CS}_j^i$ is operational and accessible when required for use, [73]. The orchestrator does *not* know at service selection moment for task $i$ which of the concrete services (for tasks $i+1$ and onwards) are available, but it does know the availability probabilities of these. In absence of constant service availability monitoring, the orchestrator could, due to the lack of knowledge, submit the request to the service that is temporarily unavailable. This issue may be overcome by establishing the response–time thresholds that indicate when certain service may be considered unavailable. This approach would further require an implementation of "watchdog timer" policy, i.e. determination of the optimal moments to terminate the current request, and perform the retry, using the same or a functionally equivalent service implementing the same task. This is considered in Chapter 8.

## 7.3 Algorithm for optimising expected revenue

In this section we describe how to optimise expected revenue in the setting described in the previous section, by formulating the dynamic service selection as a DP problem [15]. Although formulae deduced here may appear "similar" to those developed in Chapter 6, the inclusion of the services' availability within these formulae is a challenging task. The given formulae result in different (and more accurate) service composition policies, which, consequently, leads to *significant* revenue improvements, as illustrated in Section 7.6. This also stresses the need to at jointly investigate impact of availability and response–time parameters for the service composition.

Due to the fact that we take into account the actual availability of services, different from "classical" DP approach, we store a permutation (ordering) of concrete services implementing task $i$. This permutation is denoted by $j_1, \ldots, j_{M_i}$. When the orchestrator makes service selection decision, it will use these orderings to choose the 'best' concrete services among the ones that are available. Besides, if the deadline expires during the execution of the composite service request, with some tasks still not executed, the decision rule is to use the cheapest services available for each of the remaining tasks.

The decision policy is determined by the current position within the sequential work-

flow $i$, the available services at this position, and the remaining time $\Delta$ till the overall deadline $\delta_p$ will be violated. When the remaining time to deadline has the value of $\delta$ a set of expected rewards $\mathbb{E}[R^i \mid \Delta = \delta]$ is defined by the DP recursion:

$$\begin{aligned}\mathbb{E}[R^i \mid \Delta = \delta] = p^i_{j_1}\Big( - c^i_{j_1} + \mathbb{E}[R^i_{j_1} \mid \Delta = \delta] - \mathbb{E}[V^i_{j_1} \mid \Delta = \delta]\Big) + \\ + (1 - p^i_{j_1})p^i_{j_2}\Big( - c^i_{j_2} + \mathbb{E}[R^i_{j_2} \mid \Delta = \delta] - \mathbb{E}[V^i_{j_2} \mid \Delta = \delta]\Big) + \\ + \cdots + (1 - p^i_{j_1}) \cdots (1 - p^i_{j_{M_i-1}})p^i_{j_{M_i}} \cdot \\ \cdot \Big( - c^i_{M_i} + \mathbb{E}[R^i_{M_i} \mid \Delta = \delta] - \mathbb{E}[V^i_{M_i} \mid \Delta = \delta]\Big) + \\ + (1 - p^i_{j_1}) \cdots (1 - p^i_{j_{M_i}})V,\end{aligned} \tag{7.1}$$

where $i = 1, 2, \ldots, N$.

For given task $i$, and without loss of generality, there is an order $j_1, \ldots, j_{M_i}$ of concrete services such that choosing $j_1$ would lead to the highest expected CSP revenue, choosing $j_2$ to the second highest expected CSP revenue, etc. This is specified as

$$\begin{aligned}-c^i_{j_1} + \mathbb{E}[R^i_{j_1} \mid \Delta = \delta] - \mathbb{E}[V^i_{j_1} \mid \Delta = \delta] \geq \\ \geq -c^i_{j_2} + \mathbb{E}[R^i_{j_2} \mid \Delta = \delta] - \mathbb{E}[V^i_{j_2} \mid \Delta = \delta] \geq \\ \geq \cdots \geq -c^i_{j_{M_i}} + \mathbb{E}[R^i_{j_{M_i}} \mid \Delta = \delta] - \mathbb{E}[V^i_{j_{M_i}} \mid \Delta = \delta].\end{aligned} \tag{7.2}$$

The selection strategy by the orchestrator for given task $i$ is based on choosing service $\mathrm{CS}^i_{j_1}$ if available; if not, choosing $\mathrm{CS}^i_{j_2}$ if available, and so on. The term $\mathbb{E}[R^i_j \mid \Delta = \delta]$ represents the expected reward, when concrete service $\mathrm{CS}^i_j$ is executed for the given remaining time value $\delta$. The term $\mathbb{E}[V^i_j \mid \Delta = \delta]$ represents the expected penalty for exceeding the overall deadline while executing service $\mathrm{CS}^i_j$. The expected reward and penalty functions take into account the impact of *future decisions*. This leads to the optimal solution of the problem [15].

However, the evaluation of $\mathbb{E}[R^i_j \mid \Delta = \delta]$ and $\mathbb{E}[V^i_j \mid \Delta = \delta]$ requires the usage of PDF (CDF), which usually results in rather complicated integral equations 6, Section 6.4. To resolve this issue, the discretization of distributions is required. The response time of interest (end–to–end deadline) is therefore split into segments of the same size $h$. For the total number of segments $m^*$, this leads to discretized versions of the PDF ($p^i_{j,k}$) and CDF ($P^i_{j,k}$):

$$p^i_{j,k} = P\big(D^i_j \leq h[k + 0.5]\big) - P\big(D^i_j \leq h[k - 0.5]\big),\ P^i_{j,k} = \sum_{l=0}^{k} p^i_{j,l}, \tag{7.3}$$

where $i = 1, \ldots, N,\ j = 1, \ldots, M_i,\ k = 0, \ldots, m^*$.

Let terms $R^i_{m^*}$, $R^i_{j,m^*}$, and $V^i_{j,m^*}$ represent discretized versions of $\mathbb{E}[R^i \mid \Delta = \delta]$, $\mathbb{E}[R^i_j \mid \Delta = \delta]$, and $\mathbb{E}[V^i_j \mid \Delta = \delta]$, respectively. The backward recursion formulae

are then as follows:

$$\begin{aligned} R^i_{m^*} = p^i_{j_1}\Big( - c^i_{j_1} + R^i_{j_1,m^*} - V^i_{j_1,m^*}\Big) + (1-p^i_{j_1})p^i_{j_2}\Big( - c^i_{j_2} + R^i_{j_2,m^*} - V^i_{j_2,m^*}\Big) + \\ + \cdots + (1-p^i_{j_1})(1-p^i_{j_2})\cdots(1-p^i_{j_{M_i-1}})p^i_{j_{M_i}} \cdot \\ \cdot \Big( - c^i_{j_{M_i}} + R^i_{j_{M_i},m^*} - V^i_{j_{M_i},m^*}\Big) + (1-p^i_{j_1})(1-p^i_{j_2})\cdots(1-p^i_{j_{M_i}})V, \end{aligned} \tag{7.4}$$

where $i = 1, \ldots, N$, and, without loss of generality, $j_1, \ldots, j_{M_i}$ is a permutation of $(1, 2, \ldots, M_i)$, such that

$$\begin{aligned} -c^i_{j_1} + R^i_{j_1,m^*} - V^i_{j_1,m^*} \geq -c^i_{j_2} + R^i_{j_2,m^*} - V^i_{j_2,m^*} \geq \cdots \geq \\ \geq -c^{j_n}_i + R^i_{j_{M_i},m^*} - V^i_{j_{M_i},m^*}, \end{aligned} \tag{7.5}$$

$$R^i_{j,m^*} = \begin{cases} P^N_{j,m^*} R, & i = N, \\ \sum\limits_{k=0}^{m^*} p^i_{j,k} R^{i+1}_{m^*-k}, & i = 1, \ldots, N-1, \end{cases} \tag{7.6}$$

and

$$V^i_{j,m^*} = \begin{cases} \left(1 - P^N_{j,m^*}\right) V, & i = N, \\ \left(1 - P^i_{j,m^*}\right) R^{i+1}_{m^*=0}, & i = 1, \ldots, N-1, \end{cases} \tag{7.7}$$

where $j = 1, \ldots, M_i,\ k = 0, \ldots, m^*$.

While applying formulae (7.4)–(7.7), the corresponding decisions (actions) can be obtained by storing $(j_1, \ldots, j_{M_i})$ for every task $i = 1, \ldots, N$ and every possible time interval $m^*$, where $(j_1, \ldots, j_{M_i})$ is a permutation of $(1, 2, \ldots, M_i)$.

We define a (deterministic) decision strategy $S$ for every $i = 1, \ldots, N$ and every time interval $m^*$, as a permutation $S(i, m^*) := (j_1, \ldots, j_{M_i})$ of $(1, 2, \ldots, M_i)$. The choice to be made for task $i$ and given time interval $m^*$ is then preferably to choose concrete service $\mathrm{CS}^i_{j_1}$. If this service is not available, then choose $\mathrm{CS}^i_{j_2}$ and so on. We denote by $S^*$ the decision strategy that gives the optimal expected revenue, i.e. this is decision strategy such that (7.5) is satisfied. In the next section we address the relation between the optimal revenue decision strategy $S^*$ and end–to–end quality assurance $T$.

## 7.4 Relation between optimal revenue strategy and quality assurance

In general, the optimal decision strategy $S^*$ may not necessarily guarantee the highest quality assurance $T_{\max}$. In this section we will first develop the algorithm to calculate the end–to–end assurance and show strategies to increase end–to–end assurance, while sacrificing expected revenue.

### 7.4.1 Calculation of end–to–end assurance

In order to calculate the end–to–end assurance we use the (discretized) setting as described in Section 7.3, equations (7.3)–(7.7). The deadline $\delta_p$ can be expressed as $m_p = \lceil \delta_p / h \rceil$, i.e. it is expressed as the number of discretization segments. For a given decision strategy $S$, we define the end–to–end assurance $T = T(S)$ of the (discretized) decision strategy $S$ to be the probability that the deadline $m_p$ will be kept. The end–to–end assurance can be calculated using the following recursive relations:

$$T^i_{j,m^*} = \begin{cases} P^N_{j,m^*}, & i = N, \\ \sum\limits_{k=0}^{m^*} p^i_{j,k} T^{i+1}_{m^*-k}, & i = 1, \ldots, N-1, \end{cases} \tag{7.8}$$

and

$$\begin{aligned} T^i_{m^*} = {} & p^i_{j_1} T^i_{j_1,m^*} + (1 - p^i_{j_1}) p^i_{j_2} T^i_{j_2,m^*} + \cdots + \\ & + (1 - p^i_{j_1})(1 - p^i_{j_2}) \cdots (1 - p^i_{j_{M_i-1}}) p^i_{j_{M_i}} T^i_{j_{M_i},m^*}, \end{aligned} \tag{7.9}$$

where $S(i, m^*) = (j_1, \ldots, j_{M_i})$. The end–to–end assurance $T(S)$ equals $T^1_{m_p}$. Clearly $0 \leq T(S) \leq 1$ for any given strategy $S$.

In general, not any value of end–to–end assurance may be reached. The highest possible end–to–end assurance can be calculated as described with the ordering $(j_1, \ldots, j_{M_i})$ of concrete services at each task $i$ and time interval $m^*$ on the assurance factors $T^j_{m^*,i}$. The highest assurance is achieved for permutation $(j_1, \ldots, j_{M_i})$ such that

$$T^i_{j_1,m^*} \geq T^i_{j_2,m^*} \geq \cdots \geq T^i_{j_{M_i},m^*}.$$

**Remark:** The optimal revenue strategy $S^*$ defined in section 7.3 need not be unique, because there might be different permutations $(j_1, \ldots, j_{M_i})$ of $(1, \ldots, M_i)$ such that (7.5) is satisfied, namely when there are equalities. However, each revenue policy $S$ has accompanied assurance $T^i_{j,m^*}$, calculated using (7.8)–(7.9). In case when there are two or more revenue policies that yield the same revenue, these policies can be ordered according to their respective assurance. In case when two or more policies $S$ yield the same revenue, and have the same assurance, we further order them according to the lexicographic ordering of permutations. This is how the uniqueness of the selection strategy is achieved.

### 7.4.2 Optimal revenue strategy with target assurance

We denote by $S^*_V$ the unique optimal revenue decision strategy for given penalty $V$, by $\mathbb{E}[R_V(S)]$ we denote the expected revenue for strategy $S$ and given penalty $V$. It holds that $\mathbb{E}[R_V(S^*_V)] \geq \mathbb{E}[R_V(S)]$ for all selection strategies $S$. However, the

optimal revenue strategy $S_V^*$ need not have the highest end–to–end assurance, for instance due to expensive but high assurance individual services in the chain. As explained in the remark, the assurance $T(S_V^*)$ is maximal among all strategies $S$ *under condition* that $\mathbb{E}[R_V(S)] = \mathbb{E}[R_V(S_V^*)]$. The composite service provider may need to guarantee a certain minimal end–to–end assurance $T$ for given penalty $V$. This assurance might not be attained by the optimal revenue selection strategy $S_V^*$. This establishes the need to determine a different strategy $S \neq S_V^*$ that has maximal expected revenue *under the condition* that it has end–to–end assurance greater than or equal to $T$. The strategy $S$ could be determined by usage of the following theorem.

**Theorem 7.4.1.** *Let penalties $W$ and $V$ satisfy $0 \leq V < W$. For given end–to–end deadline, let $S_W$ be the optimal revenue strategy for penalty $W$ with assurance $T = T(S_W)$. Then $\mathbb{E}[R_V(S_W)] \geq \mathbb{E}[R_V(S)]$ for all selection strategies $S$ with $T(S) \geq T$.*

*Proof.* Let $C_{11}$ be the expected cost paid when strategy $S_W$ is used under the condition that the end–to–end deadline is met, and $C_{12}$ be the expected cost paid when strategy $S_W$ is used, under the condition that the end–to–end deadline is not met. Given the assurance $T = T(S_W)$, the expected revenue is

$$\mathbb{E}[R_W(S_W)] = T \cdot (-C_{11} + R) + (1 - T) \cdot (-C_{12} - W).$$

Let $S$ be any selection strategy under condition $T(S) \geq T$. Let $C_{21}$ be the expected cost when strategy $S$ is used, under the condition that the end–to–end deadline is met, and let $C_{22}$ be the expected cost paid when strategy $S$ is used, under the condition that the end–to–end deadline is not met. Then

$$\mathbb{E}[R_W(S)] = T(S) \cdot (-C_{21} + R) + (1 - T(S)) \cdot (-C_{22} - W).$$

Since $S_W$ is optimal revenue strategy for given penalty $W$, it holds that $\mathbb{E}[R_W(S_W)] \geq \mathbb{E}[R_W(S)]$, i.e.

$$T \cdot (-C_{11} + R) + (1 - T) \cdot (-C_{12} - W) \geq T(S) \cdot (-C_{21} + R) + (1 - T(S)) \cdot (-C_{22} - W).$$

Now, adding $(1-T)(W-V)$ on both sides, and given the fact that $(1-T)(W-V) \geq (1 - T(S))(W - V)$, we obtain:

$$\begin{aligned}
&\mathbb{E}[R_V(S_W)] = T \cdot (-C_{11} + R) + (1 - T) \cdot (-C_{12} - V) \\
&= T \cdot (-C_{11} + R) + (1 - T) \cdot (-C_{12} - W) + (1 - T)(W - V) \\
&\geq T(S) \cdot (-C_{21} + R) + (1 - T(S)) \cdot (-C_{22} - W) + (1 - T)(W - V) \\
&\geq T(S) \cdot (-C_{21} + R) + (1 - T(S)) \cdot (-C_{22} - W) + (1 - T(S))(W - V) \\
&= T(S) \cdot (-C_{21} + R) + (1 - T(S)) \cdot (-C_{22} - V) = \mathbb{E}[R_V(S)],
\end{aligned}$$

which completes the proof. □

The following conclusions could be made from the given theorem:

- The end–to–end assurance increases with the increase of the penalty value.

- The optimal expected revenue decreases with the increase of the penalty value.
- When optimal revenue strategy $S_V^*$ for given penalty $V$ does not meet target assurance, a new strategy $S_W$ needs to be calculated. This is done by finding the smallest penalty value $W(W > V)$, for which the optimal revenue strategy $S_W$ meets the target assurance $T$. The revenue strategy $S_W$ is then optimal under condition that target assurance is met.

## 7.5 Calculation of end–to–end response time distribution

The end–to–end assurance only describes the probability to complete the request before the deadline. The assurance corresponding to a selection strategy $S$ could be readily calculated from the end–to–end response time distribution. However, the determination of the end–to–end response time distribution allows to identify, e.g. strategies that have response time close to the deadline with relatively high probability. Therefore, we describe here how the end–to–end response time distribution $r^*$ of a strategy $S$ can be computed. In section 7.6 we will demonstrate the usage of such distributions.

