ISBN 978-82-326-1418-9 (printed version)
ISBN 978-82-326-1419-6 (electronic version)
ISSN 1503-8181

NTNU
Norwegian University of
Science and Technology

Anders Nordby Gullhav

Doctoral Thesis

Doctoral theses at NTNU, 2016:39

NTNU
Norwegian University of
Science and Technology
Faculty of Social Sciences and Technology Management
Department of Industrial Economics and Technology
Management

NTNU

Doctoral theses at NTNU, 2016:39

Anders Nordby Gullhav

Optimization-based Resource Allocation in Cloud Computing

NTNU
Norwegian University of
Science and Technology

Anders Nordby Gullhav

Optimization-based Resource Allocation in Cloud Computing

Thesis for the degree of Philosophiae Doctor

Trondheim, March 2016

Norwegian University of Science and Technology
Faculty of Social Sciences and Technology Management
Department of Industrial Economics and Technology Management

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Social Sciences and Technology Management
Department of Industrial Economics and Technology Management

ISBN 978-82-326-1418-9 (printed version)
ISBN 978-82-326-1419-6 (electronic version)
ISSN 1503-8181

Doctoral theses at NTNU, 2016:39

MILJØMERKET 241 TRYKKSAK 731

Printed by Skipnes Kommunikasjon as

Summary

This thesis considers the resource allocation problem of a cloud service provider (SP), which provides at set of services delivered through the software-as-a-service model. The SP owns and operates a private cloud, but in periods of high demand, the SP also relies on infrastructure resources provided by a public cloud in the service provisioning. Even though the market for cloud computing services has been growing and is expected to grow further in the future, low quality of service (QoS) is seen as an important issue to be resolved by the cloud computing industry.

The focused resource allocation problem translates to the problem of allocating appropriate resources to the services of the SP in a cost-efficient manner, and so that the QoS is in accordance with the requirements specified in the service level agreements (SLAs) between the provider and the users. This problem is represented as an optimization problem. However, analytic and simulation-based models are used to describe the relationship between a given allocation or resources to a service and the resulting QoS.

This thesis consists of five research papers, and in short, these papers try to answer the following two interrelated questions:

1. Given a certain resource allocation to a service, does this service satisfy the QoS guarantees of the SLA?

2. How can the set of services offered by the SP be deployed in a cost-efficient manner, while ensuring the appropriate QoS?

The two first papers of the thesis concern the former, which is answered by developing both analytic and simulation-based models. Whether the analytic or simulated-based approach should be preferred is dependent on the underlying properties of the services. The second question is considered in the three remaining papers, where optimization models are formulated and solved by both exact and inexact algorithms. Specifically, we provide exact algorithms based on branch and price (B&P) and metaheuristics based on the adaptive large neighborhood search (ALNS) framework. While the B&P approach can provide optimal solutions for small and medium-sized providers, the ALNS approach provides high-quality solutions more quickly.

Acknowledgements

During the last four years, I have done research and duty work, including teaching and supervision, at the Department of Industrial Economics and Technology Management at the Norwegian University of Science and Technology. This thesis is the result of the research done in this period. However, during these years, numerous people have contributed to the work presented herein.

First and foremost, I would like to thank my supervisor, Professor Bjørn Nygreen. Thanks for your valuable comments, your sincere advice and for always having an open door whenever I had concerns to discuss. Moreover, thanks for introducing me to some of the greatest researchers within the area of optimization. I would also like to honor your PhD course in mathematical programming. The details of the simplex algorithm, the basic algorithm of our field, becomes much more evident when you have to code it yourself.

I also owe thanks to my co-supervisors, Professor Poul E. Heegaard and Professor Lars Magnus Hvattum. Thanks to Poul for sharing your insight and knowledge of analytic methods and simulation within the field of telecommunications and computer science. Moreover, thanks to Lars Magnus for overlooking and suggesting improvements to some of my code, and particularly for introducing me to Professor Jean-François Cordeau at the Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport (CIRRELT) in Montréal, Canada. Furthermore, thanks to Jean-François for inviting me as a visitor at CIRRELT, and for all the help with the ALNS.

I'm also indebted to all of my co-authors, Bjørn, Jean-François, Poul, and Lars Magnus, for reading and improving the drafts of the papers. During these years, I have learned a lot about scientific writing.

I also want to thank Telenor's Research and Future Studies Department for the collaboration with fruitful discussions, valuable input and economical support during my first years as a PhD student.

Lastly, I would like to thank my beloved girlfriend, Berit, for being there for me in difficult times, and for cheering me towards the finish line. Thanks to my mother and father for teaching me and giving me the inspiration to be persistent and work hard.

Trondheim, December 2015

Anders N. Gullhav

Contents

Introduction

The market for cloud computing services has grown tremendously over the last decade. After cloud computing was popularized by Amazon's Elastic Compute Cloud in 2006 (Sotomayor et al., 2009), the size of the global public cloud market is expected to reach 97 billion US dollars in 2015 and further grow to 159 billion US dollars in 2020, with the software-as-a-service (SaaS) delivery model being the biggest driver (Columbus, 2015). Compared to previous computing paradigms and technologies, Armbrust et al. (2010) identify three new, somewhat related, aspects that are introduced with the advent of cloud computing. Firstly, cloud computing brings the idea of an infinite pool of resources that is available on-demand, and secondly, it eliminates the need for the up-front commitment of hardware resources by service providers. In addition, cloud computing introduces the possibility to pay for the usage of hardware resources on a short-term basis, i.e., by an hourly rate. These three aspects mean that providers of services accessed over the Internet, i.e., SaaS providers (SPs), might provide services without owning any hardware. For SPs that decide to own and operate hardware themselves, these aspects imply that they can start offering new services without investing in hardware until they see the market's response for the new services. Moreover, it also implies that they don't have to scale their hardware to handle the peaks in the service demand.

While the popularity of cloud computing is increasing, Armbrust et al. (2010) rank service availability as the leading obstacle to cloud adoption and growth. Marston et al. (2011) perform a SWOT (strengths, weaknesses, opportunities and threats) analysis of cloud computing, and name low quality of service (QoS) and availability guarantees as one of the prominent weaknesses and important concerns to be dealt with if large organizations should use mission-critical applications provided by the cloud. In addition, they see data security as a critical threat to the adoption of cloud computing in companies. However, security concerns are not discussed in this thesis.

There are several types of decision problems faced by the cloud computing industry where operations research can be used as decisions support. Still, there seems to be no agreed classification of cloud decision problems. Heilig and Voß (2014) differentiate between four classes of problems: cloud migration, service selection, cloud pricing and resource allocation problems; and three main stakeholders: consumers, cloud service providers and cloud infrastructure providers.