Suppose we have a selection strategy $S$, that will give, for each task $i$ and time $t$, a permutation $(j_1, \ldots, j_{M_i})$ of $(1, \ldots, M_i)$, i.e. $S(i,t) = (j_1, \cdots, j_{M_i})$, such that, if available, concrete service $j_1$ will be selected, else concrete service $j_2$ if available, etc. Let $r^i_{j,s}$ be the response time distribution when $\mathrm{CS}^i_j$ is available and selected, and the remaining time to deadline equals to $s$. Further, let $r^i_s$ be the response time distribution for task $i$ and remaining time to deadline $s$, i.e. no concrete service for task $i$ is selected yet. When no concrete services are available for given task, the composite request will not be completed, and, for such a scenario, we assume the response time to be infinite. Then, with remaining time to deadline $s$, we have

$$\begin{aligned} r^i_s = {} & p^i_{j_1} r^i_{j_1,s} + (1-p^i_{j_1}) p^i_{j_2} r^i_{j_2,s} + \cdots + \\ & + (1-p^i_{j_1})(1-p^i_{j_2}) \cdots (1-p^i_{j_{M_i-1}}) p^i_{j_{M_i}} r^i_{j_{M_i},s} + \\ & + (1-p^i_{j_1}) \cdots (1-p^i_{j_{M_i}}) \end{aligned} \tag{7.10}$$

where $i = 1, 2, \ldots N$.

The response–time distribution for $\mathrm{CS}^i_j$, with remaining time to deadline $s$ is

$$r^i_{j,s}(t) = \begin{cases} f^N_j(t), \quad i = N, \quad t \le s \\ \int_0^\infty f^i_j(y)(R_+(y) r^{i+1}_{s-y})(t)\, dy, \\ \qquad\qquad \text{for } i = 1, \ldots, N-1, \end{cases} \tag{7.11}$$

where $j = 1, 2, \ldots M_i$ and $(R_+(y)g)(t)$, is given by

$$(R_+(y)g)(t) = \begin{cases} g(t-y), & t \geq y, \\ 0, & t < y. \end{cases} \tag{7.12}$$

## 7.6 Numerical experiments

In this section we will give some numerical experiments of the theory described in previous sections, and state some observations coming from these experiments. Unless otherwise specified, we illustrate the results for the case of the sequential workflow that consists of $N = 4$ tasks (abstract services), and each task is implemented by four different concrete services (alternatives).

### 7.6.1 A static algorithm for comparison

We need first to determine a (static) reference to compare our results to. One approach may be [116], to determine an optimal path $W = (W_1, \cdots, W_N)$ in advance, such that for task $i$, a concrete service $W_i$ is selected. This approach is not applicable for the problem we consider here because any of the individual services in the optimal path may be unavailable, which implies that the probability that the composite service never generates a reply can become quite large. As an illustration, let us analyse a sequential workflow with $N$ tasks, where each concrete service has availability that is same and equals to a value $p < 1$. In case when an optimal path is determined, i.e. a single concrete alternative has been chosen for each task, and that choice is not changed, the probability that composite service request would not be served is $1 - p^N$, which goes to 1 quickly as $N$ becomes large.

As this would mean an unfair assessment of the static approach, we adjust it as follows: For each task $i$, $i = 1, \ldots N$, a specific permutation $w_i^1, \ldots, w_i^{M_i}$ of concrete services $(1, \ldots, M_i)$ is chosen, such that

(1) of all possible compositions, $(w_1^1, \cdots, w_N^1)$ gives the optimal expected revenue, under the condition that these concrete services do not fail,

(2) for $i \in \{1, \cdots, N\}$, and $k \in \{1, \cdots, M_i\}$, denote $\mathbb{E}[r_i^k] :=$ the expected revenue for the composition $(w_1^1, \cdots, w_{i-1}^1, w_i^k, w_{i+1}^1, \cdots, w_N^1)$. Then $\mathbb{E}[r_i^1] \geq \mathbb{E}[r_i^2] \geq \cdots \geq \mathbb{E}[r_i^{M_i}]$.

This means that, in our reference case, the optimal (static) composition $(w_1^1, \cdots, w_N^1)$ is executed. However, whenever a concrete service $w_i^1$ at task $i$ is unavailable, the concrete service $w_i^2$ is selected instead when available, otherwise $w_i^3$ if available, etc.

### 7.6.2 Revenue and assurance improvements with identical availability of services

In this subsection we illustrate the revenue and assurance improvements when our solution is applied, compared to the reference. For each abstract service we assume

Table 7.1: Parameters of concrete services implementing a single task $i$.

| | $c$ | $\mu$ | $\sigma$ |
|---|---|---|---|
| Alternative 1 | 1 | 5 | 2 |
| Alternative 2 | 2 | 3 | 2 |
| Alternative 3 | 5 | 2.5 | 2 |
| Alternative 4 | 10 | 1.25 | 3 |

we have four concrete services with parameters as in Table 7.1, i.e. with cost $c$, and parameters $\mu$ and $\sigma$ of the log–normal distribution of the response time. The end–to–end penalty deadline is $\delta_p = 18$, reward $R = 20$ and penalty value is $V = 50$. We also assume that concrete services for all tasks have the same availability probabilities, and these probabilities vary from 0.6 to 1. The results for expected revenue and end–to–end assurance are shown in Figure 7.2. The dynamic run–time selection strategy significantly outperforms the static reference strategy, with respect to both revenue and end–to–end assurance. The absolute difference among expected revenue for the dynamic and static strategy remains more or less constant, independent of availability. However, the end–to–end assurance comes farther apart as the availability probabilities approach one.

When availability probabilities differ from one, our dynamic strategy takes future availability uncertainties into account, while the static strategy does not. As we see in Figure 7.2 that the difference in expected revenue stays more or less constant, while the availability probabilities decrease, we conjecture that in this case the main contribution to the advantage of the dynamic strategy versus the static strategy lies in the fact that the dynamic strategy takes realised response times into account, and not that it takes future availability probabilities into account. When the availability of concrete services equals one, the difference between expected revenue for the dynamic and static strategies results from the fact that the dynamic strategy takes into account realised response times of the subservices when selecting the concrete services for the next task. The dynamic strategy can therefore take advantage of a low realised response time, even if the probability for this realisation is small.

### 7.6.3 Expected revenue and quality assurance

In this section we illustrate an application of Theorem 7.4.1. Again, the setting consists of a sequential workflow with four tasks, of which each has four alternatives with parameters specified in Table 7.1. Furthermore, we assume that reward $R = 20$,

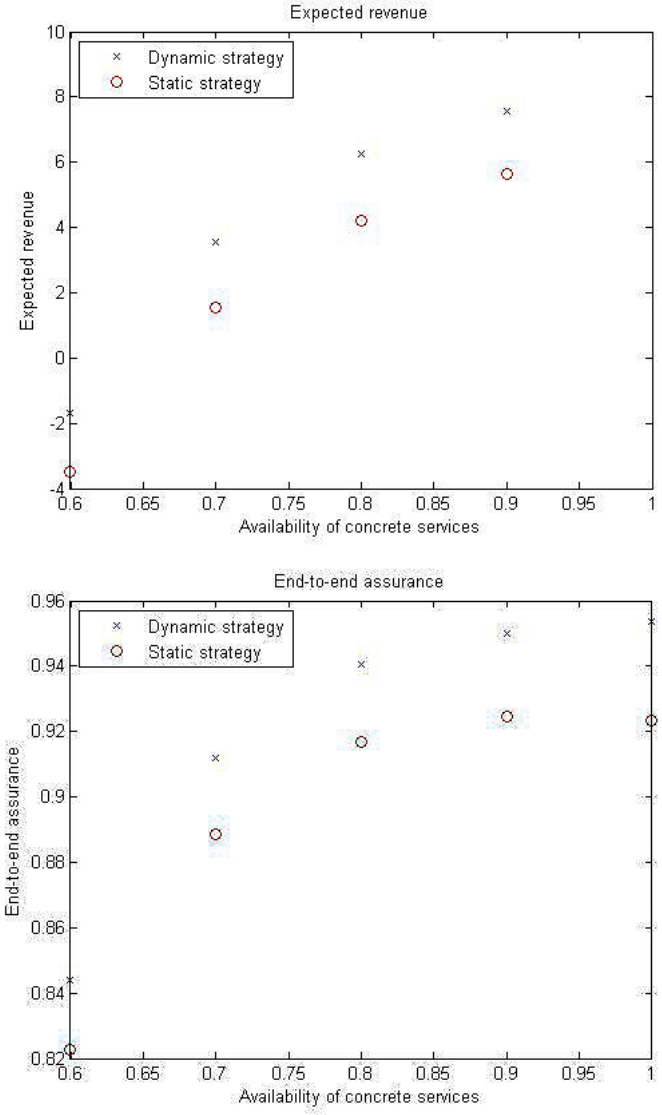


Figure 7.2: Comparison between the static and dynamic strategies for service parameters given in Table 7.1.

the penalty $V = 50$, the penalty deadline $\delta_p = 18$, and the concrete services all have an availability of 1. The dynamic strategy in this case realises an end–to–end assurance of 0.9537, see also Figure 7.2. Let us now suppose that the minimal target end–to–end assurance is $T_t = 0.97$, and let us determine the strategy that yields

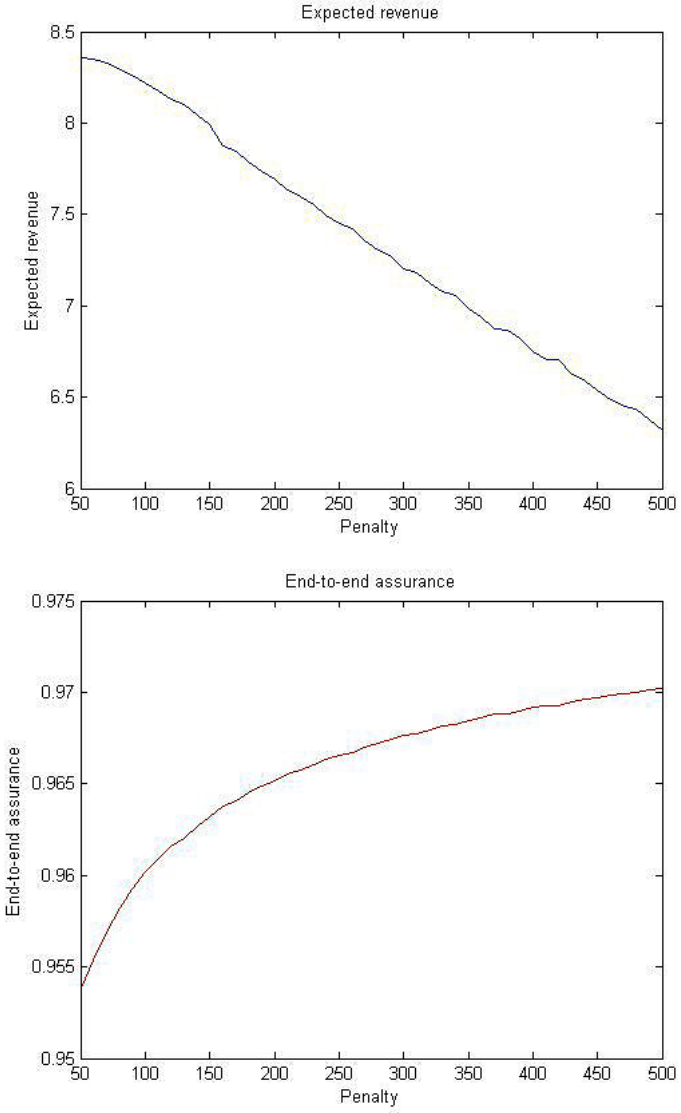


Figure 7.3: Expected revenues and end–to–end assurance for different penalty values.

optimal expected revenue under this additional requirement. We first check whether target assurance is achievable at all. To do so, we determine maximum assurance $T_{\max}$, as described at subsection 7.4.1. We obtain the value of $T_{\max} = 0.98$, with an (negative!) expected revenue of $-5.862$. The target assurance of 0.97 is therefore possible.

Applying the Theorem 7.4.1 we develop the following steps to determine the strategy:

1. Increase the initial penalty $V = 50$ by increments $\Delta V$, say $\Delta V = 10$.
2. In $n$–th iteration, determine the strategy $S_n$ that gives the optimal expected revenue for penalty $V + n\Delta V$. Calculate the resulting end–to–end assurance under strategy $S_n$, i.e. $T(S_n)$.
3. For the first iteration $n$ which results in assurance $T(S_n) \geq T_t$ compute the expected revenue for strategy $S_n$ using the initial penalty $V = 50$.

The approach is illustrated in Figure 7.3 for the initial penalty value $V = 50$, $\Delta V = 10$ and $T_t = 0.97$. The target assurance $T_t$ is reached for iteration $n = 44$, i.e. the penalty value of 490. This results in strategy $S_n$ which yields expected revenue of 6.37 for $V = 50$. By Theorem 7.4.1 this is the highest expected revenue that can be achieved under the condition that the end–to–end assurance is greater than or equal to 0.97.

### 7.6.4 Impact of availability

In this set of experiments we investigated the alternating individual service availability, and the impact it has to expected revenue and end–to–end assurance. The parameters of the concrete services are specified in Table 7.1. However, not all services have the same availability; we set the availability of all concrete services implementing task 1 to 0.55, while concrete services implementing tasks 2, 3 and 4 have availability of 0.65, 0.75 and 0.85, respectively.

In the Table 7.2 all 24 possible compositions are shown. For example, configuration with label a uses alternative 4 at position (i.e. task) 1, alternative 3 at position 2, and so on. For respective configurations the resulting revenues and end–to–end assurance are illustrated in Figures 7.4 and 7.5, respectively. Both Figure 7.4 and Figure 7.5 show "gaps" of dramatic revenue increase (respectively assurance increase), noted by A, B and C in the figure. Careful examination of permutations from Table 7.2 indicates that gap A in Figure 7.4 results from the fact that the alternatives for the last two tasks change from 2 and 1 (with availabilities 0.65 and 0.55 respectively) to alternatives 3(4) and 1 for the same tasks. Similarly, gap B results from the fact that the alternative for the last task is not 1 any more, but 2.

This leads to the following observations:

- The highest optimal revenue is achieved when the services with highest availability probabilities are placed at or near the end, and services with lowest availability probabilities are placed at or near the beginning of the service chain. One logical explanation for this is as follows: when optimal concrete services are unavailable at the beginning of the service chain, there is still enough "room" (in the time budget) to compensate this with a different strategy. When they are unavailable near the end, this "room" to manoeuvre is much smaller.

Table 7.2: Indices of alternative assurance configurations

| label → | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ | $j$ | $k$ | $l$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| position 1 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 |
| position 2 | 3 | 3 | 2 | 2 | 1 | 1 | 4 | 4 | 2 | 2 | 1 | 1 |
| position 3 | 2 | 1 | 3 | 1 | 2 | 3 | 2 | 1 | 4 | 1 | 2 | 4 |
| position 4 | 1 | 2 | 1 | 3 | 3 | 2 | 1 | 2 | 1 | 4 | 4 | 2 |
| label → | $m$ | $n$ | $o$ | $p$ | $q$ | $r$ | $s$ | $t$ | $u$ | $v$ | $w$ | $x$ |
| position 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| position 2 | 3 | 3 | 4 | 4 | 1 | 1 | 3 | 3 | 2 | 2 | 4 | 4 |
| position 3 | 4 | 1 | 3 | 1 | 4 | 3 | 2 | 4 | 3 | 4 | 2 | 3 |
| position 4 | 1 | 4 | 1 | 3 | 3 | 4 | 4 | 2 | 4 | 3 | 3 | 2 |

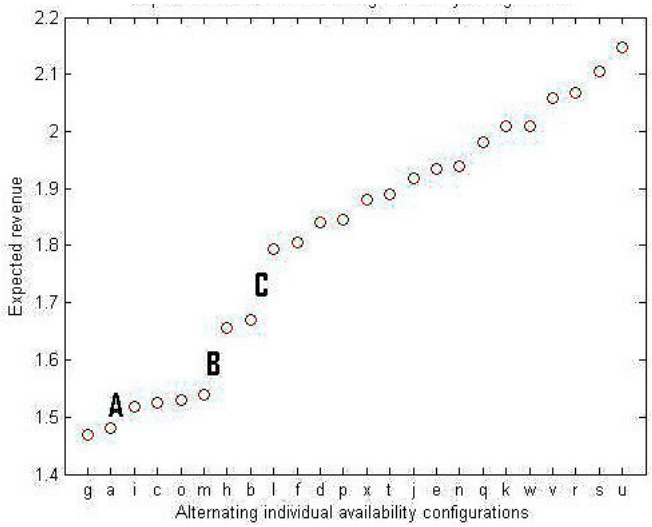


Figure 7.4: Expected revenues for different availabilities, sorted by permutation indices.

- The above observation indicates that when considering which service alternatives could play a role in constructing a composite service, extra care must be taken to ensure that the service alternatives near the end of the chain have sufficient availability probabilities.

### 7.6.5 Response time distribution

We showed in Section 7.5 how the response time PDF can be determined, and we illustrate the results in Figure 7.6. The same setting and parameters as in subsection 7.6.3 are taken. From the respective PDFs we notice that dynamic response

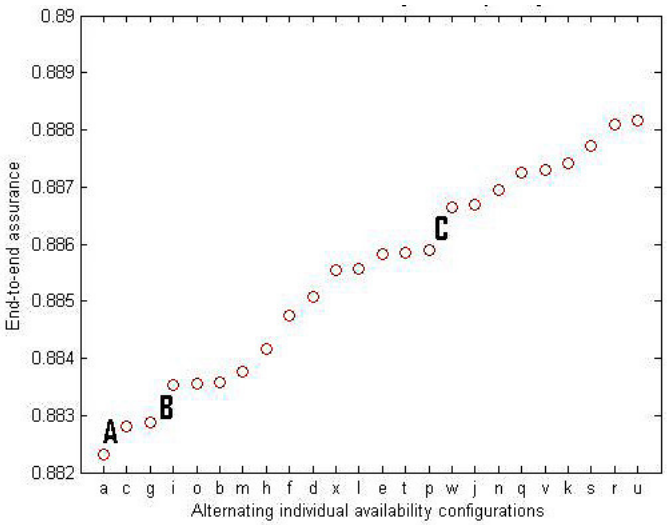


Figure 7.5: End–to–end assurance for different availabilities, sorted by permutation indices.

time PDFs shift more towards the deadline compared to the reference strategy. The dynamic strategy focuses on optimal expected revenue, and when more time is available, the strategy uses this time and select cheaper but slower services. It is also clear from Figure 7.6 that dynamic strategy results in higher assurance than the reference strategy.

## 7.7 Conclusions

In this chapter we presented a dynamic programming based solution for run–time web service composition with two potentially conflicting goals: revenue maximization and quality assurance (specified as the probability that a request made would meet a given deadline). An important feature of the considered model compared to the one used in Chapter 6 is that we take into account partial service availability, next to the stochastic model of services' response time and service costs. We apply the dynamic programming approach to derive the optimal service selection policy for the problem at hand. The highest optimal revenue is achieved when services with high availability probabilities are used at the end of the workflow. It may happen that optimal service selections for the tasks at the beginning of the workflow are low–cost, "slow" services, which more often than other choices may be unavailable. When these services at the beginning of the workflow are unavailable, there is still enough "room" (in the time budget and costs) to compensate for this when executing the remaining tasks. However, the choice narrows down to just a few (or none) options

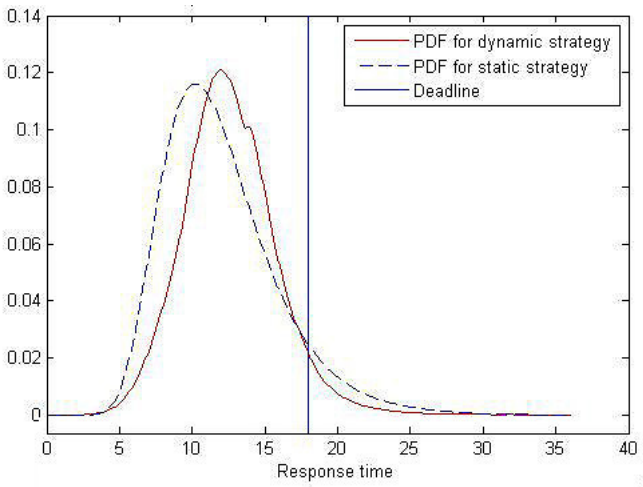


Figure 7.6: Comparison of response–time PDFs for dynamic and reference (static) strategy when end–to–end deadline is 18.

in case when optimal service selections are not available at the end of the workflow. We have shown how to determine the policy that would result in maximum revenue for a given target quality assurance.

Using our model, we also determined the end–to–end response–time PDF under the optimal selection policy. We have shown that the probability mass in the PDF and the mean shift to the higher values when deadline increases. The dynamic strategy has a goal of maximizing the expected revenue, and not to minimize the end–to–end response time. For example, it may happen that more time budget remains to meet the deadline due to the relatively fast execution of tasks at the beginning of the workflow. In such a case, the strategy exploits this and selects cheaper and slower services for the last tasks. This results in shift of the probability mass and the mean of end–to–end PDF. Naturally, in case the optimization goal emphasises quality assurance, the response–time distributions have their weight closer to zero, and further away from the deadline.

We also compared end–to–end response–time distributions resulting from the developed service selection policy and static benchmark service selection. As the deadline increases, the mean in case of static service selection strategy remains almost constant, while in the case of dynamic run–time service selection mean increases, thus it is closer to the deadline. The model discussed in this chapter assumes that the orchestrator has ideal knowledge whether services implementing the next task to be executed are available at the moment (optimal) selection is made. In Chapter 8 we will use actual requests to derive conclusions whether particular service is available or not.

CHAPTER 8

# Optimal Service Selection with Conditional Request Retries

In the Chapters 4–7 we considered the dynamic, run–time service selection as end–to–end QoS–control mechanism for composite web services. This chapter addresses an additional QoS–control mechanism based on *conditional retries*. The conditional retry may be issued when the execution of a concrete service within a composition lasts too long. The retry can then either invoke a functionally equivalent web service that implements the same task, or possibly, the same concrete service. The goal of such adaptation is to maximize the revenues of the composite service provider (CSP), while meeting the end–to–end deadline specified in the SLA that the CSP has with its clients, even when some of the services used for the composition becomes *transiently* less responsive (its performability worsens).

Note that in Chapter 7 the assumption used is that the orchestrator "knows" (e.g. due to constant monitoring of the services) which of the service alternatives are available for given task at the decision moment. However, this requires constant monitoring of the services by probes. Probes are service requests that are not part of the composite service execution. Taking into account that each single service invocation involves costs, this approach is not desirable. Moreover, in case when services are transiently unavailable, a service may cease to function while serving actual requests.

In this chapter we derive the *optimal* runtime service selection policy for the setting described above. In contrast to the selection policies considered in Chapter 6 and Chapter 7, the policy derived in this chapter also specifies the (optimal) moments when retry should be attempted, and which service should be used for a retry. The derived policy is optimal with respect to the expected revenue of the CSP.

The main contributions of this chapter are as follows:

- Development and design of a performance model for the system with the conditional retry mechanism.
- Using a dynamic programming approach the optimal decision policy with respect to profit maximization of the composite service provider is derived. A step–by–step procedure that leads to the optimal decision policy is specified.

- The revenue improvements of the proposed solution are quantified and compared to orchestrated composition strategies that do not apply conditional retries. These alternative strategies range from (optimal) static service composition to the dynamic service composition strategy without the conditional retry mechanism developed in Chapter 6.
- As the simulations for the given scenarios indicate that only a few retry options may be enough for significant revenue improvements we also present the analysis of the scheme with a single retry possibility for the whole workflow. This special case leads to insightful special formulae for the optimal retry moment.

The chapter is organized as follows: in the next section we provide an overview of the related work. In Section 8.2 we describe the system model and the assumptions taken. Besides, we specify a step–by–step procedure that leads to the decision policy. Based on the given solution, we describe the simulation results for a number of scenarios in Section 8.3. The analysis of the single retry case is presented in Section 8.4 and we conclude the chapter in Section 8.5.

This chapter is based on papers [119] and [120].

## 8.1 Related work

Retry mechanisms as a solution for QoS control of temporarily unavailable services, have been identified and classified, among others, in [7, 13]. The performance of the retry mechanisms has been analysed in detail by van Moorsel, Wolter, et al. [97,98,103]. Their work has focused on optimal retry mechanisms for a single service in order to *minimize the completion time*. The number of retries could either be finite or infinite, and the authors determine optimal retry moments, i.e. minimization of completion time. Okamura et al. [74] analyse the optimal restart policies when deadline is given and develop on–line adaptive algorithms for estimating the optimal restart time interval via reinforcement learning. None of these solutions analyse the problem using the penalty or reward of any kind. The cost of the retries are defined as additional time to re–issue the service request. Besides, as in [97, 98, 103], the retry mechanism is analysed from the single service point of view.

Yousefi et al. [110] describe a strategy for QoS aware service selection which takes advantage of the existing variability in QoS data to provide higher quality services with less cost compared to the conventional QoS aware service selection methods. In their method, *each request* is replicated over multiple independent services implementing the same task to achieve the required QoS, i.e. limit the response time by certain pre–assigned value. Although the execution costs are taken into account for the analysis, no penalties are considered and the replication mechanism is analysed for a single task only. In this chapter we optimize request replication from the point of increasing the profit of composite service provider, with the aim to issue request replication only when it really may be useful. Besides, our analysis is performed for service compositions described by a workflow that may have more than one task.

## 8.2 System model and dynamic programming approach

In this section we describe the system model of a composite web service represented by a sequential workflow with conditional retries and dynamic programming approach to determine the optimal retry moments for such a system. The dynamic programming [15] optimizes composite service provider's revenue by taking into account the effect of *future* (optimal) decisions. Since the decisions within DP take in account effects of subsequent decisions, a "backward recursion" method is needed for finding the optimal solution. The application of DP will result in a decision policy. The decision policy indicates which service is selected for given deadline and given task, and indicates, when applicable, what is the optimal moment for a retry. It may happen, that for given deadline, and selected service, retry is not possible at all, as specified by the policy.

An optimal decision policy with respect to the expected revenue for the CSP is determined using dynamic programming. According to the *principle of optimality*, [15], an optimal policy determined using dynamic programming leads to optimal revenue for all stages of the model.

The given model is applicable to any workflow that could be aggregated and mapped into a sequential one. Some basic rules for aggregation of a non–sequential into the sequential workflow have been illustrated in, e.g. [115, 116]. The aggregation naturally leads to coarser control, since decisions could not be taken for a single service within the aggregated workflow, but rather for the aggregated workflow patterns themselves.

### 8.2.1 Notation and assumptions

The orchestrator executes tasks one–by–one as indicated by the sequential workflow. There are in total $N$ tasks in the workflow and the task position within workflow (*stage*) is indexed by $i$, $i = 1, 2, \ldots, N$. Each task $i$ maps onto $M_i \geq 1$ concrete services, where service $j$ implementing task $i$ is denoted by $\mathrm{CS}_j^i$, see also Figure 8.1. The cost that the composite provider pays for the single invocation of service $\mathrm{CS}_j^i$ is denoted by $c_j^i$. The probabilistic nature of response time for service $\mathrm{CS}_j^i$ is modelled by PDF or respective CDF. The PDF and CDF for service $\mathrm{CS}_j^i$ are denoted by $f_j^i(t)$ and $F_j^i(t)$, respectively. The composite service provider guarantees that a single request will be executed within end–to–end penalty deadline $\delta_p$. CSP is rewarded by $R$ when single request is completed within the deadline, otherwise, the composite service provider pays a penalty $V$ to the end customer when this deadline is not met. For a given stage $i$, $\theta_{j\to k}^i$ represents the moment of the substitution of service $\mathrm{CS}_j^i$ by service $\mathrm{CS}_k^i$, where $j, k = 1, 2, \ldots, M_i$. This substitution takes place only when service $\mathrm{CS}_j^i$ does not provide a response before $\theta_{j\to k}^i$. Note that it is possible

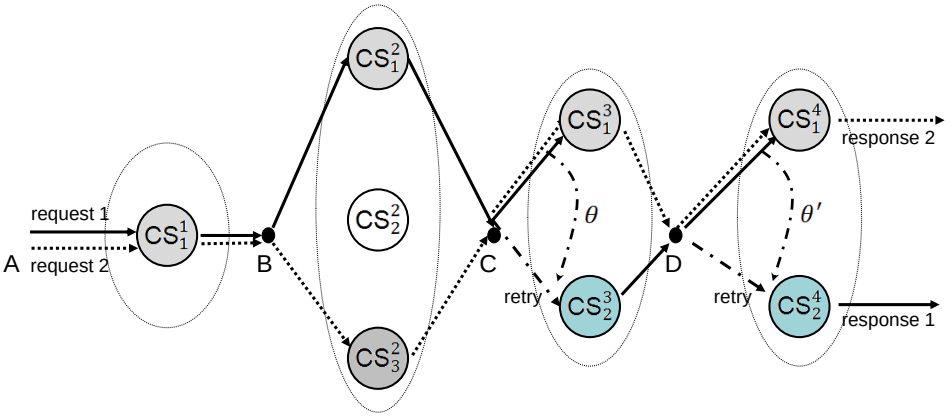


Figure 8.1: Conditional retry adaptation of orchestrated service composition. Run–time service composition and adaptation is based on pre–calculated policy. Decisions are taken at points A–D, and e.g. it is known to the orchestrator at C how long to wait before retry is made.

to issue the retry to the same service. Once the end–to–end deadline expires and there are still tasks to be executed, the retries are not performed any more, and only the cheapest alternative for each of the remaining tasks is invoked.

We denote by $W_j^i(\delta) = W_j^i(\mathcal{D}^i = \delta)$ the revenue when $\mathrm{CS}_j^i$ is selected, no retry is issued, and remaining time–to–deadline $\mathcal{D}^i$ has the value of $\delta$. The expected value of the revenue $W_j^i(\delta)$ is $\mathbb{E}[W_j^i(\delta)]$. Similarly, we denote by $W_{j\to k}^i(\delta, \theta_{j\to k}^i)$ the revenue when, $\mathrm{CS}_j^i$ is initially selected with $\mathcal{D}^i = \delta$, and then substituted by $\mathrm{CS}_k^i$ after $\theta_{j\to k}^i$. The expected revenue of this case with retry is $\mathbb{E}[W_{j\to k}^i(\delta, \theta_{j\to k}^i)]$, where $i = 1, 2, \ldots, N$, and $j, k = 1, 2, \ldots M_i$.

The optimal expected revenue for given stage $i$ and given deadline $\delta$ is denoted as $W^{i*}(\delta)$, and represents maximum among all expected revenues (with and without retry) for that stage, i.e.

$$W^{i*}(\delta) \stackrel{\mathrm{def}}{=} \max_{\substack{j,k=1,2,\ldots,M_i \\ \theta_{j\to k}^i \in [0,\delta]}} \{\mathbb{E}[W_j^i(\delta)], \mathbb{E}[W_{j\to k}^i(\delta, \theta_{j\to k}^i)]\}. \tag{8.1}$$

The model could be specified when the number of retries for a single task is arbitrary. In order to simplify the notation we would discuss the model when a single retry is possible within a single task execution. In practice, multiple retries are not that likely for a single service. Therefore, the maximum total number of retries in our model equals the number of tasks in the sequential workflow.

The notation is summarized in Table 8.1.

Table 8.1: Overview of model parameters

| Parameter | Definition |
|---|---|
| $\mathrm{CS}_j^i$ | Concrete service $j$ implementing task $i$ |
| $f_j^i$ | Response–time distribution of $\mathrm{CS}_j^i$ |
| $F_j^i$ | Cumulative distribution of response time for $\mathrm{CS}_j^i$ |
| $\mathcal{D}^i$ | Time–to–deadline for task $i$ |
| $\delta_p$ | End–to–end deadline |
| $\theta_{j\to k}^i$ | Time instant when $\mathrm{CS}_j^i$ is substituted by $\mathrm{CS}_k^i$ |
| $c_j^i$ | Cost of invocation of $\mathrm{CS}_j^i$ |
| $R$ | Reward per request completed within $\delta_p$ |
| $V$ | Penalty per request not completed within $\delta_p$ |
| $W_j^i(\delta)$ | Revenue without request replication when $\mathrm{CS}_j^i$ is selected; time–to–deadline is $\delta$ |
| $W_{j\to k}^i(\delta,\theta_{j\to k}^i)$ | Revenue with request replication when $\mathrm{CS}_j^i$ is substituted by $\mathrm{CS}_k^i$ at $\theta_{j\to k}^i$ |
| $W^{i*}(\delta)$ | Optimal revenue for task $i$ and given deadline $\delta$ |
| $\pi^i(\delta)$ | Policy for task $i$ and given deadline $\delta$ |

### 8.2.2 Dynamic programming

We will describe now the dynamic programming formulae taking into account the well–known backward recursion procedure. This means that DP is first defined/solved for the last task $N$, and then iteratively for tasks $N-1, N-2, \ldots, 1$. When solving DP for task $i$ and given $\delta$, it is already known what is the value of $W^{(i+1)*}(\delta)$. For given task $i$, given end–to–end deadline $\delta_p$, the procedure calculates $\mathbb{E}[W_j^i(\delta)]$, and $\mathbb{E}[W_j^i(\delta,\theta_{j\to k}^i)]$ where $j,k = 1,2,\ldots M_i$ for all $\delta \in [0,\delta_p]$ and $0 \le \theta_{j\to k}^i \le \delta$, as retries made after the deadline expires make no sense. The maximum revenue is then determined based on Equation 8.1. We present now the formulae that allows evaluation of the maximum revenue per stage, i.e. $W^{i*}(\delta)$.