In this view, a cloud service provider, which offers services to consumers, might also be a consumer of a cloud infrastructure provider. In short, cloud migration decisions, faced by the consumers, relate to whether the consumer should buy IT resources or lease them from the cloud. Service selection includes decisions concerning the combination of basic services to form a composite service, and must take into account the QoS and price of the basic services. This problem is faced by both consumers and cloud service providers. Furthermore, pricing decisions are a concern for both cloud service providers and cloud infrastructure providers. Lastly, resource allocation concerns the problem of allocating appropriate resources to different service requests and tasks, and is typically represented as an optimization problem. A common target is to optimize the server utilization to reduce the cost or energy consumption, while maintaining a satisfactory QoS. Heilig and Voß (2014) note that the implementation of mechanisms to improve the fault tolerance by appropriate redundancy allocation is an integral part of the provider's resource allocation problem.

In this thesis, the focus is on the resource allocation problem of a cloud service provider (SP) that is offering at set of cloud services, based on the SaaS delivery model, to its users. The SP owns and operates some hardware itself, but relies on resources provided by a cloud infrastructure provider in periods of high demand. The service delivery is legally handled by contracts, termed service level agreements (SLAs), between the provider and the users. The SLAs include specifications and requirements on the QoS (performance, dependability and security) of the provided services that the provider needs to fulfill.

To further narrow the scope of this thesis, we are considering two interrelated problems within the class of resource allocation problems: the assignment of appropriate resources, including redundancy allocation, to a single service such that the requirements of the SLA is satisfied, and the placement of the set of services delivered by the SP on the infrastructure, either provided by the SP itself or leased from a public cloud infrastructure provider. This decision problem can be solved both statically and dynamically. With a demand that varies over time, but that remains stationary in a stochastic sense for sufficiently long periods, it is possible to obtain stationary resource allocations for each demand period. On the other hand, a dynamic approach will seek to find a solution that adapts continuously to the demand.

This thesis is composed of five research papers. The first two papers (Chapters 2 and 3) propose analytic and simulation-based methods to analyze the performance of cloud services under the existence of failures. Since cloud services often run in multiple virtual machines and might have some redundancy incorporated, a failure does not necessary bring the services down, but rather degrades their performance. This makes it valuable to consider failures while analyzing the performance of such services. The three last papers (Chapters 4, 5 and 6) model the problem of placing the virtual machines of the services on the infrastructure as an optimization problem. This optimization problem is solved by both exact and heuristic methods in the papers.

The outline of the rest of the introduction is as follows. Section 1.1 gives a short presentation of the concepts of cloud computing, and thereafter, provides an overview of the resource allocation problem and the literature related to the content of this thesis. Moreover, Section 1.2 delimits the thesis by presenting its purpose and the outline of the papers. Section 1.3 discusses the contributions of the thesis to the research community and industry, and finally, some concluding remarks and directions for future research are given in Section 1.4.

1.1. Background

This section aims at describing the research scope and background of this thesis, by setting the context and the resource allocation problem in focus. We also give a review of some related literature.

1.1.1. Cloud Computing Terminology

Cloud computing is the most recent computing paradigm that promises to deliver computing as the fifth utility, after water, electricity, gas and telephony, over the Internet (Buyya et al., 2009). The term *cloud computing* encompasses both the applications delivered as services over the Internet and the hardware and software systems composing the infrastructure running the applications (Armbrust et al., 2010). The National Institute of Standards and Technology (NIST) defines cloud computing as *a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction* (Mell and Grance, 2011). The service provider is in this context the provider of a service that potentially uses the infrastructure provided by the cloud to provide its service.

The NIST further defines three service (delivery) models for Cloud computing: software-as-a-service (SaaS), platform-as-a-service (PaaS) and infrastructure-as-a-service (IaaS). In the SaaS service model, the cloud user is delivered software applications over the Internet. This eliminates the need for the users to install and run applications on their own computers, facilitating management and maintenance. Examples of this service model are document management systems like Google Docs, customer relationship management systems like Salesforce.com, and webmails like Gmail. In the IaaS service model, the user is provided fundamental computing resources like processing power, storage and network, onto which the user can deploy and run own software, including operating systems and applications. The cloud user in this model is often service providers offering SaaS services to their own users. In the IaaS model, the user does not manage or control the underlying infrastructure, and the service model thereby simplifies the management of the cloud user's service provisioning. Examples of this service model include Amazon Web Services and Rackspace. The PaaS model lies between the SaaS and IaaS service models, as it provides the cloud-user not only computing resources but also software servers and application environments, which can assist the cloud-user to deploy own applications. Examples of PaaS are Google AppEngine and Force.com.

Furthermore, the NIST distinguishes between four different types of cloud deployment models: private clouds, community clouds, public clouds and hybrid clouds. Private clouds are clouds where the cloud infrastructure is operated solely for a single organization, by the organization itself or a third party. In a community cloud, the infrastructure is shared by several organizations. Organizations that in common employ this cloud model often have a shared mission or shared requirements related to security. On the other hand, public clouds are clouds where the infrastructure is available to the general public and is owned by an organization selling cloud services, denoted a (public) cloud provider. Lastly, hybrid clouds are a composition of two or more clouds, which are bound together by some sort of technology that enables data and application portability.

Virtualization is one of the key enablers of cloud computing, as virtual machines (VMs) give the possibility to run several virtual servers on a physical server, and thereby letting several software applications utilize the same hardware (Wang et al., 2008). In this respect, the number of physical servers, also referred to as nodes throughout this thesis, can be reduced, and the economies of scale become more prominent. Besides, VMs isolate the software running on a VM from other VMs running on the same node and the node itself. This is advantageous, in that if a VM on a node fails because of a fault in its running software, the fault is not affecting other VM instances on that node (Rosenblum, 2004).

1.1.2. Resource Allocation in Cloud Computing

In the literature, there exist descriptions and specifications of different variants of the resource allocation problem. While some works focus on the dynamic problem of continuously adapting the resource allocation according to an assumed non-stationary demand, other works focuses on a static problem variant. For some services, the demand can be assumed to be constantly fluctuating, and hence, solution approaches that consider the dynamic problem seem appropriate. However, other services have stationary demand periods, such as working hours, evenings and weekends, which might justify a static approach.