**The last stage:** for the last stage, i.e. when $i = N$, we need to determine whether it would be optimal to select a single service $\mathrm{CS}_j^N$, and wait till it generates response, or it would be better to select service $\mathrm{CS}_j^N$, and, at given moment $0 \le \theta_{j\to k}^N$ perform a retry, and wait for service $\mathrm{CS}_k^N$ to complete. The expected reward without retry is

$$\mathbb{E}[W_j^N(\delta)] = -c_j^N + R\cdot F_j^N(\delta) - V\cdot(1-F_j^N(\delta)) = -c_j^N - V + (R+V)F_j^N(\delta). \quad (8.2)$$

The expected reward when a retry is made after $\theta_{j\to k}^N$ is given as:

$$\mathbb{E}[W_{j\to k}^N(\delta,\theta_{j\to k}^N)] = \underbrace{-c_j^N}_{\text{term 1}} + \underbrace{R\cdot F_j^N(\theta_{j\to k}^N)}_{\text{term 2}} + \quad (8.3)$$
$$+\underbrace{(1-F_j^N(\theta_{j\to k}^N))}_{\text{term 3}}\cdot\underbrace{\left\{-c_k^N - V + (R+V)F_k^N(\delta-\theta_{j\to k}^N)\right\}}_{\text{term 4}}.$$

Term 1 represents the costs of executing the service $\mathrm{CS}_j^N$. Term 2 represents the reward $R$ that CSP incurs when service $\mathrm{CS}_j^N$ responses before the retry moment $\theta_{j\to k}^N$, with probability $F_j^N(\theta_{j\to k}^N)$. Term 3 represents the probability there is a retry at the moment $\theta_{j\to k}^N$, and term 4 represents the expected reward when retry is made at moment $\theta_{j\to k}^N$. The retry implies that service costs $c_k^N$ are paid, and the remaining time to deadline is $\delta - \theta_{j\to k}^N$.

**Stages** $i = N-1, \ldots, 1$**:** The expected reward when service $\mathrm{CS}_j^i$ is selected, no retry is made, and with remaining time–to–deadline $\delta$, is given as

$$\mathbb{E}[W_j^i(\delta)] = -c_j^i + \int_0^\delta f_j^i(\tau) W^{(i+1)*}(\delta - \tau) d\tau + \big(1 - F_j^i(\delta)\big) \cdot W^{(i+1)*}(0). \quad (8.4)$$

The second term at the right hand side models the situation when service $\mathrm{CS}_j^i$ delivers response within time $\tau \leq \delta$, in which case the expected reward is given by the convolution of $f_j^i(\cdot)$ and the (optimal) expected reward of the next stage, i.e. $W^{(i+1)*}(\cdot)$, already defined in Equation 8.1. The third term at the right hand side of the given formula represents the probability that the service $\mathrm{CS}_j^i$ will not generate response within given deadline $\delta$, and therefore the expected reward for the remaining stages should be calculated with the deadline that equals to zero, which reduces to the selection of the cheapest service(s).

The expected reward $\mathbb{E}[W_{j\to k}^i(\delta, \theta_{j\to k}^i)]$ when service $\mathrm{CS}_j^i$ is selected and, after time interval $\theta_{j\to k}^i$ substituted by $\mathrm{CS}_k^i$, is as follows:

$$\mathbb{E}[W_{j\to k}^i(\delta, \theta_{j\to k}^i)] = \underbrace{-c_j^i}_{\text{term 1}} + \underbrace{\int_0^{\theta_{j\to k}^i} f_j^i(\tau) W^{(i+1)*}(\theta_{j\to k}^i - \tau) d\tau}_{\text{term 2}}$$
$$+ \underbrace{(1 - F_j^i(\theta_{j\to k}^i)) \cdot \mathbb{E}[W_k^i(\delta - \theta_{j\to k}^i)]}_{\text{term 3}}. \quad (8.5)$$

The first term represents the costs of $\mathrm{CS}_j^i$ execution and are paid, regardless of retry. The second term represents the expected reward when service $\mathrm{CS}_j^i$ responses within time interval $[0, \theta_{j\to k}^i]$ and in such a case no retry is made. The third term represents expected reward in case of retry, i.e. substitution of $\mathrm{CS}_j^i$ by $\mathrm{CS}_k^i$. The retry occurs with probability $(1 - F_j^i(\theta_{j\to k}^i))$ and the remaining time–to–deadline after retry is $\delta - \theta_{j\to k}^i$. The term $\mathbb{E}[W_k^i(\delta - \theta_{j\to k}^i)]$, needs to be further defined, taking into account that the remaining time–to–deadline is $\mathcal{D}^i = \delta - \theta_{j\to k}^i$, and that, for given stage $i$, *there are no further retries possible*, since our model allows only one retry per stage.

Then, $\mathbb{E}[W_k^i(\delta - \theta_{j\to k}^i)]$ becomes

$$\mathbb{E}[W_k^i\ (\delta - \theta_{j\to k}^i)] = -c_k^i + \int_0^{\delta-\theta_{j\to k}^i} f_k^i(\tau) W^{(i+1)*}(\delta - \theta_{j\to k}^i - \tau) d\tau$$
$$+\ \left(1 - F_k^i(\delta - \theta_{j\to k}^i)\right) \cdot W^{(i+1)*}(0). \tag{8.6}$$

Equation 8.6 is almost the same as Equation 8.4 since no more than one retry is possible. The only difference is the remaining deadline; therefore, second term at the right hand side of Equation 8.6 models the case when service $\mathrm{CS}_k^i$ delivers response within time $\tau \leq \delta - \theta_{j\to k}^i$. The third term at the right hand side of Equation 8.6 represents the probability that the service $\mathrm{CS}_k^i$ will not generate response within given deadline $\delta - \theta_{j\to k}^i$.

**Remark:** In case when multiple retries would be possible, e.g. substitution $\mathrm{CS}_j^i \to \mathrm{CS}_k^i \to \mathrm{CS}_l^i$, equation 8.6 would be modified in such a way that convolution would be calculated given upper limit $\theta_{k\to l}^i$ and $\mathcal{D}$ would have the value of $\delta - \theta_{j\to k}^i - \theta_{k\to l}^i$, and so on.

The end–to–end policy $\pi(\delta_p)$ for given end–to–end deadline $\delta_p$ comprises of policies $\pi^i(\delta), \delta \in [0, \delta_p]$ for stages $i = 1, 2, \ldots N$. Policy $\pi^i(\delta)$ is determined either from

$$\pi^i(\delta) = \underset{j=1,2,\ldots,M_i}{\operatorname{argmax}} \{\mathbb{E}[W_j^i(\delta)]\}, \tag{8.7}$$

or from

$$\pi^i(\delta) = \underset{\substack{j,k=1,2,\ldots,M_i \\ \theta_{j\to k}^i \in [0,\delta]}}{\operatorname{argmax}} \{\mathbb{E}[W_{j\to k}^i(\delta, \theta_{j\to k}^i)]\}. \tag{8.8}$$

The former, Equation 8.7 is used for those values of deadline $\delta$ when expected reward without retry is bigger than expected reward with retry, i.e. when $\mathbb{E}[W_j^i(\delta)] > \mathbb{E}[W_{j\to k}^i(\delta, \theta_{j\to k}^i)], \forall j, k, \theta_{j\to k}^i \in [0, \delta]$. In such a case the policy represents indices of services resulting in maximum revenue for given deadline $\delta$. The latter, Equation 8.8 is used for those values of deadline $\delta$ when retries are useful. It contains indices of services selected for deadline $\delta$, indices of services to be used for retry, as well as the optimal retry moment.

### 8.2.3 Discretization

In order to numerically evaluate $\mathbb{E}[W_j^i(\delta)]$ and $\mathbb{E}[W_{j\to k}^i(\delta, \theta_{j\to k}^i)]$ as given in Equations 8.2 to 8.6, a discretization of both time–to–deadline $\delta \in [0, \delta_p]$ and retry moment $\theta \in [0, \delta]$ is necessary. The discretization steps are denoted by $\Delta\tau$ for the deadline, and by $\Delta\theta$ for the retry moment. The equal number $m$ of discretization steps for both $\delta$ and $\theta$ within any given stage is a natural and convenient choice. In such a case, the discretization steps are $\Delta\tau = \Delta\theta = \delta_p/m$.

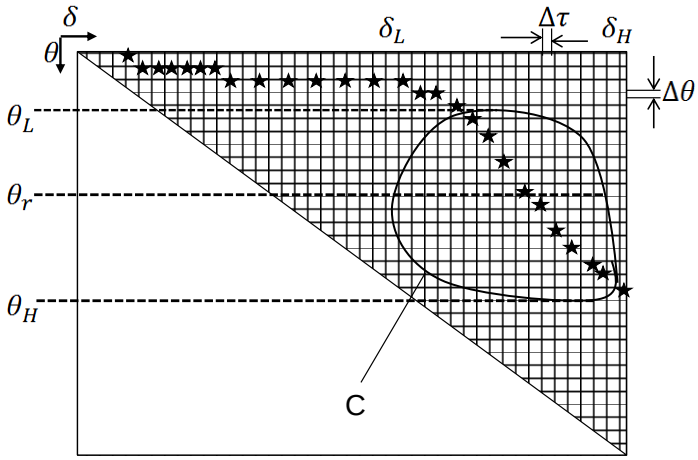


Figure 8.2: Evaluation grid for expected revenue in case of conditional retry for a single stage.

As $\mathbb{E}[W_j^i(\delta)]$ depends on $\delta$ only, the number of points in which it is evaluated is $m$. On the other hand, as $\mathbb{E}[W_{j\to k}^i(\delta, \theta_{j\to k}^i)]$ depends on both $\delta$ and $\theta_{j\to k}^i$, it is evaluated within a grid. The evaluation grid of $\mathbb{E}[W_{j\to k}^i(\delta, \theta_{j\to k}^i)]$ is illustrated in Figure 8.2. For each point within such a grid, the value of the function is calculated based on Equations 8.4 to 8.6. The number of evaluation points is therefore $m \cdot (m+1)/2$. The grid points where function $\mathbb{E}[W_{j\to k}^i(\delta, \theta_{j\to k}^i)]$ reaches maximum are represented by symbol $\star$ at Figure 8.2. Therefore, these points also specify optimal retry moments for given deadline value $\delta$.

Besides, the integrals in equations 8.5 and 8.6 are convolution integrals of respective (continuous) functions and numerical evaluation of these integrals is done by discretization of the continuous functions. The numerical evaluation of convolution integrals has been described e.g. in [75].

### 8.2.4 Policy aggregation

The optimal decision policy derived from the model, formulae and procedure presented above may be very efficiently numerically calculated. However, for each (discretized) value of $\delta$, different (discretized) value of $\theta_{j\to k}^i$ may give maximum of expected rewards when retries are made. This is further illustrated in Figure 8.2 – we see that, for each value of $\delta \in [\delta_L, \delta_H]$, different retry moment is optimal. In the worst case (and general) scenario, when the number of discretization steps is $m$ (as defined previously), as many as $m$ entries may exist for optimal retry moments, for a single stage.

Figure 8.2 illustrates contour $C$ representing area in which expected rewards with retry are larger than expected rewards without retry. We can therefore identify three areas in this figure. First, the area where $\delta < \theta$ is not considered, as retries cannot occur after the deadline expires. Second, the area bordered by contour $C$ is the one where retry leads to revenue improvements. The rest of Figure 8.2 represents area where retries do not lead to revenue improvements, hence, these should not be attempted. This means that optimal retry moments depicted by $\star$ in this area are not considered when determining policy $\pi^i(\delta)$.

The goal of policy aggregation is to reduce the number of policy entries that refer to the area bounded by contour $C$ in Figure 8.2. Using the notation from this figure, the simplest approach would be to define a single retry moment $\theta_r$ as $\theta_L + \theta_H/2$, where $\theta_L$ and $\theta_H$ are the lowest and highest value of the retry moments for contour $C$, respectively. This means that for $\delta \in [\delta_L, \delta_H]$ single retry moment $\theta_r$ is specified within the policy, thus reducing policy's size. Naturally, this is not the only possible solution for the policy aggregation, and some other, more complex solutions are discussed in [120].

The "penalty" for the aggregation is that expected revenue is reduced compared to the optimum. However, the revenue resulting from aggregated policy is still larger than expected revenue when retries are not applied.

### 8.2.5 The procedure to determine policy with conditional retries

We conclude this section with the procedure that should be followed in order to calculate policy that leads to revenue improvements when applied to service compositions using conditional request retries. It is important to realise this policy is pre–determined, i.e. developed before composite service is deployed. Therefore, we do not re–compute the policy once the service is deployed, and the decision procedure (i.e. which service is selected for the next task in the workflow and when to perform a retry) is reduced to a simple lookup. The only required parameter at runtime in order to make optimal decision is remaining time–to–deadline, $\delta$.

The inputs to the procedure are:

- The end–to–end deadline $\delta_p$
- The execution costs $c_j^i$ per request, and
- The response–time distributions for each service $\mathrm{CS}_j^i$,

where $i = 1, 2, \ldots, N$ (workflow tasks) and $j = 1, 2, ..., M_i$ (services that implement task $i$).

The procedure to calculate the policy using a pseudo code description is as follows:

**A.** Perform discretization of $\delta_p$, which results in discretized values of $\delta \in [0, \delta_p]$.

**B.** Perform discretization of $\theta$, which results in discretized values of $\theta \in [0, \delta_p]$.

**C.** Determine maxima of expected rewards without retry, i.e. $W_j^{i*}(\delta) = \max\{\mathbb{E}[W_j^i(\delta)]\}$, $\forall i, j, \delta$, as given in Equations 8.2 and 8.4.

**D.** For every task $i$

**D1.** For every substitution of service $\mathrm{CS}_j^i$ by service $\mathrm{CS}_k^i$ determine $\mathbb{E}[W_{j\to k}^i(\delta, \theta_{j\to k}^i)]$, $\forall \delta, \theta_{j\to k}^i$ as specified in Equations 8.4, 8.5 and 8.6.

**D2.** For each discrete value of $\delta$ determine $\theta^*$ that maximizes $\mathbb{E}[W_{j\to k}^i(\delta, \theta_{j\to k}^i)]$, i.e. $W_{j\to k}^{i*}(\delta, \theta^*)$.

end for

**E.** For each task $i$ and $\forall \delta$

if$(W_j^{i*}(\delta) > W_{j\to k}^{i*}(\delta, \theta^*))$ then

$\pi^i(\delta) = j$

else

$\pi^i(\delta) = j$ and after $\theta^*$ substitute $\mathrm{CS}_j^i$ by service $\mathrm{CS}_k^i$.

end if

end for

**F.** (Optional) Aggregate policy for entries where retries are beneficial.

## 8.3 Numerical experiments

In this section we will give some numerical experiments that illustrate the theory described in Section 8.2 and state some observations coming from these experiments. We show the benefits of the proposed conditional request replication by presenting simulation results in the case of four sequential workflows that consist of one, two, three, or four tasks, respectively. Each task is implemented by two different concrete services, as shown in Figure 8.3 for the workflow that consists of four tasks. This figure illustrates all possible retry attempts, but, as already emphasized, a single (best) retry is allowed for a given task. Hence, the maximum total number of retries in this particular composition is four. These example workflows are presented in order to illustrate the influence of different parameters to the potential gain in expected revenue obtained by our solution versus service compositions without the

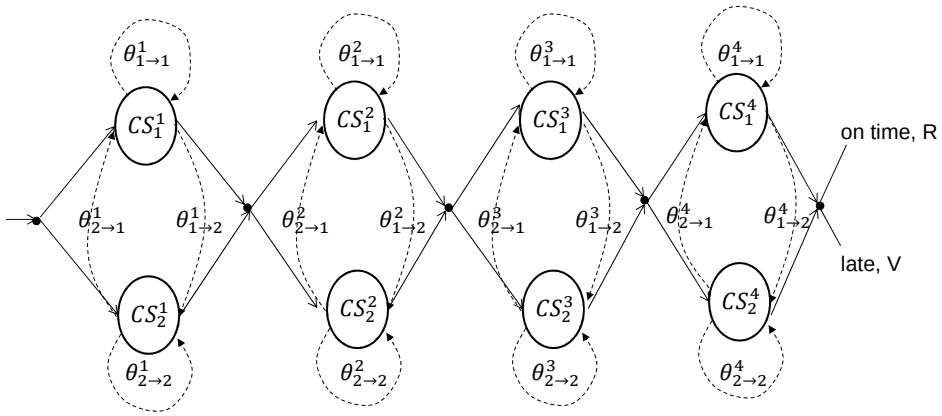


Figure 8.3: The example of a sequential workflow with four tasks used for the numerical experiments.

conditional retry mechanism. To this end, we have performed a wide variety of numerical experiments. The most illustrative results of these experiments are outlined below.

### 8.3.1 The algorithms for comparison

Two reference service composition algorithms are used to compare our results to.

*Reference* 1 – *static composition (SC)*: this reference composition determines an optimal path $\mathcal{P} = (\mathrm{CS}^1_{j_1}, \ldots, \mathrm{CS}^N_{j_N})$ for given end–to–end deadline $\delta_p$ in advance. The optimality criteria for paths considered is maximum expected revenue for given deadline. For the small scale experiment that we consider, we have used the exhaustive search among all possible service compositions to identify the optimal SC for each considered end–to–end deadline $\delta_p$. In order to determine the expected revenue for given composition, we have first calculated the probability that the deadline is met, using convolution of services' response–time distributions, and then applied Equation 8.2, with the costs being the execution costs of whole composition.

*Reference* 2 – *dynamic composition (DC)* based on dynamic programming approach without conditional retries: this run–time, dynamic reference composition is in detail explained in Chapter 6.

### 8.3.2 Parameter settings

The parameter space for our experiments is large, even when we consider only two different concrete services for every task. Therefore, we assume that services $\mathrm{CS}^i_1$ for any task $i$ have the same execution cost $c^i_1 = c_1 = 1$, and similarly, the execution

costs for services $\mathrm{CS}_2^i$ are $c_2^i = c_2 = 5$. The reward when the deadline is met has been set to $R = 12.5$, and the penalty is set to $V = 10$.

The works of [86] and [49] indicate that services' response times measured in practice show heavy–tailed properties. The services with heavy–tailed response–time distributions are "ideal candidates" to apply the retries. This is because heavy–tailed distributions experience what is often referred to as the *expectation paradox* [31]. The expectation paradox indicates that when we make observations of heavy–tailed distributions representing the web service's response time, then the longer we have waited, the longer we should expect to wait.

We have modelled the response–time distribution for all services using the Weibull distribution [101]. The (two parameter) Weibull PDF is expressed as

$$f(t) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1} e^{-\left(\frac{t}{\eta}\right)^{\beta}},\ t \geq 0.$$

Depending on parameters $\beta > 0$ (shape) and $\eta$ (scale) characteristics of Weibull distribution may be very versatile, and this has been the main reason to choose this particular distribution for the experiments, as wide range of response time distributions could be easily simulated. For example, Weibull distributions with $\beta < 1$ are heavy–tailed, while Weibull distributions with $\beta > 1$ are not. Besides, Weibull distribution is defined for non–negative random variables, which reflects the nature of the response time(s) as well.

We modelled response times of $\mathrm{CS}_1^i$ for any task $i$ by the same Weibull distribution, $f_1^i = f_1$ where $\beta = 1.2$ and $\eta = 14$ sec. The response–time distribution of $\mathrm{CS}_2^i$ for any task $i$ is modelled by $f_2^i = f_2$ where $\beta = 5.5$ and $\eta = 5$ sec. The distribution $f_1$ has a right tail that is "much longer" than the tail of the distribution $f_2$. The probability density functions $f_1$ and $f_2$ and the corresponding cumulative distribution functions of service times for services $\mathrm{CS}_1^i$ and $\mathrm{CS}_2^i$ are illustrated in Figure 8.4. Looking at this figure we see, that *on average* $\mathrm{CS}_2^i$ responses faster than $\mathrm{CS}_1^i$, and we say that $\mathrm{CS}_2^i$ is faster and more expensive than $\mathrm{CS}_1^i$. The orchestrator therefore could achieve faster service at higher expense, which is often the case in reality.

### 8.3.3 Revenue improvements

We first discuss revenue improvements when the composition with conditional retries is compared to the SC and DC reference scenarios, see Figure 8.5. The revenues are shown for the sequential workflow that consist of four tasks. We see that our (proposed) scenario is never worse than neither DC nor SC reference. Besides, Figure 8.5 illustrates that the expected revenue when our solution is applied remains positive over much wider interval of deadlines. The positive revenue with our scenario is achieved for the end–to–end deadlines larger than approximately 30 seconds, compared to respective deadlines of approximately 40 and 50 seconds for DC and SC scenarios. This is due to the fact that retries in case of relatively smaller deadlines

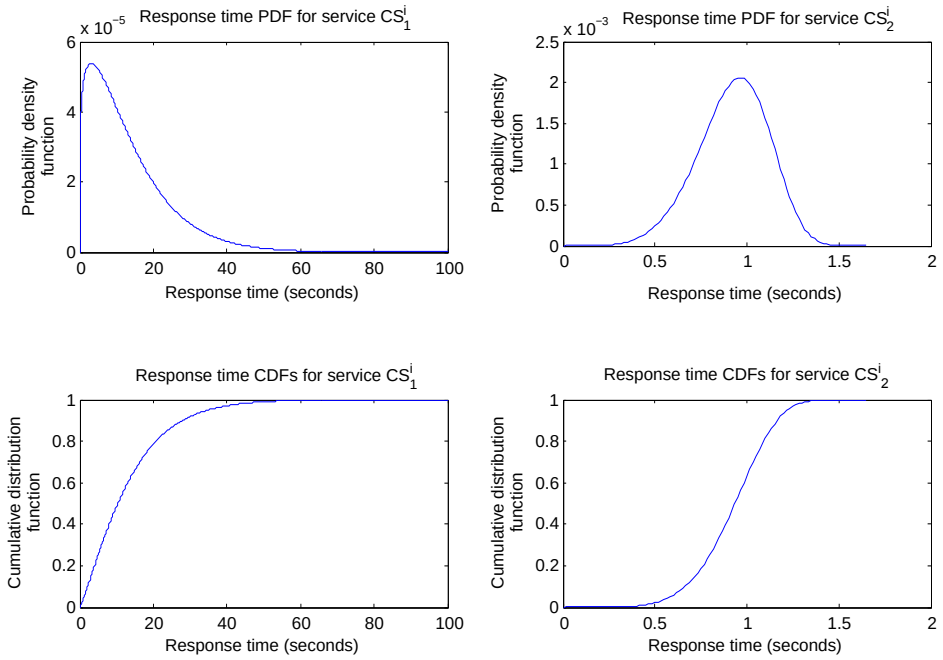


Figure 8.4: Services' response time distributions used for the experiments.

are beneficial, allowing the request to be processed within the overall deadline. As this is rewarded, the positive revenues are achieved. We see that DC (faster) and SC (slower) scenarios approach retries scenario when the value of given deadline is large, as the probability that selected service would complete execution before deadline approaches one. In such a case, no retry is necessary, and we see that from the graph.

The relative expected revenue improvements are presented in Figure 8.6, which illustrates revenue gain for the workflow that consists of four tasks. The relative gain $G$ in expected revenue obtained by using dynamic composition with conditional retries $W_{\text{retries}}$ compared to the revenue obtained using DC reference $W_{\text{DC}}$ is defined as follows:

$$G := \frac{W_{\text{retries}} - W_{\text{DC}}}{W_{\text{retries}}} \times 100\%.$$

The relative gain $G$ in case when $W_{\text{retries}}$ is compared to the revenue obtained using SC reference $W_{\text{SC}}$ is defined similarly.

As the range of relative revenue improvements for both comparisons is wide, we have decided to present relative improvements using logarithmic scale. The range of

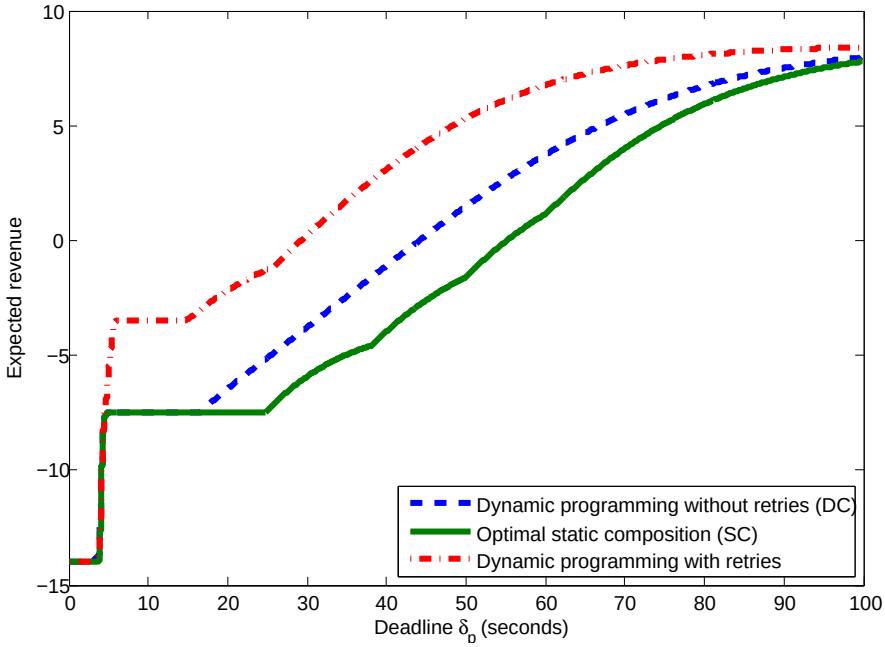


Figure 8.5: Comparison of revenues achieved with our scenario to revenues achieved with DC and SC scenarios. The comparison is shown for the sequential workflow with four tasks.

relative revenue improvements in case of comparison with DC scenario is from 5% (deadline $\delta = 100$ seconds) to over 200% ($\delta = 55$ seconds). The range of relative revenue improvements in case of comparison with SC scenario varies from around 8% ( $\delta = 100$ seconds) to over 50000% ($\delta = 55$ seconds). The latter value comes to no surprise taking into account that for $\delta = 55$ seconds revenue achieved with SC scenario is very small. From Figures 8.5 and 8.6 we see that SC scenario is inferior to other two scenarios, and therefore, we will focus on comparisons between DC scenario and our scenario with retries from now on.

Therefore, in Figure 8.7 we present the revenues achieved by both considered scenarios for workflows that consist of one, two, three and four tasks, respectively. The revenues for these workflows are represented as a function of time–to–deadline $\delta$. These revenues have non–decreasing trend, as expected, since more run–time dynamic compositions are possible for larger end–to–end deadline value $\delta_p$. We see that scenario with retries is never worse than DC reference. For small values of deadline, no retries are attempted, and considered scenarios are identical. Elsewhere is our method with retries clearly above the expected revenue of the corresponding DC reference. However, the larger the deadline $\delta_p$, the smaller is the revenue difference between the two approaches. This is explained by the fact that for large values of

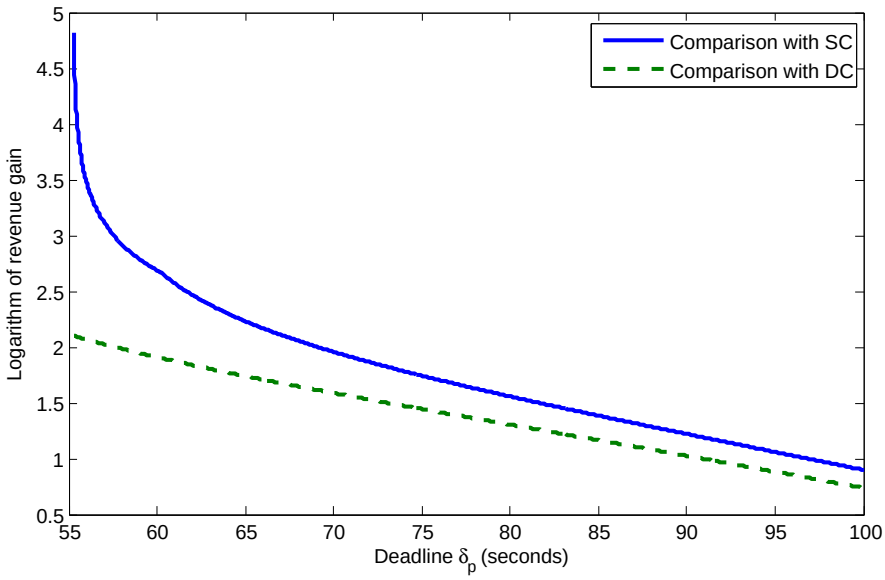


Figure 8.6: Relative gain of our approach with retries compared to DC and SC scenarios. Comparison is shown for the sequential workflow with four tasks. Note that for the smallest value of deadline shown, the revenue of SC scenario is positive, but close to zero.

deadline, the optimal policy (without retries) is to choose at each stage the cheapest service.

### 8.3.4 Comparison to DC scenario when number of retries is smaller than number of tasks

So far we have compared our solution to reference scenarios when a single retry is possible for each task in the workflow. We call this scenario the optimal scenario from now on. The model, formulae and procedure described in Section 8.2 result in optimal policy with respect to expected revenue. However, we want to analyze the case when retries are not performed for each task in the workflow. The question that needs to be addressed is whether for some tasks retries could be omitted without too much revenue loss compared to the optimal scenario.

We will therefore first analyze the revenues achieved by the optimal scenario and dynamic composition scenario with conditional retries possible for all stages except the first one. The results are shown in Figure 8.8, for the workflows that consist

of a single task, two, three and four tasks, respectively. The case of a single task workflow is already shown, as it reduces to comparison of optimal scenario and DC scenario and it is shown only for completeness. Points $A$ and $B$ in each sub–graph of Figure 8.8 specify the deadline interval $(A, B)$ in which the optimal scenario is superior to the scenario in which conditional retries are possible for each task in the workflow, except for the first one.

The larger the number of tasks in the workflow, the smaller the interval $(A, B)$. We see that for the four–tasks workflow this interval does not exist. In other words, the retry for the first task in the workflow does not play any role when it comes to revenue improvements, and revenues for two considered scenarios are overlapping. Therefore, for given scenario and given service parameters, the retry for the first task may be omitted.

Further, in Figure 8.9 we have analyzed the revenues achieved by dynamic composition for the four–task workflow with conditional (single) retry possible for all four, the last three, the last two stages, the last stage, and no stage at all (this comes down to DC scenario). We see, that for given services' parameters, the retries at the first two stages yield no extra revenue. Therefore, the policy that yields revenue that is very close to optimal could be deduced for the case when retries are performed for the last two–stages only. Naturally, the execution costs $c_2$ and $c_1$ play a significant role here, as the ratio considered is $c_2/c_1 = 5$. We also analyzed the cases when $c_2/c_1 = 2$, shown in Figure 8.10 and $c_2/c_1 = 8$ (not shown). The former case (with cost ratio 2) shows that at most three retries need to be considered. It is also clear that revenues achieved when three retries are performed do not differ much from revenues achieved when only retries at the last two stages are made. The latter case (with cost ratio 8) shows that only retry at the last stage leads to revenue improvements.

## 8.4 Single retry analysis

In this section we present the analysis of the request replication for a single task in the workflow. This case of a single retry in the workflow is the only case for which, in general, it is possible to derive exact formula for calculation of optimal (single) retry moment. We first define the formulae that could be used to find the optimal retry moment when replication is possible for the last task in the workflow. Then we analyze the optimal choice of a single request replication, i.e. at which stage should there be a possibility for a retry in order to maximize the expected revenue.

**Analysis of single retry for the last workflow task**

Let us observe the sequential workflow with $N$ tasks in which the first $N-1$ tasks have been executed with the elapsed time $\tau$, see Figure 8.11. The remaining time–to–deadline for the last task is thus $\delta = \delta_p - \tau$. In order to simplify the notation used hereafter, let us write $c_i^N = c_i,\ f_i^N = f_i,\ F_i^N = F_i,\ i = 1, 2$. Let us further

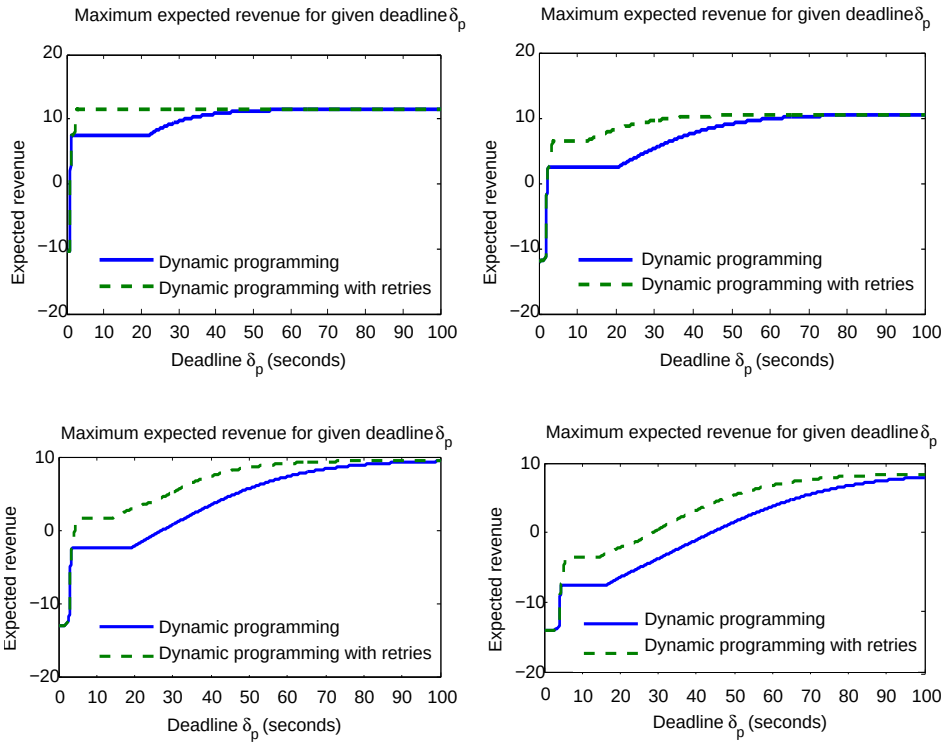


Figure 8.7: Revenues achieved by dynamic composition with and without conditional retries. The revenues are shown for sequential workflows that consist of a single task (top left), two tasks (top right), three tasks (bottom left) and four tasks (bottom right).