A lot of research on resource allocation takes the perspective of an IaaS provider, which is natural since the infrastructure operated by an IaaS provider is often enormous with great possibilities for resource optimization. Nevertheless, resource optimization is also relevant for SPs that own some hardware themselves. The SaaS applications provided by the SP can be represented as multi-tier services, composed of several components, and a much cited example of a multi-tier service, is the three-tier web service composed of a web server component, an application logic component and a database server component. Figure 1.1 illustrates the structure of a three-tier web service. In this context, the resource allocation problem can be modeled as an optimization problem that seeks to find the most cost-efficient allocation of resource to the components, while ensuring a QoS in correspondence with the SLA. An SLA is a contract between a service provider and a user, which specifies the services delivered to the user, the metrics used to quantify the QoS, the bounds on the QoS metrics, the amount of traffic the user can transmit, and the penalties that incur if the provider fails to provide a service within the bounds (Wu and Buyya, 2012). The QoS metrics in an SLA for a cloud service might relate to the performance, e.g., the response time, and the dependability, e.g., the uptime or availability, but also consider the security. In the papers of this thesis, we consider the response time as the total time required by the multi-tier service to process the request of a user. We refer to Wu and Buyya (2012) for an overview and discussion of SLAs in the context of cloud computing.

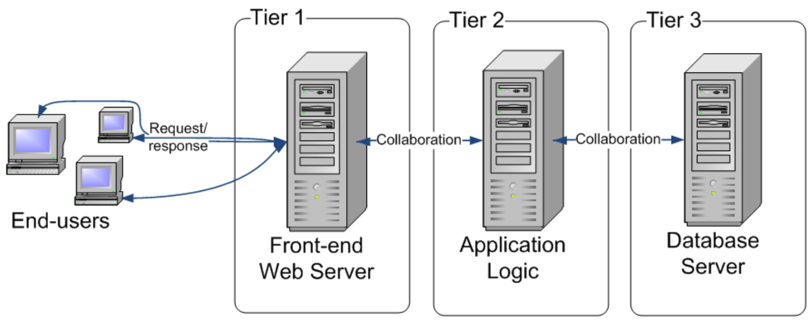

Figure 1.1.: Illustration of the components of three-tier web service

The allocation of resources to services is done by running the components in VMs, one component per VM, which in turn are run on the (physical) nodes in a private or public cloud. Each VM has a given resource specification, for instance, 1 CPU core, 2 Gigabytes (GB) of memory and 100 GB of storage, and all VMs on a node share the node's limited resources. The resource scaling of the service components can be done in two principal ways, either by horizontal or vertical scaling. Horizontal scaling refers to the process of changing the resource allocation by decreasing or increasing the number of VMs of a predefined size, while vertical scaling refers to the process of decreasing or increasing the size of the already deployed VMs. Horizontal scaling results in a more coarse-grained resource scaling compared to vertical scaling, and might results in sub-optimal resource allocation (Sedaghat et al., 2013). Nevertheless, the typical scaling mechanism implemented in the cloud management systems today is the horizontal scaling.

Simulation models and analytic queuing models are the typical tools used to quantify the performance of multi-tier services in the literature (Ardagna et al., 2014). If the basic assumptions of the stochastic behavior of the services are "easy", e.g., the inter-arrival and service times are exponentially distributed, analytic queuing models can be used to describe the performance of the services. Otherwise, simulation might be a viable modeling tool. Section 1.1.3 presents different approaches, described in the literature, to performance modeling of multi-tier services by using simulation and analytic modeling techniques.

When one is to provide a service run on a failure-prone infrastructure, such as the hardware composing private and public clouds, one should consider the effects of the faults on the services. No matter how well-designed a system is there is always a risk of failure if faults are too frequent, and a key to be able to tolerate faults is to introduce redundancy (Gärtner, 1999). An overview of the field of dependability in computing systems, including the dependability metrics, the different types of faults and means to achieve dependability is given by Avižienis et al. (2004).

In the context of cloud computing, there exists some literature that considers fault tolerance management. Undheim et al. (2011) present different ways to implement fault tolerance in cloud services through standby redundancy by replication of VMs. Standby redundancy refers to the principle and techniques of allocating standby resources that can be activated in case of a failure. In this case, the standby resources correspond to VMs. The standby redundancy classification of Undheim et al. (2011) is depicted in Figure 1.2. At the top level, they distinguish between hot and cold standbys, which are powered and non-powered VMs placed on the nodes, respectively. For hot standbys, one distinguishes between different levels of synchronization, which reflect how updated the state of the standby can be expected to be when a failure occurs. For some services, named stateful services, a standby VM requires an updated state to be activated. On the other hand, stateless services don't have this requirement. Typically, a higher level of synchronization requires more usage of resources. The highest level of synchronization is the one with a fully updated and synchronized state, and this technique requires the allocation of dedicated resources to the standby VMs. An example of this hot standby techniques is VMware Fault Tolerance (VMware White Paper, 2009), where also the standby VMs process all requests to ensure a fully updated state. This means that a standby can take over service delivery without a noticeable interruption. An example with a lower level of synchronization is Remus (Cully et al., 2008), where only information about the state is transferred to the standby VMs at specific points in time. In this case, the state of the standby is not fully updated, and the standby needs a short amount of time to process the last update before being activated. In the latter approach, the standby VMs consume fewer resources than the active VMs, and it allows the standbys to share backup resources. Compared to the two reviewed techniques, cold standbys are still loaded on the nodes, but are not updated and, thus, more ideal for stateless services. In addition, they require some time to be powered before they can start serving demand, and hence, the services applying this technique should tolerate a short downtime. The redundancy scheme applied in the models of this thesis is a form of standby replication with hot, updated standby replicas with shared backup resources, like Remus. Other works that present conceptual frameworks and software systems utilizing replicated VMs include Distler et al. (2011), where a standby VM can be put into a paused state on a node, from where they can be unpaused and activated rapidly.

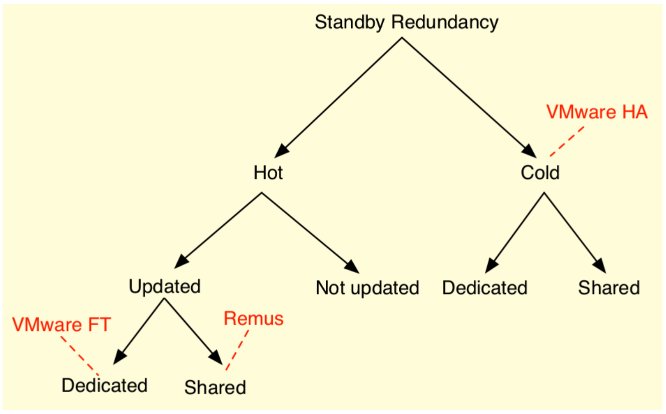

Figure 1.2.: Standby redundancy classification (Undheim et al., 2011)

In the context of implementing fault tolerance by redundancy, a related resource allocation problem is the redundancy allocation problem (Kuo and Wan, 2007). This problem consists of allocating parallel components to different subsystems in series with an objective of minimizing the cost, while maintaining a reliability greater than a threshold. This type of problem is found in many contexts, including design of software systems, which is considered in Belli and Jedrzejowicz (1991) and Ashrafi et al. (1994).