denote the execution costs of tasks already executed by $C_E$. The expected revenue without request replication when $\mathrm{CS}_1^N$ is selected is given as

$$\mathbb{E}[W_1^N(\delta)] = -C_E - c_1 - V + (R+V) \cdot F_1(\delta).$$

The expected revenue consists of costs incurred for the workflow tasks already executed, $C_E$, the cost of the service execution, the reward $R$ obtained when the task is executed within given deadline $\delta$ with probability $F_1(\delta)$, and the penalty $V$ paid when deadline is not met, with probability $1 - F_1(\delta)$. Naturally, when $\delta \leq 0$, it holds that $F_1(\delta) = 0$. When conditional retry mechanism is applied, the expected revenue for given deadline $\delta$ and retry moment $\theta = \theta_{1\to 2}^N$ when service $\mathrm{CS}_1^N$ is substituted by $\mathrm{CS}_2^N$ is as follows:

$$\mathbb{E}[W_{1\to 2}^N(\delta,\theta)] = -C_E - c_1 + R \cdot F_1(\theta) + (1 - F_1(\theta)) \cdot \{-c_2 - V + (R+V) \cdot F_2(\delta - \theta)\}.$$

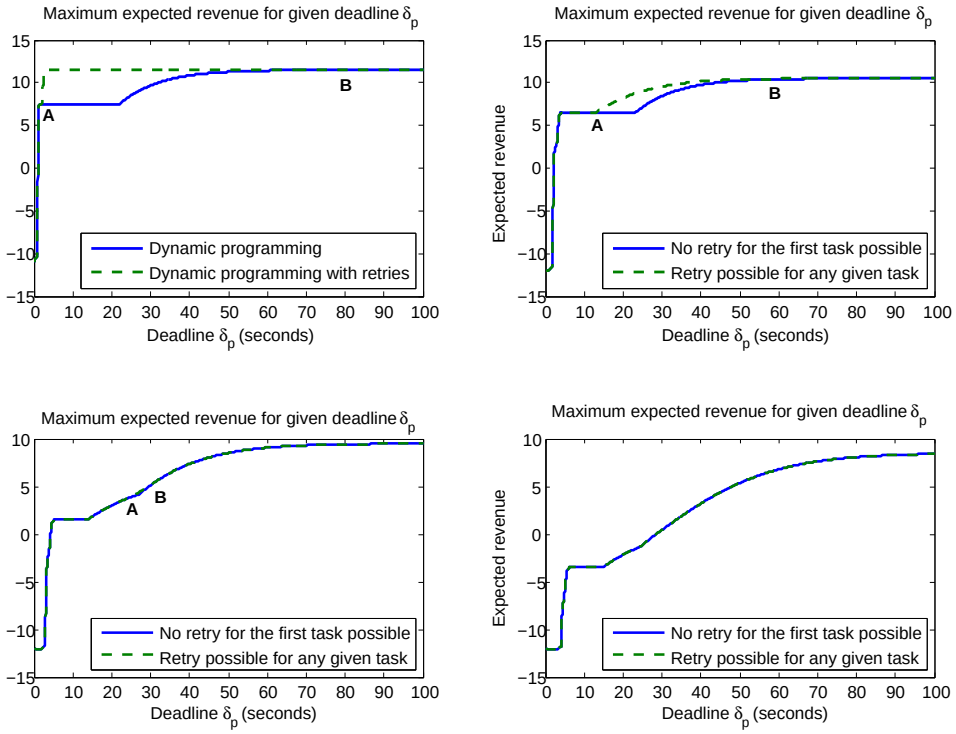


Figure 8.8: Revenues achieved by dynamic composition with conditional retries possible for all stages except the first one, and dynamic composition with conditional retries possible for all stages (original optimal scenario). The revenues are shown for sequential workflows that consist of a single task (top left), two tasks (top right), three tasks (bottom left) and four tasks (bottom right).

We see that the reward is obtained in two cases: the first case is when service $\mathrm{CS}_1^N$ completes the execution before timeout $\theta$ expires, which happens with probability $F_1(\theta)$. With probability $1 - F_1(\theta)$ we make a conditional retry. In case the retry is made, the deadline for service $\mathrm{CS}_2^N$ is $\delta - \theta$ and this deadline is met with probability $F_2(\delta - \theta)$. This also means that, when retry is made, CSP obtains reward $R$ with probability $(1 - F_1(\theta)) \cdot F_2(\delta - \theta)$.

The optimal value $\theta = \theta^*$ represents retry moment for which $\mathbb{E}[W_{1\to 2}^N(\delta, \theta)]$ reaches maximum at interval $[0, \delta]$. We consider the case when $\mathbb{E}[W_{1\to 2}^N(\delta, \theta)]$ is continuous in $\theta$, so the optimal value $\theta^*$ satisfies the following condition

$$\frac{\partial \mathbb{E}[W_{1\to 2}^N(\delta, \theta)]}{\partial \theta}|_{\theta=\theta^*} = 0.$$

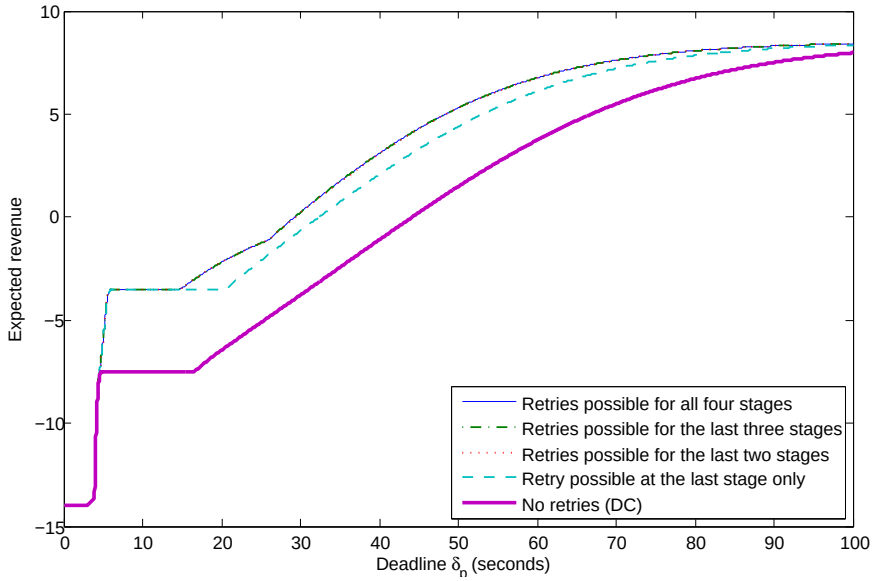


Figure 8.9: Revenues achieved by dynamic composition for the four–task workflow with conditional retries possible for the last four, three, two stages, only the last stage, and not possible at all (DC scenario), with retry execution cost $c_2 = 5$. The graphs when retries are possible for all four tasks, the last three and the last two tasks are practically identical

Elementary transformations give the following expression

$$\frac{f_1(\theta^*)}{1-F_1(\theta^*)} + \frac{c_2}{R+V} \cdot \frac{f_1(\theta^*)}{1-F_1(\theta^*)} \cdot \frac{1}{1-F_2(\delta-\theta^*)} = \frac{f_2(\delta-\theta^*)}{1-F_2(\delta-\theta^*)}.$$

Solving this equation for given response time distributions allow us to determine the optimal retry moments for the last task in the workflow. This equation could also be expressed by its equivalent form, namely

$$h_1(\theta^*) \cdot \left\{ 1 + \frac{c_2}{R+V} \cdot \frac{1}{1-F_2(\delta-\theta^*)} \right\} = h_2(\delta-\theta^*),$$

where $h_1(t)$ and $h_2(t)$ are hazard–rate functions, defined by

$$h_1(t) = \frac{f_1(t)}{1-F_1(t)}, \quad \text{and} \quad h_2(t) = \frac{f_2(t)}{1-F_2(t)}.$$

Other than results from [74, 97, 98, 103] the cost structure plays important role in determining the optimal timeout value $\theta^*$. Besides the optimal value does not depend from the costs of the first attempt, $c_1$. It is trivial to determine the $\theta^*$ when the same service is considered for the conditional retry.

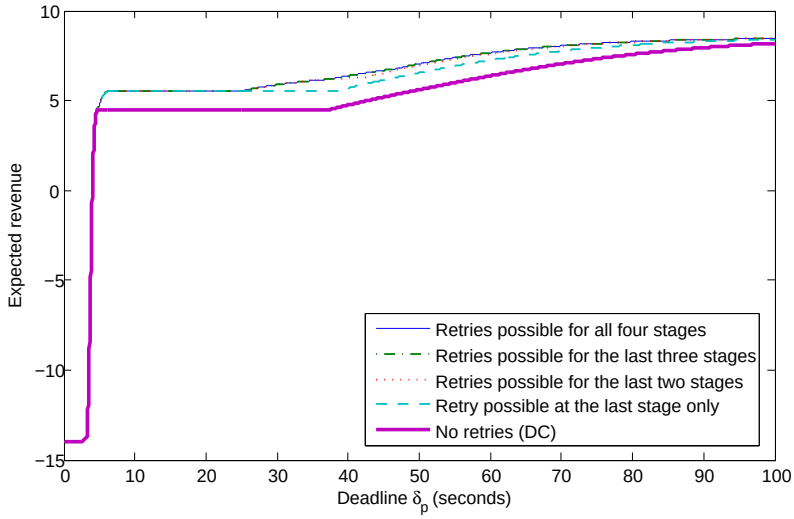


Figure 8.10: Revenues achieved by dynamic composition for the four–task workflow with conditional retries possible for the last four, three, two stages, only the last stage, and not possible at all (DC scenario), with retry execution cost $c_2 = 2$. The graphs when retries are possible for all four task and the last three tasks are practically identical.

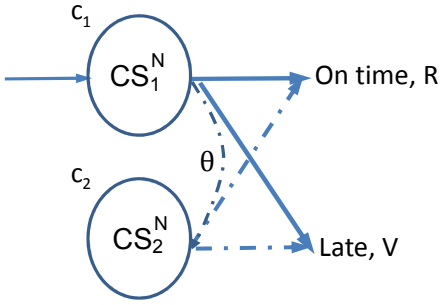


Figure 8.11: The execution of the last task with the conditional request replication. Remaining time to deadline is $\delta$, and the retry moment is $\theta$.

### Optimal task choice for a single retry

Suppose we have a sequential workflow with $N$ tasks, implemented by services $\mathrm{CS}_1^i$, $i = 1, 2, \ldots N$. The corresponding response times $X_i$ of all services are random variables which are i.i.d. and the execution costs are the same and equal $c_1$. There

is a single service used for substitution, $\mathrm{CS}_2^i$, which corresponding response time is random variable $Y$ that dominates $X_i$. The question we briefly address here is: if there is an opportunity to perform a single retry within the workflow, which task $i = i^*$ should be chosen for conditional retry in order to maximize the expected reward of the composite service provider?

If we choose to consider the substitution for the last task $N$, the orchestrator has the full information of the actual service times of the other, already executed services. The optimal retry moment is then $\theta^*$. If the orchestrator performs retry for task $i \neq N$, the orchestrator has less information. Another way to explain this is: when the orchestrator performs the retry for the last task, it may ignore all the information that is revealed between stages $i$ and $N$. However, the orchestrator uses the information gathered during the execution of the last task $N$ as it unfolds. When the optimal retry moment $\theta$ is solely based on the information gathered within the last task, it would not be better than the optimal choice based on all information up till stage $N$, i.e. $\theta^*$. This retry moment $\theta$ is the same as when the "stage with reply" is placed e.g. in between stages $i$ and $(i+1)$. Thus for any realization of random variables $X_1, \ldots, X_{N-1}$, the composition where the retry is made at the last stage does not result in a lower expected revenue.

The conclusion is that the optimal task choice for a single retry is the last task in the sequential workflow. This holds for the case when initial composition comprises of services with identical costs and the same response time distributions.

## 8.5 Conclusions

In this chapter we have investigated a run–time service adaptation mechanism for service compositions that is based on conditional retries. When the execution of a concrete service within composition lasts too long a retry to an alternative service (or, possibly, the same service) may be issued. We have developed an approach based on dynamic programming to calculate the optimal time instances when request replication should be done, and the service that should be invoked in case of a retry.

Depending on the actual model parameters, the calculated policy may become complex, which leads to a large lookup table. The retry moments are calculated for discretized values of deadline, and theoretically, each value of deadline may have a unique value of retry moment. One sub–optimal yet practical approach to calculate policy in such case may be to aggregate the retry moments, as discussed in this chapter. We quantified the revenue improvements of our conditional retry scheme compared to orchestrated compositions that do not apply retries as studied in Chapter 6. Further, we investigated how much is gained by our mechanism when retry is possible for a limited set of tasks in a workflow. We showed for the analyzed scenarios that the majority of revenue improvements result from the retries that are only possible for the last couple of tasks within the workflow.

CHAPTER 9

# Concluding remarks and directions for future research

The focus of this thesis is on end–to–end QoS control mechanisms and revenue optimization in SOA. One of the most important features of the SOA paradigm is the opportunity to develop so–called composite services. However, as the services used for composition are usually deployed in a volatile execution environment, performance–related problems could occur relatively often during services' execution processes. For example, a concrete service used for service composition that performs poorly has a potentially big impact to the end–to–end QoS of a composite service. This leads to client's dissatisfaction and loss of revenue for the CSP.

The main issue we faced is that, although available QoS–control concepts are powerful, little is known about how to cost–efficiently exploit the possibilities for end–to–end QoS–control in SOA. A complicating factor in controlling QoS in service–oriented applications is that the ownership of the services used for the composition is decentralized, i.e. these services are offered by third parties, each with their own business incentives. In this thesis, we considered in particular admission control, and dynamic service selection for QoS–aware service composition as the end–to–end QoS–control mechanisms. These mechanisms allow for per–request state–dependent control decisions, i.e. *per–request, per–task* QoS–control.

In light of this, the main research questions that we investigated in this thesis are:

1. How to model the effect of the parameter settings of the QoS–control mechanisms on the end–to–end QoS? The models should capture the dominant factors that influence QoS yet allow for (mathematical) analysis and optimization.

2. How to analyse and use these models to derive optimal settings of the parameters of the control mechanisms? The optimal settings should maximize (long–term) expected revenue subject to pre–set QoS requirements and related costs and rewards/penalties.

The models designed in the chapters 3 to 8 respectively, address the first research question posted. The results of the analysis and optimal parameter settings of the models that address the second research question posted, are presented in each of

the respective chapters. Therefore, we shed a light on the main results relevant for the answer to the second question in what follows.

In Chapter 3 we used queueing–theory based models to derive practical admission control rules. The main objective of these rules is to keep both end–to–end response time and availability at agreed QoS levels. As the rules are designed to be aware of the execution state within the composition, they also accommodate for per–request, per–task QoS–control. This results in dramatic improvements of end–to–end performance metrics, in particular the average number of successfully served requests per second.

In Chapter 4 the performance potential of run–time Web service selection is investigated for a single task implemented by a number of alternative concrete services. The queueing–theory based models are used for performance comparison of different service selection strategies in realistic scenarios that include the presence of background traffic and stale system state information. We demonstrate the effectiveness of certain service selection strategies, and illustrate the spectacular response–time improvements that can be obtained. In fact, the observed performance improvements result from exploiting workload fluctuations that occur at relatively small time scales mainly caused by the random behaviour of potential clients.

In Chapter 5 we designed models to identify optimal stopping policies that maximize the profit of the CSP given the end–to–end QoS constraints. Depending on the actual response times of executed services, taking into account the execution costs as well as revenue and penalty functions, a dynamic programming based algorithm and a heuristic solution are compared. For both solutions the huge revenue gains are quantified when compared to the baseline case of static service composition.

In Chapters 6 to 8 we develop models for the analysis of various QoS–aware per–request per–task service composition approaches. Using these models, we formulate algorithms to identify optimal policies for service compositions. The derived policies are optimal with respect to the profit of the CSP.

First, in Chapter 6 we consider the case when for each of the tasks within a given workflow, a number of service alternatives may be available. The results do not only indicate *that* there is enormous potential gain compared to other, non–dynamic approaches, but also show *how* one can realize such gain. In particular, we have shown that the optimal service selection policy can be implemented as a lookup table, which also means negligible decision–making time.

Further, in Chapter 7 we extend the model derived in chapter 6 such that certain assurance could be stated that end–to–end response time would be met in presence of partially available services used for the composition. We have shown how to determine the policy that would result in maximum revenue for a given target quality assurance. Finally, in Chapter 8 we develop models of conditional retry mechanisms for service composition. These mechanisms are used to control end–to–end response time in presence of temporarily unresponsive services. Based on the model we derive the optimal moments to decide a particular service is unresponsive, which leads to

decisions to perform a retry using an alternative service that implements the same task.

Overall, this thesis puts a spotlight on the use of the possibilities for efficient end–to–end QoS control within SOC. We developed and analysed the quantitative models that describe the effect of QoS–control actions. Based on these models, we presented a number of solutions that dramatically improve perceived QoS by the clients and expected revenue of the CSP. This is possible due to our per–request, per–task, dynamic, QoS–aware service selection, that takes into account the associated costs of QoS–control actions as well. The proposed solutions in form of policies have a very small footprint, i.e. the algorithms and rules presented require nothing more than simple implementation in the form of, e.g. lookup table. Naturally, the realisation of our algorithms and policies for actual web services requires attention in the future.

**Directions for future research**

There are a couple of opportunities to extend the work presented in this thesis. The first step would be to fine–tune the models, and to implement and validate proposed algorithms using commercial composite services. For such a validation in practical environment, the development of a QoS–control architecture framework that supports run–time service orchestration is necessary, and our first steps towards this direction are presented in [34].

Next to this, in Chapters 5 to 8 we have considered the case when response–time SLOs are time–invariant, i.e. these SLOs, represented by respective PDFs, do not change. In practice however, the response–time characteristics may be subject to change over time (e.g. day/night usage patterns or "flash crowds"). Besides, in the models we described and analysed within this thesis, we have neglected the dependencies between response times of subsequent requests to the same service. In reality such dependence often exists and may have a main impact on the response–time performance of subsequent requests. This may mean that the actual response times deviate from the SLO specifications, and that QoS–control policies derived using our models are therefore not any more optimal. In this context, an important next step is to devise and implement models and methods to support *closed–loop control*, where the QoS–control policies are recalculated in an optimal way (e.g., with respect to the frequency of response–time updates). The results presented in this thesis form an excellent basis for extension towards closed–loop controlled system.

We have mainly conducted our analyses for the case of a sequential workflow, under the assumption that other workflow patterns could be aggregated into a sequential one. The workflow aggregation naturally leads to an issue of a coarser (i.e. sub–optimal) decision–making process. Another issue with the workflow aggregation is whether developed models should be accommodated for different workflow patterns and complex workflows. Besides, the dynamic programming solution considered may tend to slow down when the number of services is extremely large. Therefore, to provide good service quality for complex compositions of many services, there is a need for the development of fast yet efficient heuristic solutions.

# Bibliography


[1] M. Abundo, V. Cardellini, and F. L. Presti. An MDP-based Admission Control for Service-oriented Systems. Technical Report RR-11.86, University of Roma "Tor Vergata", 2011.

[2] M. Alrifai and T. Risse. Combining global optimization with local selection for efficient QoS-aware service composition. In *Proceedings of the 18th International Conference on World Wide Web*, (WWW '09), pages 881–890, Madrid, Spain, 2009.

[3] E. Altman, U. Ayesta, and B. Prabhu. Load balancing in processor sharing systems. In *Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools*, (ValueTools '08), pages 12:1–12:10, Athens, Greece, 2008.

[4] Apache. Apache Axis2. `http://ws.apache.org/axis2/`. Accessed July 2013.

[5] Apache. Apache JMeter. `http://jakarta.apache.org/jmeter`. Accessed July 2013.

[6] Apache. Apache Tomcat. `http://tomcat.apache.org`. Accessed July 2013.

[7] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani. Paws: A framework for executing adaptive web-service processes. *IEEE Softw.*, 24(6):39–46, 2007.

[8] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Trans. Softw. Eng.*, 33(5):369–384, 2007.

[9] K. J. Arrow, D. Blackwell, and M. A. Girshick. Bayes and minimax solutions of sequential decision problems. *Econometrica*, 17(3/4):213–244, 1949.

[10] J. Aweya, M. Ouellette, D. Y. Montuno, B. Doray, and K. Felske. An adaptive load balancing scheme for web servers. *Int. J. Netw. Manag.*, 12(1):3–39, 2002.

[11] S. Baker and S. Dobson. Comparing service–oriented and distributed object architectures. In *Proceedings of the 2005 Confederated international conference on On the Move to Meaningful Internet Systems*, pages 631–645, Springer–Verlag, Berlin–Heidelberg, 2005.

[12] H. Bannazadeh and A. Leon-Garcia. A distributed probabilistic commitment control algorithm for service-oriented systems. *IEEE Trans. on Netw. and Serv. Manag.*, 7(4):204–217, 2010.

[13] L. Baresi, C. Ghezzi, and S. Guinea. Towards self-healing composition of services. In *Contributions to Ubiquitous Computing*, volume 42 of *Studies in Computational Intelligence*, pages 27–46. Springer–Verlag, Berlin–Heidelberg, 2007.

[14] N. Bartolini, G. Bongiovanni, and S. Silvestri. Self-* through self-learning: Overload control for distributed web systems. *Comput. Netw.*, 53(5):727–743, 2009.

[15] R. Bellman. *Dynamic Programming*. Dover Publications, Inc., Mineola, New York, the second edition, 2003.

[16] R. Bohn and J. Short. Measuring consumer information. *International Journal of Communication*, 6:980–1000, 2012.

[17] K. Boloor, R. Chirkova, T. Salo, and Y. Viniotis. Analysis of Response Time Percentile Service Level Agreements in SOA-Based Applications. In *Proceedings of the Global Communications Conference*, GLOBECOM (2011), pages 1–6, 5–9 December, Houston, TX, USA, 2011.

[18] F. Bonomi. On job assignment for a parallel system of processor sharing queues. *IEEE Trans. Comput.*, 39(7):858–869, 1990.

[19] S. C. Borst. Optimal probabilistic allocation of customer types to servers. *SIGMETRICS Perform. Eval. Rev.*, 23(1):116–125, 1995.

[20] D. Box. Code name indigo: A Guide to Developing and Running Connected Systems with Indigo. `http://msdn.microsoft.com/en-us/magazine/cc164026.aspx`, January 2004. Accessed July 2013.

[21] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. An approach for QoS-aware service composition based on genetic algorithms. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, (GECCO '05), pages 1069–1075, 2005.

[22] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. QoS-aware replanning of composite Web services. In *Proceedings of the IEEE International Conference on Web Services*, (ICWS 2005), pages 121–129, July 11-15, Orlando, FL, USA, 2005.

[23] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. A framework for QoS-aware binding and re-binding of composite web services. *J. Syst. Softw.*, 81(10):1754–1769, 2008.

[24] J. Cao and C. Nyberg. An Approximate Analysis of Load Balancing using Stale State Information for Servers in Parallel. In *Proceedings of the Second International Conference on Communications, Internet, and Information Technology*, (IASTED 2003), pages 276–286. ACTA Press, 2003.

[25] V. Cardellini, E. Casalicchio, V. Grassi, and F. L. Presti. Adaptive management of composite services under percentile-based service level agreements. In *Proceedings of 8th International Conference on Service Oriented Computing*, ((ICSOC 2010), pages 381–395, 2010.

[26] H. Chen and P. Mohapatra. Overload control in QoS-aware web servers. *Comput. Netw.*, 42(1):119–133, 2003.

[27] L. Cherkasova and P. Phaal. Session-based admission control: A mechanism for peak load management of commercial web sites. *IEEE Trans. Comput.*, 51(6):669–685, 2002.

[28] G. L. Choudhury and D. J. Houck. Combined queueing and activity network based modeling of sojourn time distributions in distributed telecommunication systems. In *Proceedings of 14th International Teletraffic Congress*, (ITC 14), pages 525–534, 1994.

[29] Y.-C. Chow and W. H. Kohler. Models for dynamic load balancing in a heterogeneous multiple processor system. *IEEE Trans. Comput.*, 28(5):354–361, 1979.

[30] E. G. Coffman, Jr., R. R. Muntz, and H. Trotter. Waiting time distributions for processor-sharing systems. *J. ACM*, 17(1):123–130, 1970.

[31] M. E. Crovella. Performance Evaluation with Heavy Tailed Distributions (Extended Abstract). In *Job Scheduling Strategies for Parallel Processing*, pages 1–10, 1991.

[32] Département d'Informatique et de Recherche Opérationnelle (DIRO), Université de Montréal. Stochastic Simulation in Java, SSJ. `http://www.iro.umontreal.ca/~simardr/ssj/indexe.html`. Accessed July 2013.

[33] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma. Dynamic workflow composition: Using markov decision processes. *Int. J. Web Service Res.*, 2(1):1–17, 2005.

[34] L. Duijvestein, J. W. Bosman, R. van der Mei, H. B. Meeuwissen, and M. Živković. Run–time quality–of–service control framework for composite services. *Submitted*, 2013.

[35] S. Dustdar and W. Schreiner. A survey on web services composition. *International Journal on Web and Grid Services*, 1:1–30, 2005.

[36] D. Dyachuk. *Ensuring Service Level Agreements for Composite Services by Means of Request Scheduling*. PhD thesis, University of Saskatchewan, Saskatoon, Canada, 2011.

[37] D. Dyachuk and R. Deters. Scheduling of Composite Web Services. In *OTM Workshops*, volume 4277 of *Lecture Notes in Computer Science*. Springer–Verlag, 2006.

[38] D. Dyachuk and R. Deters. Improving Performance of Composite Web Services. In *IEEE International Conference on Service-Oriented Computing and Applications*, (SOCA '07), pages 147–154, 19-20 June, Newport Beach, CA, USA, 2007.

[39] S. Elnikety, E. Nahum, J. Tracey, and W. Zwaenepoel. A method for transparent admission control and request scheduling in e-commerce web sites. In *Proceedings of the 13th International Conference on World Wide Web*, (WWW '04), pages 276–286, 17–22 May, New York City, NY, USA, 2004.

[40] T. Erl. *Service–Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[41] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[42] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, 2002.

[43] L. Flatto and H. P. McKean. Two queues in parallel. *Comm. on Pure and App. Mathematics*, 30:255–263, 1977.

[44] A. Gao, D. Yang, S. Tang, and M. Zhang. Web service composition using markov decision processes. In *Proceedings of the 6th international conference on Advances in Web-Age Information Management*, WAIM'05, pages 308–319, 2005.

[45] K. Gottschalk, S. Graham, H. Kreger, and J. Snell. Introduction to web services architecture. *IBM Syst. J.*, 41(2):170–177, 2002.

[46] M. H. D. Groot. *Optimal Statistical Decisions*. John Wiley & Sons, Hoboken, NJ, WCL edition, 2004.

[47] J. Guitart, J. Torres, and E. Ayguadé. A survey on performance management for internet applications. *Concurrency Comput.: Pract. Exper.*, 22(1):68–106, 2010.

[48] V. Gupta, M. Harchol-Balter, K. Sigman, and W. Whitt. Analysis of join-the-shortest-queue routing for web server farms. *Perform. Eval.*, 64(9-12):1062–1081, 2007.

[49] S. Haddad, L. Mokdad, and S. Youcef. Response time of BPEL4WS constructors. In *Proceedings of 2010 IEEE Symposium on Computers and Communications*, (ISCC '10), pages 695–700, June 22-25, Riccione, Italy, 2010.

[50] G. J. Hoekstra, R. D. van der Mei, Y. Nazarathy, and A. P. Zwart. On The Sojourn Time Tails Of A File-Splitting Processor Sharing Network. In *Network Control and Optimization*, volume 5894 of *Lecture Notes in Computer Science*. Springer–Verlag, 2009.

[51] W. Hoschek. The Web Service Discovery Architecture. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, (Supercomputing '02), pages 1–15, November 18-21, Los Alamitos, CA, USA, 2002.

[52] M. N. Huhns and M. P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.

[53] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-services: a look behind the curtain. In *Proceedings of the 22nd ACM SIGMOD symposium on Principles of database systems*, (PODS '03), pages 1–14, San Diego, California, 2003.

[54] IBM. Big Data and the speed of business. `http://www-01.ibm.com/software/data/bigdata/`, 2002. Accessed July 2013.

[55] ISO. Quality vocabulary (ISO/IEC Standard 8402). Iso, International Organization for Standardization, Geneva, Switzerland, 1986.

[56] M. C. Jaeger, G. Rojec-Goldmann, and G. Mühl. QoS Aggregation for Web Service Composition using Workflow Patterns. In *Proceedings of the Eighth IEEE International Conference on Enterprise Distributed Object Computing*, (EDOC '04), pages 149–159, 20-24 September, Monterey, CA, USA, 2004.

[57] V. Kanodia and E. Knightly. Multi-class latency-bounded web services. In *Proceedings of the Eighth International Workshop on Quality of Service (QoS)*, (IWQoS 2000), pages 231–239, 2000.

[58] M. Kihl and N. Widell. Admission control schemes guaranteeing customer QoS in commercial web sites. In *Proceedings of IFIP Conference on Network Control and Engineering for QoS, Security and Mobility*, pages 305–316, Paris, France, 2002.

[59] E. Kim and Y. Lee. OASIS Web Services Quality Model TC. Technical report, OASIS, 2005.

[60] L. Leemis. The minimum of $n$ mutually independent and identically distributed geometric random variables is geometric. `http://www.math.wm.edu/~leemis/chart/UDR/PDFs/GeometricM.pdf`. Accessed October 2013.

[61] P. Leitner. Ensuring Cost-Optimal SLA Conformance for Composite Service Providers. In *Proceedings of the PhD Symposium of the International Conference on Service-Oriented Computing 2009*, (ICSOC 2009), pages 290–297, 2009.

[62] P. Leitner. *On preventing Service Level Agreement Violations in Composed Services using Self-Adaptation.* PhD thesis, Technische Universität Wien, Austria, 2011.

[63] G. A. Lewis, E. Morris, S. Simanta, and L. Wrage. Common misconceptions about service-oriented architecture. In *Proceedings of the Sixth International IEEE Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems*, (ICCBSS '07), pages 123–130, 2007.

[64] Y. Liu, A. H. Ngu, and L. Z. Zeng. QoS computation and policing in dynamic web service selection. In *Proceedings of the 13th International Conference on World Wide Web*, (WWW '04), pages 66–73, 17–22 May, New York City, NY, USA, 2004.

[65] Z. Liu, N. Niclausse, and C. Jalpa-Villanueva. Traffic model and performance evaluation of web servers. *Perform. Eval.*, 46(2-3):77–100, 2001.

[66] Maddy. Maddy's blog - Six blind men and elephant. `http://maddy06.blogspot.nl/2012/10/six-blind-men-and-elephant.html`, 2002. Accessed July 2013.

[67] A. Mani and A. Nagarajan. Understanding quality of service for Web Services. `http://www.ibm.com/developerworks/webservices/library/wsquality/index.html`, 2002. Accessed July 2013.

[68] P. J. Meulenhoff, D. R. Ostendorf, M. Živković, H. B. Meeuwissen, and B. M. M. Gijsen. Intelligent overload control for composite web services. In *Proceedings of 7th International Conference on Service-Oriented Computing*, (ICSOC '09), pages 34–49, 24-27 November, Stockholm, Sweden, 2009.

[69] M. Mitzenmacher. How useful is old information? *IEEE Trans. Parallel Distrib. Syst.*, 11(1):6–20, 2000.

[70] M. D. Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, University of California, Berkeley, California, USA, 1996.

[71] R. D. Nelson and T. K. Philips. An approximation to the response time for shortest queue routing. *SIGMETRICS Perform. Eval. Rev.*, 17(1):181–189, 1989.

[72] R. D. Nelson and T. K. Philips. An approximation for the mean response time for shortest queue routing with general interarrival and service times. Technical Report RC15429, IBM T.J. Watson Research Lab, 1990.

[73] L. O'Brien, P. Merson, and L. Bass. Quality attributes for service-oriented architectures. In *Proceedings of the International Workshop on Systems Development in SOA Environments*, (SDSOA '07), 2007.

[74] H. Okamura, T. Dohi, and K. S. Trivedi. On–line Adaptive Algorithms in Autonomic Restart Control. In *Proceedings of the 7th international conference on Autonomic and Trusted Computing* (ATC 2010), volume 6407 of *Lecture Notes in Computer Science*, pages 32–46, Springer–Verlag, Berlin–Heidelberg, 2010.

[75] A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete-time signal processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, the second edition, 1999.

[76] A. Padovitz, S. Krishnaswamy, and S. W. Loke. Towards Efficient Selection of Web Services. In *2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2003.

[77] M. P. Papazoglou and D. Georgakopoulos. Introduction: Service-oriented computing. *Commun. ACM*, 46(10):24–28, 2003.

[78] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.

[79] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[80] T. Pilioura and A. Tsalgatidou. E-services: Current technology and open issues. In *Proceedings of Second International Workshop on Technologies for E-Services*, (TES 2001), pages 1–15, 14-15 September, Rome, Italy, 2001.

[81] S. R. Ponnekanti and A. Fox. SWORD: A developer toolkit for web service composition. In *Proceedings of the 11th International WWW Conference*, (WWW2002), Honolulu, HI, USA, 2002.

[82] S. Ran. A model for web services discovery with QoS. *SIGecom Exch.*, 4(1):1–10, 2003.

[83] S. Rosario, A. Benveniste, S. Haar, and C. Jard. Probabilistic QoS and soft contracts for transaction-based web services orchestrations. *IEEE Trans. Serv. Comput.*, 1(4):187–200, 2008.