The traditional way to allocate standby resources in a non-virtualized computing system is to dedicate M backup nodes to tolerate M failures in the N active nodes (Sun et al., 2001). However, likewise virtualization enables resource savings by consolidating several active VMs on a single node, virtualization enables the opportunity to place standby VMs across the nodes running active VMs (Loveland et al., 2008). In this situation, one should ensure that VMs of the same software component never run on the same node.

There seems to be little research done on a combined analysis of the performance and dependability of cloud services. Such a combined analysis is sometimes denoted performability analysis (Trivedi et al., 1993; Al-Kuwaiti et al., 2009). Performability is a concept that is applicable for *degradable systems*, which have the property that during a period of time, the system can exhibit different performance levels as a result of faults. For fault-tolerant systems, a fault does not necessary lead to a failed system, but to a system with degraded performance. For SaaS applications that are composed of replicated tiers, the performability concept is of interest since a fault in one of the VMs of a tier, will not necessarily bring the service down, but rather reduce the service performance.

In addition to the allocation of server resources, such as CPU power and memory, and allocation of redundancy, the underlying network is also affecting the performance and dependability of cloud services. The network latency between data centers is typically significant, and the access network of the users is also affecting the user's perceived QoS. Moreover, the network topology is an important factor of the dependability of cloud services (Jennings and Stadler, 2015). However, the effects of the underlying network are not modeled and discussed in the papers of this thesis.

We have already reviewed some related literature on concepts related to resource allocation in cloud computing, including research on redundancy allocation problems. In the two next sections, we will firstly review research on performance and dependability modeling, and then some of the literature on placement problems in clouds.

1.1.3. Related Work on Performance and Dependability Modeling of Cloud Services

Ardagna et al. (2014) present a survey on workload modeling, system modeling, and their applications to resource allocation in the area of QoS management in cloud computing. While workload modeling, e.g., demand predication and measurement-based techniques, is out of the scope of this thesis, much of the reviewed literature on system modeling is relevant here.

Analytic models of SaaS and web service performance are often based on queuing theory. In the following review of related performance modeling literature, the focus is on the modeling of multi-tier services, which typically is based on some form of queuing network. In this relation, Ramesh and Perros (2001) model a client-server system as a multi-layered queuing network, where the servers are organized in a tiered structure. The network is analyzed by an approximate algorithm to obtain the mean response time of the requests, and the results are validated by a simulation model. Urgaonkar et al. (2005) design a closed queuing network to analyze a multi-tier Internet service, where each of the tiers might be replicated into a number of parallel load-balanced servers. They derive the mean response time of the requests by mean-value analysis. Another queuing model is given in Xiong and Perros (2009). Therein, a cloud service is modeled as an open queuing network composed of two M/M/1 queues in series with feedback from the second to the first queue and exits after each queue. The focused performance metric is not the mean response time, but rather the high percentiles, e.g., the 90th and 95th, of the response time distribution. They develop an approximate method for deriving the Laplace-Stieltjes transform of the response time distribution in the queuing network model, and numerically compare the results of their approximation with the results of a simulation model. Moreover, Singh et al. (2011) propose a tool to analyze the average response time of complex applications spanning hundreds of nodes in a data center, by modeling the application as a queuing network. There also exist approaches that use layered queuing networks (Franks et al., 2009) to model multi-tier services. Jung et al. (2008) extract the mean response time of such network models, while more recently, Perez and Casale (2013) compute an approximate response time distribution to obtain values for the higher percentiles.

There exists several approaches to obtain response time distributions of queuing networks, and a review, now fairly dated, is given by Boxma and Daduna (1990). In cases where the network is not overtake-free, the determination of the response time distribution is generally hard, and the usual approach in such cases is to obtain an approximate distribution. A type of approximation is based on decomposing the network and studying the individual queues separately (Woolet, 1993). Woolet's approximation method for M/M/c/b queues is extended to phase-type service times in Grottke et al. (2011).

Other related literature is works that study queues or queuing networks with failing servers. Queuing networks with server breakdowns and repairs are considered in Chakka and Mitrani (1996) and Sauer and Daduna (2003). However, these works do not try to obtain the full response time distribution of the networks.

Pure dependability models for cloud services, without performance considerations, are studied in Undheim et al. (2011) and Dantas et al. (2012). Undheim et al. (2011) present a cloud availability model that accounts for different types of failures in data centers, such as failures with the power supply, network failures and server failures. They use reliability block diagrams to model power supply and network failures, and model the service failures for a fault-tolerant single tier service by using continuous time Markov chains (CTMCs) (Trivedi, 2002). They suggest development of failure models for multi-tier services as future work. Dantas et al. (2012) present an availability model for the Eucalyptus platform (Hewlett Packard Enterprise, 2015) for building private, public and hybrid clouds, now acquired by HP. They only consider and model the inclusion of redundancy in the cloud architecture, and do not model the services running in the cloud.

There exist a few works on a combined performance and dependability analysis in a cloud computing context. Ghosh et al. (2010) study the performability of the service provisioning process of an IaaS provider, that is, the process of providing VM instances on request. They analyze the process by modeling the performance and dependability of the provisioning as interacting CTMCs. The focused QoS metrics are the effective probability of not provisioning a VM and the effective delay from a VM instance is requested to it is ready for service provisioning, under the existence of failures. Another use case for performability analysis in cloud computing systems is given by Qian et al. (2011). Therein, they perform a combined performance and dependability analysis of the provisioning of online services, such as voice or IP, streaming and search, through an IaaS provider. They consider replication of the service VMs, and construct a cloud performance (response time) and dependability model, in addition to a network latency model. They argue that for the users, the interesting QoS metric is the probability of observing a latency less than a specified threshold.

In cases where the properties of the system make analytic modeling difficult or inaccurate, simulation is often a viable tool. Alam et al. (2012) present a discrete event simulation tool designed to support services providers in the performance management of their services. A more extensive tool named CloudSim (Calheiros et al., 2011) supports modeling and simulation of other cloud elements, such as data centers, virtual machines and networks, in addition to the services. There also exist several extensions to CloudSim. An example is CloudAnalyst (Wickremasinghe et al., 2010), which allows one to model and evaluate service behavior in situations with geographically distributed demand. However, to our knowledge, there does not exist adequate simulation models or tools that allow one to analyze cloud service performance under the existence of failures.