[84] A. Sahai, V. Machiraju, M. Sayal, A. P. A. v. Moorsel, and F. Casati. Automated SLA monitoring for web services. In *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems, Operations and Management*, (DSOM '02), pages 28–41, Springer–Verlag, London, UK, 2002.

[85] J. G. Saxe. The Blind Men and the Elephant. `http://en.wikisource.org/wiki/The_Blindmen_and_the_Elephant`. Accessed July 2013.

[86] M. Scharf. On the Response Time of Large-scale Composite Web Services. In *Proceedings of the 19th International Teletraffic Congress, series = (ITC 19)*, 2005.

[87] R. W. Schulte and Y. V. Natis. Service Oriented Architecture. SSA Research Note SPA-401-068, Gartner Group, 1996.

[88] Y. A. Shaaban and J. Hillston. Cost-based admission control for internet commerce QoS enhancement. *Electron. Commer. Rec. Appl.*, 8(3):142–159, 2009.

[89] S. Sharifian, S. A. Motamedi, and M. K. Akbari. A content-based load balancing algorithm with admission control for cluster web servers. *Future Gener. Comput. Syst.*, 24(8):775–787, 2008.

[90] A. Strunk. QoS-Aware Service Composition: A Survey. In *Proceedings of 8th European Conference on Web Services*, (ECOWS 2010), pages 67–74, 2010.

[91] Tecnomatix. *eM-Plant 7.0 Manual*. Tecnomatix GmbH, Stuttgart, Germany, 2004.

[92] H. C. Tijms. *A First Course in Stochastic Models*. Wiley, 2003.

[93] UDDI.org. UDDI Technical White Paper. `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`, 2000. Accessed July 2013.

[94] B. Urgaonkar and P. Shenoy. Cataclysm: Scalable overload policing for internet applications. *J. Netw. Comput. Appl.*, 31(4):891–920, 2008.

[95] W. M. P. van der Aalst, A. H. M. t. Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.

[96] R. D. van der Mei, B. Gijsen, and P. J. Meulenhoff. Overload control for web services: Web admission control. In *Proceedings of 30th International Computer Measurement Group Conference*, (CMG '04), pages 689–696, December 5-10, 2004, Las Vegas, NV, USA, 2004.

[97] A. P. A. van Moorsel and K. Wolter. Optimal restart times for moments of completion time. In *Proceedings of UK Performance Engineering Workshop*, pages 219–223, 2004.

[98] A. P. A. van Moorsel and K. Wolter. Analysis of restart mechanisms in software systems. *IEEE Trans. Softw. Eng.*, 32(8):547–558, 2006.

[99] H. Wang, X. Zhou, X. Zhou, W. Liu, W. Li, and A. Bouguettaya. Adaptive service composition based on reinforcement learning. In *Proceedings of the 2010 International Conference of Service Oriented Computing*, (ICSOC 2010), pages 92–107, 2010.

[100] C. Ward, M. J. Buco, R. N. Chang, and L. Z. Luan. A Generic SLA Semantic Model for the Execution Management of e-Business Outsourcing Contracts. In *Proceedings of the Third International Conference on E-Commerce and Web Technologies*, (EC-WEB '02), pages 363–376, 2002.

[101] Wikipedia. Wikipedia: Weibull distribution. `http://en.wikipedia.org/wiki/Weibull_distribution`. Accessed July 2013.

[102] M. Wirsing, A. Clark, S. Gilmore, M. Hölzl, A. Knapp, N. Koch, and A. Schroeder. Semantic-based development of service-oriented systems. In *Proceedings of the 26th IFIP International Conference on Formal Techniques for Networked and Distributed Systems*, (FORTE '06), pages 24–45, Springer–Verlag, Berlin–Heidelberg, 2006.

[103] K. Wolter. *Stochastic Models for Fault Tolerance - Restart, Rejuvenation and Checkpointing*. Springer–Verlag, 2010.

[104] World Wide Web Consortium (W3C). SOAP version 1.2 part0: Primer.

[105] World Wide Web Consortium (W3C). Web Services Description Language (WSDL) Version 2.0 Part 0: Primer - W3C Candidate Recommendation 27 March 2006. `http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327/`. Accessed July 2013.

[106] D. Worm, M. Živković, H. van den Berg, and R. van der Mei. Revenue maximization with quality assurance for composite web services. In *Proceedings of Fifth IEEE International Conference on Service-Oriented Computing and Applications*, (SOCA '12), 17-19 December, Taipei, Taiwan, 2012.

[107] B. Xi. Quality of service (QoS) for web-based applications. Technical report, TNO-ICT and Eindhoven University of Technology, 2007.

[108] H. Xianglan, L. Yangguang, X. Bin, and Z. Gang. A survey on QoS-aware dynamic web service selection. In *Proceedings of 7th International Conference on Wireless Communications, Networking and Mobile Computing*, (WiCOM '11), pages 1–5, 2011.

[109] Z. Xu and G. v. Bochmann. A probabilistic approach for admission control to web servers. In *Proceedings of International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, (SPECTS 2004), pages 787–794, San Jose, CA, USA, 2004.

[110] A. Yousefi and D. G. Down. Request replication: An alternative to QoS aware service selection. In *Proceedings of 2011 IEEE International Conference on Service-Oriented Computing and Applications*, (SOCA 2011), pages 1–4, 12-14 December, Irvine, CA, USA, 2011.

[111] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end-to-end QoS constraints. *ACM Trans. Web*, 1(1), 2007.

[112] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web services composition. In *Proceedings of the 12th International Conference on-World Wide Web*, WWW '03, pages 411–421. ACM Press, 2003.

[113] L. Zeng, B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS–Aware Middleware for Web Services Composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, 2004.

[114] L. Zeng, C. Lingenfelder, H. Lei, and H. Chang. Event–driven quality of service prediction. In *Proceedings of the 6th International Conference on Service-Oriented Computing*, (ICSOC '08), pages 147–161, Springer–Verlag, Berlin–Heidelberg, 2008.

[115] H. Zheng, J. Yang, W. Zhao, and A. Bouguettaya. QoS analysis for web service compositions based on probabilistic QoS. In *Proceedings of the 9th international conference on Service-Oriented Computing*, ICSOC 2011, pages 47–61, Berlin–Heidelberg, 2011. Springer–Verlag.

[116] M. Zivković, J. W. Bosman, H. van den Berg, R. van der Mei, H. B. Meeuwissen, and R. Núñez-Queija. Run-time revenue maximization for composite web services with response time commitments. In *Proceedings of IEEE 26th International Conference on Advanced Information Networking and Applications*, (AINA 2012), pages 589–596, March 26-29, 2012, Fukuoka, Japan, 2012.

[117] M. Zivković, J. W. Bosman, H. van den Berg, R. van der Mei, H. B. Meeuwissen, and R. Núñez-Queija. Run-time revenue maximization for composite web services with response time commitments. Submitted, 2013.

[118] M. Zivković, J. W. Bosman, J. L. van den Berg, R. D. van der Mei, H. B. Meeuwissen, and R. Núñez-Queija. Dynamic profit optimization of composite web services with SLAs. In *Proceedings of the IEEE Global Communications Conference*, (GLOBECOM 2011), pages 1–6, 5-9 December 2011, Houston, TX, USA, 2011.

[119] M. Zivković and H. van den Berg. Analysis of revenue improvements with runtime adaptation of service composition based on conditional request retries. In *Proceedings of First European Conference on Service-Oriented and Cloud Computing*, (ESOCC 2012), pages 169–183, September 19-21, 2012, Bertinoro, Italy, 2012.

[120] M. Zivković and H. van den Berg. Revenue optimization of service compositions using conditional request retries. In *Proceedings of the IEEE International Conference on Web Services*, (ICWS 2013), June 27-July 2, 2013, Santa Clara, CA, USA, 2013.

[121] M. Zivković, H. van den Berg, H. Meeuwissen, and B. Gijsen. Performance evaluation of dynamic web service selection. In *2011 Sixth International Conference on Internet and Web Applications and Services*, (ICIW 2011), March 20-25, 2011, St. Maarten, The Netherlands Antilles, 2011.

# Acronyms

| | |
|---|---|
| AC | Admission Control |
| BL | Bernoulli |
| BPEL | Business Process Execution Language |
| CDF | Cumulative Distribution Function |
| CPU | Central Processing Unit |
| CSP | Composite Service Provider |
| DP | Dynamic Programming |
| FCFS | First Come First Served |
| HTTP | Hypertext Transport Protocol |
| i.i.d. | Independent and identically distributed |
| ISO | International Organization for Standardization |
| ISP | Internet Service Provider |
| JSQ | Join the Shortest Queue |
| MCDM | Multiple Criteria Decision Making |
| MMKP | Multi-Choice Knapsack Problem |
| MCOP | Multi-Constrained Optimal Path |
| PDF | Probability Density Function |
| PS | Processor Sharing |
| QoS | Quality of Service |
| REST | Representational State Transfer |
| RR | Round–Robin |
| r.v. | Random variable |
| SLA | Service Level Agreement |
| SLO | Service Level Objective |
| SOC | Service Oriented Computing |
| SOA | Service Oriented Architecture |
| SOAP | Standard Object Access Protocol |
| SJF | Shortest–Job First |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |
| UDDI | Universal Description, Discovery, and Integration |
| W3C | World Wide Web Consortium |
| WS | Web Service |
| WSDL | Web Services Description Language |
| XML | eXtensible Markup Language |
| XSD | XML Schema Definition Language |

# About the author

Miroslav Živković was born in Belgrade, Serbia, on 3rd August, 1969. After completing secondary education at Mathematical Gymnasium in Belgrade, he enrolled at Faculty of Electrical Engineering, University of Belgrade. He obtained his dipl.–ing. title in February 1994, major Electronics and Telecommunications.

He worked, among others, as Member of Technical Staff at Bell Labs Europe, Alcatel–Lucent, the Netherlands and, at Performance of Networks and Systems group, Dutch institute for applied research (TNO), Delft, the Netherlands. He is currently with System and Network Engineering group, Informatics Institute, University of Amsterdam (UvA).

He was awarded the Bell Labs President's Gold Award, for the work on Intelligent Services Gateway in 2004, and the Central Bell Labs Teamwork Award for the work on 3G and 3G services in 2002.

## Scientific publications

This thesis is based on the following publications that have appeared or have been submitted for publication


1. M. Živković, H. van den Berg: Revenue Optimization of Service Compositions using Conditional Request Retries. *International Journal of Web Services Research (IJWSR)*, Vol. 10 no. 2, pp. 1–22, 2013.
2. M. Živković, H. van den Berg: Revenue Optimization of Service Compositions using Conditional Request Retries. In: *Proceedings of 20th IEEE International Conference on Web Services* (ICWS 2013), 2013.
3. M. Živković, J. Bosman, H. van den Berg, R.D. van der Mei: Profit Maximization with Dynamic Service Selection in Service Oriented Architecture. Submitted, 2013.
4. L. Duijvestein, J. W. Bosman, R. van der Mei, H. B. Meeuwissen, M. Živković: Run–time Quality–of–Service Control Framework for Composite Services. Submitted, 2013.
5. D. Worm, M. Živković, H. van den Berg and R. van der Mei: Revenue Maximization with High Quality Assurance for Composite Web Services. In: *Proceedings of IEEE International Conference on Service Oriented Computing & Applications* (SOCA 2012), 2012.
6. M. Živković, H. van den Berg: Analysis of Revenue Improvements with Runtime Adaptation of Service Composition Based on Conditional Request Retries. In: *Proceedings of European Conference on Service-Oriented and Cloud Computing* (ESOCC 2012), 2012.

7. M. Živković, J. Bosman, H. van den Berg, R. van der Mei, H. Meeuwissen, R. Núñez–Queija: Runtime revenue maximization for composite Web services with response–time commitments. In: *Proceedings of 26th IEEE International Conference on Advanced Information Networking and Applications* (AINA 2012), March 2012.
8. M. Živković, J. Bosman, H. van den Berg, R. van der Mei, H. Meeuwissen, R. Núñez–Queija: Dynamic profit optimization for composite web services with SLAs. In: *Proceedings of 54th IEEE Global Communications Conference* (GLOBECOM 2011), 2011.
9. M. Živković, J.L. van den Berg, B.M.M. Gijsen, E. Meeuwissen: Performance evaluation of dynamic web service selection strategies in Service Oriented Architectures. In: *Proceedings of International Conference on Internet and Web Applications and Services* (ICIW 2011), 2011.
10. P. J. Meulenhoff, D. Ostendorf, M. Živković, H. Meeuwissen, and B. Gijsen: Intelligent Overload Control for Composite Web Services. In: *Proceedings of 7th International Conference on Service Oriented Computing* (ICSOC 2009), pp. 34–49, 2009.

Other scientific publications of the author are

11. P. Whiting, G. Kramer, C. Nuzman, A. Ashikhmin, A. van Wijngaarden, M. Živković: Analysis of Inverse Crosstalk Channel Estimation Using SNR Feedback, *IEEE Trans. Signal Processing*, Vol. 59, No. 3, Mar. 2011, pp. 1102–1115.
12. M. Živković, G. Kramer, C. Nuzman, C. Posthuma, J. Wheeler, P. Whiting, and A. J. van Wijngaarden: Performance of digital subscriber line spectrum optimization algorithms, *Bell Labs Techn. J.*, Vol. 13, No. 1, pp. 129–146, Spring 2008.
13. P. Whiting, A. Ashikhmin, S. Borst, J. Jennen, G. Kramer, A. J. van Wijngaarden, and M. Živković: Performance results for digital subscriber line precoders, *Bell Labs Techn. J.*, Vol. 13, No. 1, pp. 147–161, Spring 2008.
14. C. Story, M. Živković, J. Verlinden, A. J. van Wijngaarden: Aspects of Dynamic Spectrum Management Level 3, *Bell Labs Techn. J.*, Vol. 13, No. 1, pp. 117–128, Spring 2008.
15. M. Živković, M. Buddhikot, K. Lagerberg, J. van Bemmel: Authentication across heterogeneous networks, *Bell Labs Techn. J.* Vol. 10, No. 2, pp. 39–56, Aug 2005.
16. B. Peelen, M.Živković, D.Bijwaard, H.Teunissen: Supporting QoS in Broadband Wireless and Wired Access, *Bell Labs Techn. J.*, Vol. 8, No. 2, pp. 65–81, Aug 2003.
17. L. Brandwacht, E. Meeuwissen, H. van den Berg, M. Živković: Models and Guidelines for Dimensioning Private Clouds. In: *Proceedings of 6th IEEE International Conference on Cloud Computing* (CLOUD 2013).
18. W. Ellens, M. Živković, J. Akkerboom, R. Litjens, H. van den Berg: Performance of Cloud Computing Centers with Multiple Priority Classes. In: *Proceedings of 5th IEEE International Conference on Cloud Computing* (CLOUD 2012).
19. P. Whiting, A. Ashikhmin, G. Kramer, C. J. Nuzman, A. de Lind van Wijngaarden, M. Živković, M. Peeters, M. Guenach, J. Maes, J. Verlinden: DSL Crosstalk Coefficient Acquisition Using SNR Feedback. In: *Proceedings of 51st IEEE Global Communications Conference* (GLOBECOM 2008), pp. 3428–3432, 2008.

20. T. Stevens et. al.: Autoconfiguration and Authentication Mechanisms in Broadband Multiservice and Multi–provider Access Networks, presented at Global IPv6 summit, Barcelona, 6–10 June 2005.
21. M. Živković, E. Meeuwissen, H. Teunissen: On security challenges in next generation mobile and converged networks. Wireless World Research Forum, Paris, 6–9 Dec 2005.
22. H. Chen, M. Živković, D. Plas: Transparent End–User Authentication Across Heterogeneous Wireless Networks. In: *Proceedings of 58th IEEE Vehicular Technology Conference* (VTC 2003–Fall), 2003.
23. M. Cvetković, A. Kojović, J. Novaković, M. Živković, D. Mitraković: Problem book in Fundamentals of Electrical Sciences, Chapter 2 (in Serbian), Faculty of Technology, University of Belgrade, Serbia, 2001. ISBN: 86–7401–149–7.

## Patent publications

The list of granted patents (co–inventor):

1. Controlling levels of traffic in a telecommunications network, and a network node therefor, European patent No. 1,278,340
2. Method for secure authentication of mobile devices, US patent No. 8,041,339.
3. Determining channel crosstalk matrices by correlated transmissions to different channels, US patent No. 7,830,978.
4. Determining a channel matrix by measuring interference, US patent No. 7,843,990.
5. Methods and apparatus for providing synchronization in a multi–channel communication system, US patent No. 7,860,044.
6. Method and apparatus for a self–tuning precoder, US patent No. 8,054,925.
7. Interpolation method and apparatus for increasing the efficiency of crosstalk estimation, US patent No. 8,300,726.

The list of pending patent applications (co–inventor):

8. Method to send kill–pill messages.
9. A method for triggering retraining of the digital subscriber lines' modems.
10. Channel estimation and tracking for digital subscriber line.
11. Transfer of a service session with a mobile from a first wireless local area network to one of its neighbours.