There also exist approaches in the literature that include analytic queuing theory-based relations within optimization models to model the behavior of multi-tier services e.g., Goudarzi and Pedram (2011) and Ardagna et al. (2012). These approaches are reviewed in Section 1.1.4.

1.1.4. Related Work on Placement Problems in Clouds

In the literature on placement problems, most of the research focuses on the placement or scheduling problem of an IaaS provider. However, since the SP considered in this thesis also owns and operates a private infrastructure, the research focusing on IaaS providers is relevant herein. Moreover, in this limited literature survey, we do not review scheduling problems for batch type workloads, e.g., scientific computations and analysis and other workloads with infrequent requests, each with a relative high resource demand, but rather problems with web service-like workload patterns.

Many models and algorithms referred in the literature consider the placement of a set of independent VMs on a common infrastructure, and model this VM placement problem as a kind of bin packing problem. Hermenier et al. (2009) present a constraint programming model that dynamically assign VMs to nodes in clusters. The aim of this model is to minimize the number of (physical) nodes used, in order to reduce the energy consumption, while ensuring that the VMs are allocated enough resources, in this case, CPU power and memory. Furthermore, the approach also provides reconfiguration plans for migration of the VMs as a response to demand changes. A related approach is provided by Ferreto et al. (2011), in which another bin packing-like model is studied and solved using heuristic methods.

In addition to minimizing the number of nodes used and putting the unused nodes in a power-saving state, another technique to reduce the energy usage of data centers is a strategy named dynamic voltage and frequency scaling (DVFS). DVFS is a power optimization technique where the frequency and voltage of the CPU is varied according to the demand. Kramer et al. (2012) consider this technique when modeling their static VM placement problem as a one-dimensional variable-sized bin-packing problem, where the node capacities are variable and dependent on the CPU frequency. They decompose the problem, and design a column generation algorithm and column generation-based heuristics to solve the problem. Static and dynamic optimization models that include frequency scaling decisions are given by Petrucci et al. (2010). They start by modeling the problem as an one-dimensional variable-sized bin-packing problem, and extend their models by introducing dynamicity by modeling costs for switching nodes on and off between two time periods, and costs for migrating VMs from a node to another. An additional variant of the bin-packing problem in a VM placement context is studied in Cambazard et al. (2013), where the cost of using a bin is dependent on its utilization.

Compared to the previously reviewed problems, placement problems that include structural relations between the VMs share more features with the problem studied in this thesis. Such structural relations might be constraints specifying that a set of VMs should run on different nodes, run on the same nodes, be placed in different geographical regions, or run on a specific subset of the nodes. Google proposed a dynamic placement problem of an IaaS provider in the ROADEF/EURO challenge 2012 (ROADEF, 2012). In this problem, named the machine reassignment problem, a set of processes, i.e., VMs, is to be assigned to a set of machines with the goal to improve the machine usage. The problem includes constraints ensuring a geographical distribution of the identical VMs and constraints related to the migration of VMs from a machine to another. For this problem, the winning approach was presented by Gavranović and Buljubašić (2014), which solution method is based on a local search heuristic with multiple starts and a noising strategy to escape local optima.

A static VM placement problem with multiple time periods with different demand is considered in Speitkamp and Bichler (2010). Moreover, they present model extensions, which include constraints requiring that a set of VMs should be placed on different nodes, and they solve their MIP models by using both a general-purpose solver and an LP-relaxation-based heuristic. Another static placement problem is regarded in Goudarzi and Pedram (2012), where VMs can be split in a variable number of smaller VMs that fit better on the nodes in the data center. The authors argue that this might increase the utilization and reduce the energy consumption of the data center. Breitgand and Epstein (2011) concern a placement problem of an IaaS provider in which multiple VM copies of a service should be placed on different nodes. Due to computational difficulties, they reformulate the problem and use column generation to solve the reformulated model. They consider both a static version of the problem and an extension that is based on an initial placement.

None of the reviewed placement literature above, considers that VMs might be part of multi-tier services. However, such regards are modeled by Goudarzi and Pedram (2011) and Ardagna et al. (2012). Goudarzi and Pedram (2011) include performance requirements specified in the SLAs which the provider has to comply with. The multi-tier services are modeled as queuing networks and the considered performance metric is the average response time of the requests. The authors present a non-linear model for optimizing the static placement and service scaling in a stationary time period. To solve the model, they propose a heuristic based on local search. Ardagna et al. (2012) also consider a resource allocation problem with QoS constraints, for which they propose a solution framework that includes service scaling, VM placement and means to reduce the energy consumption. Their framework iterates between solving an overall placement model and a capacity allocation model at each server. Both models contain non-linearities due to the relations of the performance model that is based on queuing networks, and the models are solved by local search-based heuristics.

An earlier model of a placement problem considering multi-tier services is given in Urgaonkar et al. (2007). In this problem, an infrastructure provider seeks to maximize the number of multi-tier services deployed on a set of nodes, and deploying a service implies that all of its tiers need to be placed on the nodes. The authors propose solution methods for both a static variant of the problem and a dynamic variant, where services is placed one by one as they are requested deployed on the infrastructure. The solution method for the static problem variant is based on the classical first fit heuristic, while the aim of the dynamic variant is to find a feasible deployment, or show that one does not exist. Another approach that combines a queuing model with an optimization model is given by Jung et al. (2008). The optimization model is a bin packing-like model with the objective to maximize a utility function, which is based on the performance of the multi-tier services running in the data center of an infrastructure provider. The proposed algorithm selects the replication level at each tier, and places the resulting components by the first fit heuristic. In the search for the best replication levels, according to the utility function, the algorithm starts with the maximum replication levels (i.e., maximum performance and utility) and gradually reduce them until the algorithm stops when finding a feasible packing of the components.

There are also works that consider placement in multiple clouds. Csorba et al. (2010) propose a solution method based on ant colony optimization to place replicated VMs in a hybrid cloud environment. They impose requirements specifying that replicas of the same components should run on different nodes, and possibly in different node clusters. Tordsson et al. (2012) model a decision problem of a cloud broker, which is an entity optimizing the placement of VMs among multiple IaaS providers on behalf on an SP. They model the problem statically as a binary IP, and maximize the amount of resources assigned to the SP's VMs under a given budget. The model is solved by a general-purpose MIP solver.

As discussed in Section 1.1.2, a studied method to achieve fault tolerance is to dedicate a number of backup nodes to tolerate failures. However, a more resource efficient fault tolerance strategy is to place a number of redundant VMs on different nodes (Machida et al., 2010). Machida et al. (2010) compute the number of replicated VMs of an application by a performance model based on analyzing an M/M/1 queue. To achieve fault tolerance, a number of extra, redundant VMs are deployed, one per failure that should be tolerated. All VMs placed on the infrastructure have identical size, so the resulting static placement problem is easily solved by a greedy algorithm. Bin et al. (2011) consider a VM placement problem of an IaaS provider where each VM should be reserved a number backup locations on other nodes, where this number is equal to the number of node failures that should be tolerated. The motivation of the backup locations is to ensure that a VM can be relocated on a new node in case of a number of node failures. The authors propose to solve the problem using constraint programming.

1.2. Purpose and Outline

The purpose of this thesis is discussed in Section 1.2.1. Afterwards, we outline and present the purpose and contributions of each of the five included papers in Sections 1.2.2 to 1.2.6.

1.2.1. Purpose of Thesis

The purpose of this thesis is to present new optimization models and algorithms that should support SaaS providers (SPs) in their service provision. In addition, the presented optimization models are supported by analytic and simulation-based models used to evaluate the performance and dependability of multi-tier SaaS services.

The different components, i.e., tiers, of SaaS services are typically run in a load-balanced configuration with many VMs serving the requests in parallel. If a failure brings down a VM in such a setting, the service might still be able to operate, however, at a lower performance level. To increase the fault tolerance of the service, we argue for the placement of additional standby VMs that are run in a passive state, but activated when a failure occurs. We denote these two types of VM replicas as *active* and *passive* replicas. However in general, modeling of performance relations, e.g., based on queuing models, explicitly in optimization models introduces non-linearities. To formulate the resource allocation problem with linear optimization models, we introduce a modeling structure referred to as *replication patterns* in the optimization models. A replication pattern specifies the number of active and passive replicas allocated to each tier of a single service, i.e., the resource and redundancy allocation of the service, and it is assumed to indicate the QoS of the service. By this, we also assume that the placement of the VMs, and the underlying network, does not affect the service performance, which is a simplification of the reality.

Since the services run on the same infrastructure, it is not trivial to decide which out of possibly several different replication patterns with satisfactory QoS that is the most cost or resource-efficient. Therefore, the decision variables of the optimization models include the selection of replication patterns in addition to the placement.

The introduction of replication patterns can be seen as a way to decompose the problem into two: the problem of finding replication patterns for each service, and the problem of optimizing the selection of replication patterns and the placement of the replicated VMs. The problem of finding replication patterns is not discussed explicitly in the papers, but can potentially be done by enumeration. However, we provide methods to check whether a replication pattern satisfy the QoS guarantees.

This thesis consists of five papers. The two first papers focus on analytic and simulation-based models, while the three last papers focus on optimization models and algorithms. In short, the papers provide decision support to answer the following two related questions:

1. Given a certain allocation of resources and redundancy to a multi-tier service, does this service satisfy the QoS requirements guaranteed in the SLA?

2. How can the set of services offered by the SP be deployed in a cost-efficient manner, while ensuring the guaranteed QoS through selection of replication patterns?

Papers I and II study the problem of assessing the response time distribution of a fault-tolerant multi-tier service. The papers assume slightly different dependability models (also termed failure models), but differs fundamentally in the tools they use to obtain the response time distribution. In Paper I, we make the assumption of exponentially distributed inter-arrival and services times of the service requests, and obtain an approximate response time distribution by developing an analytic model. In Paper II, we compare the approximations of Paper I with estimated response time distributions obtained by simulation, and see that there is a close match between the two. In addition, we show the applicability of the simulation approach for non-exponential service times.

Papers III, IV and V consider the combined optimization problem of selecting replication patterns and placing a set of services in either a private or hybrid cloud environment. This problem is referred to as the *service deployment problem*, and is studied in a context where there exist sufficiently long stationary demand periods, for which it is appropriate to find and implement a static deployment of the services. In Paper III, the problem is formalized and modeled as MIPs. These models illustrate the hardness of the problem. In the subsequent papers, we propose both exact and heuristic solution methods. The exact approach, proposed in Paper IV, is based on the decomposition of the problem into a master problem and a subproblem that is solved by branch and price (B&P). Paper V presents adaptive large neighborhood search (ALNS) heuristics that utilize local search (LS) operators on top of the standard ALNS framework.

1.2.2. Paper I: Approximating the Response Time Distribution of Fault-tolerant Multi-tier Cloud Services

In this paper, we propose a method for approximating the response time distribution of a multi-tier service, which also accounts for failures. The tiers of the service are realized by several VMs that share the load of the users. The VMs of the service fail according to an exponential failure time distribution, however, we assume that the VMs fail independently. When failed, the time to repair a VM is exponentially distributed according to a repair time distribution. In case of a failure, unless all VMs of tier has failed or a tier becomes completely saturated, e.g., due to higher demand than service capacity, the users will not perceive the service as unavailable, but the response time might be increased. Moreover, to improve the fault tolerance of the service, we allow the SP to deploy additional passive VMs, i.e., passive replicas. A way to attain the QoS of such fault-tolerant services is by a combined performance and dependability analysis.

Without regarding the passive replicas, a multi-tier service can be modeled as a queuing network composed of parallel queues in series. In this paper, we assume exponentially distributed inter-arrival and service times. Obtaining the response time distribution, sometimes also denoted sojourn time distribution, of a queuing network is in general difficult (Boxma and Daduna, 1990). Another difficulty of the problem is the inclusion of failures in the queuing model, as this would lead to a large number of states in the underlying CTMC. In addition, since failure rates are in the order of hours, while arrival rates and service times are in the range of tens of milliseconds to seconds, solving a monolithic CTMC would lead to numerical problems (Trivedi et al., 1993). However, because of the two very distinct time scales, it is possible to decompose the total problem into a dependability submodel and a performance submodel.

The main contribution of this paper is a method for assessing the combined performance and dependability of a fault-tolerant multi-tier service, which returns an approximated response time distribution. The dependability submodel, denoted the failure model in the paper, is based on a Markov reward model assuming exponential failure and repair times. The performance submodel is based on the response time block method of Grottke et al. (2011).

The paper is co-authored with Professor Bjørn Nygreen and Professor Poul E. Heegaard, and published in the proceedings of the *IEEE/ACM 6th International Conference on Utility and Cloud Computing*, 2013, pp. 287-291.

1.2.3. Paper II: Simulation of the Response Time Distribution of Fault-tolerant Multi-tier Cloud Services

This paper considers the same problem as in Paper I, but the proposed approach offers greater flexibility in the underlying assumptions about exponentially distributed inter-arrival and service times. The methodological distinction between the two papers lies in the performance model, which in this paper is designed as a discrete event simulation model. However, the issue of the two very dissimilar time scales is still present in the problem, and we discuss the difficulties of simulating a complete model that includes both the failure/repair processes and the arrival/service processes. Due to these difficulties, we utilize stratified sampling (Lewis and Orav, 1989) to simulate the response times by decomposing the problem into separate dependability and performance submodels. The strata of the simulation correspond to failure scenarios, which are analyzed and quantified by the dependability model.

The main contribution is a framework for obtaining an estimated response time distribution of a fault-tolerant multi-tier service by simulation. In addition, the paper demonstrate that the approximation method presented in Paper I performs well for exponential inter-arrival and service time distributions, but that for other distributions, the simulation model is valuable.

The paper is co-authored with Professor Bjørn Nygreen and Professor Poul E. Heegaard, and is submitted to an international journal.

1.2.4. Paper III: Deployment of Replicated Multi-tier Services in Cloud Data Centres

This paper considers an optimization problem of an SP that seeks to find the least cost deployment of a set of fault-tolerant multi-tier service on an infrastructure consisting of a private and public cloud. The SaaS provider owns and operates a private cloud that is composed of several servers, denoted nodes. In some periods, the private cloud is large enough to run all of the services provided by the SP, but in other periods, the SP has to utilize the public cloud offerings of IaaS providers in his service provisioning. In this problem, the motivation of using the public cloud is limited to be that the private cloud is fully utilized, and thus, we don't include explicit requirements that call for usage of multiple clouds, e.g., requirements for geographical distribution of replicas.

The main contribution of this paper is linear MIP models of the problem, both for the case where all services are run in the private cloud and for the case where using a public cloud is necessary. The MIP models include decisions both regarding the replication levels of the services and the placement of the VMs of the service, which is modeled using binary variables. The cost of placement in the private cloud is modeled to reduce the energy usage in times where the demand is low, and counts the number of nodes turned on. On the other hand, when the demand is high and the private cloud is fully utilized, the concerned cost component is the cost of running VMs in the public cloud. The placement of replicas is modeled by including technical constraints such as the resource capacity constraints of the nodes and constraints ensuring that the replicas of the same tier is placed on different nodes. In addition to placement of load-balanced active replicas, the model allows placement of passive replicas, which are assigned fewer resources in a failure-free situation. However, each node running at least one passive replica, will maintain a pool of shared backup resources, so that the passive replicas are able to be activated. A set of replication patterns for each service is given to the model as input, and the model forces the selection of one replication pattern for each service. All included replication patterns are by the model assumed to give a satisfactory QoS, as specified in the SLAs.

Our models contain some novelties in comparison to works that model placement of multi-tier services in the literature. While Goudarzi and Pedram (2011) and Ardagna et al. (2012) propose placement and resource allocation models that include explicit response time requirements, they don't consider placement of redundant, passive VMs and the allocation of shared backup resources. In addition, although Csorba et al. (2010) model placement of replicated VMs in private and public clouds, they don't include decisions related to performance or dependability in their models.

The paper shows that to solve a direct MIP model of the problem with a general-purpose MIP solver is very time-consuming and unpractical for problems of reasonable size. Another contribution of the paper is a reformulated model, which is based on *node patterns*. A node pattern corresponds to a set of replicas that can be placed on the same node while respecting the node-specific placement constraints. The reformulation couples the selection of replication patterns, which decides the number of replicas to place, with the selection of node patterns, which decides where to place the replicas. However, the number of feasible node patterns grow exponentially with the problem size, and including all feasible patterns in the model is feasible only for very small cases. Nevertheless, we compare the efficiency of solving the direct MIP formulation and solving the reformulation, by optimizing over a small subset of the node patterns. The results show that the reformulation performs better.

This paper is co-authored with Professor Bjørn Nygreen, and published in *International Journal of Cloud Computing*, Vol. 4, No. 2, 2015, pp. 130 - 149.

1.2.5. Paper IV: A Branch and Price Approach for Deployment of Multi-tier Software Services in Clouds

This paper consider the same problem as in Paper III, but instead of generating the node patterns in advance of calling the solver on the reformulation, we propose to generate the node patterns dynamically, in a column generation fashion. The column generation is implemented in a branch and bound (B&B) framework, to form a B&P. The master problem corresponds to the reformulation of Paper III, while the subproblem has similarities with a knapsack problem. The subproblem is formulated both as a MIP and as a shortest path problem with resource constraint (SPPRC). The underlying network of the SPPRC utilizes the properties of the problem, and is to our knowledge novel. The subproblem formulations are respectively solved by both a MIP solver and a label-setting algorithm (LSA). The branching on the node pattern variables is done by a problem specific branching rule, which imposes new constraints and variables in the subproblem MIP, and has implications for the LSA.

A contribution of the paper is a heuristic label-setting algorithm, which based on a reduced network and simplified dominance rule, produces new node patterns quicker than the MIP solver. To maintain an exact and complete solution method, the heuristic LSA is complemented with the exact MIP solver. However, in some nodes of the enumeration tree, no improving node patterns can be found, and hence, using the heuristic algorithm is inefficient. Another contribution of this paper is a simple rule to decide whether the heuristic algorithm should be used in a node, or if the exact MIP solver should be called directly. The computational study of the paper shows the benefits of using a heuristic subproblem solver in addition to the exact MIP, and also demonstrates that the B&P approach outperforms the solution approach used in Paper III.

This paper is co-authored with Professor Bjørn Nygreen, and is submitted to an international journal.

1.2.6. Paper V: Adaptive Large Neighborhood Search Heuristics for Multi-tier Service Deployment Problems in Clouds

The purpose of this paper is to present a heuristic solution approach that produces solutions quicker than the B&P of Paper IV. While we in Paper III and IV emphasize that we are considering a static problem with sufficiently long and stationary demand periods, there might also be situations where a new solution is needed more quickly. In this paper, we propose two adaptive large neighborhood search (ALNS) heuristics, one for the case where only a private cloud is used for deployment and one for the hybrid cloud setting. The neighborhood operators of the two algorithms differ to some degree.

The main contribution of the paper is an adaption of the standard ALNS framework to include LS operators on top of the repair operators. That is, after a solution is destroyed and subsequently repaired, an LS operator is called to find the local minimum with respect to the operator. A set of swap operators are included, and in each iteration, one operator is selected and used. The selection is based on the standard adaptive operator selection principle applied for the destroy and repair operators in the literature. Moreover, another feature of the proposed heuristics is the usage of a general-purpose MIP solver in one of the repair operators. This operator solves a reduced version of the direct MIP model presented in Paper III, where parts of the variables are fixed. The fixed variables correspond to the non-destroyed part of the solutions. Since the MIP-based repair operator is used side by side with faster, but simpler insertion-based repair operators, we modify the scoring mechanism of the repair operators to consider the time consumption.

The numerical results presented in the paper show that the inclusion of the LS operators is beneficial. Furthermore, the results show that the ALNS performs better than the B&P on the larger test cases, also with a longer run time. However, on small and medium-sized cases in the hybrid cloud setting, the B&P is clearly better than the ALNS when given enough run time.

This paper is co-authored with Professor Jean-François Cordeau, Professor Lars Magnus Hvattum and Professor Bjørn Nygreen, and is submitted to an international journal.

1.3. Contributions

This section presents the contribution of this thesis to the research community and the industry. Lastly, a note on the author's contributions to the papers is given.

1.3.1. Thesis Contribution to the Research Community

The contribution of each paper to the research community is already discussed in Sections 1.2.2 through 1.2.6. The papers are presented at several operations research conferences, including INFORMS 2011 and 2013, ISMP 2012, Optimization Days 2014, IFORS 2014, and EURO 2015. In addition, Paper I is presented at the IEEE/ACM 6th International Conference on Utility and Cloud Computing in 2013.

Papers I and II conduct a combined performance and dependability analysis of fault-tolerant SaaS services. In addition to the contribution of the methods, we believe that studying the performance and dependability of cloud service simultaneously has a value, and deserves more attention in the research community. While there exists several works that either analyze the performance or the dependability of cloud services, there seems to be little work done on a combined analysis.

Papers III, IV, and V consider the same service deployment problem. In addition to the proposed methods, a contribution of these papers is the description of an optimization model that regards placement and resource reservation for passive standby replicas, such that when a failure occurs, a passive replica is ready to take over the service delivery. This consideration is novel compared with the related works in the literature. Regarding the generality of the B&P and ALNS approaches, they are definitely tailored to the problem, but they also utilize some ideas that can be useful in other applications. The subproblem of the B&P is formulated as an SPPRC and solved by an LSA. Furthermore, the formulation is heuristically reduced and the dominance rule of the LSA is thereby simplified. Even though problem-specific features are used in the formulation and the following simplification, we feel that the formulation might be used as inspiration in other applications. The main contribution of ALNS algorithms, with respect to generality, is the inclusion of LS operators. Although the operators are designed for the problem at hand, they are in principle general swap operators. In addition, the idea of using LS operators on top the repair operators are entirely general, and can be used in every application.

1.3.2. Thesis Contribution to the Industry

Many of the guarantees promised in the SLAs of IaaS and SaaS providers are sometimes so imprecise that even major outages do not violate them (Undheim et al., 2011). If the SLA guarantees by purpose are designed to be imprecise, and perhaps worthless, is not discussed in this thesis. However, it might be the case that providers lack the tools to be able to guarantee a reasonable QoS. If this is the case, some of the models incorporating fault tolerance considerations in this thesis might be of use. In many applications, fault tolerance is an important issue, and in many real-world use cases, the users do not tolerate severe service downtimes. As already pointed out in the literature, to increase the adoption of cloud services for business applications, a shift towards higher dependability is needed.

1.3.3. The Author's Contributions to the Papers in the Thesis

All papers included in this thesis has two or more authors, and the aim of this section is to describe the contribution of the first author (the candidate) of these papers. The first author is named the Author in this section. For all five papers, the Author has been responsible for the writing. Nonetheless, all co-authors have given valuable feedback on the writing of the various papers.

The ideas and problems studied in this thesis were established and developed during and after the master's thesis of the Author (Gullhav, 2011). Both Bjørn Nygreen and Poul E. Heegaard took part in this process. Additionally, during the work with Paper III, we had a cooperation with Telenor's Research and Future Studies Department (a Norwegian Telecommunications company), which also provided us with ideas.

The ideas for the solution methods in Papers I and II were proposed by Poul E. Heegaard, but were implemented and refined by the Author with input from Poul E. Heegaard and Bjørn Nygreen. The ideas for the solution methods in Paper III and IV were proposed by Bjørn Nygreen, and the final algorithm was developed as a collaboration between the Author and Bjørn Nygreen. However, the Author did the implementation. Paper V is based on research done at the Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport (CIRRELT) in Montréal, Canada. The ideas of the solution method came from both Lars Magnus Hvattum and Jean-François Cordeau, with whom I had to the pleasure to collaborate with at CIRRELT. The development of the algorithm was done in cooperation with Jean-François Cordeau, Lars Magnus Hvattum, and Bjørn Nygreen. The implementation was done by the Author, but with valuable input from Lars Magnus Hvattum, which helped speed up the code.

1.4. Concluding Remarks and Future Research

Operations research can assist the cloud computing industry in solving different types of decision problems. This thesis studies some problems related to resource allocation, and provides models and methods to aid cloud services providers when confronting these problems. Even though the market for cloud services has been growing and is expected to continue to grow in the coming years, there are still obstacles that may hinder further growth. Low service availability and low QoS guarantees are mentioned among the largest obstacles and weaknesses of cloud computing. We argue that due to the faulty nature of the underlying infrastructure of clouds today, there is a need to account for failures in the QoS assessment of the provided services. This type of analysis is a combined performance and dependability analysis, also denoted performability analysis. Previous research names redundancy as the key to be able to tolerate faults, and a way to implement redundancy in a cloud context is to replicate the VMs of the services. In this view, optimization-based resource allocation methods that regard redundancy techniques should be important for service providers.

Papers I and II consider the problem of obtaining the response time distribution of a multi-tier cloud service by means of a combined performance and dependability analysis. The papers differ in the type of assumptions made; while Paper I assumes exponentially distributed inter-arrival and service times of the requests, and proposes pure analytic methods, Paper II assumes a non-exponential service time distribution, and propose a hybrid approach with an analytic dependability model and a simulation-based performance model. Papers III, IV and V present optimization models with associated solution methods for an SP's resource allocation problem, where the aim is to find a cost-efficient static deployment of a set of services while maintaining a satisfactory QoS. Papers III and IV propose column generation-based algorithms to solve the optimization problem, while the solution method of Paper V is based on the ALNS framework.

Even though the models and methods proposed in the papers of this thesis are ultimately tailored to the problem at hand, there are definitely some general concepts that can be used in other applications. To name a few: the decomposition applied in the analysis in Papers I and II are applicable to many problems, the heuristic simplification of the SPPRC and the corresponding LSA might contain ideas that are useful in other applications, and the concept of using local search operators on top of the standard ALNS framework, as done in Paper V, is still unexplored for different types of problems.

Every research project has its scope and limitations. In this thesis, we are considering some problems with associated features and simplifications within the domain of resource allocation problems. The resources provided by the network that connects users and clouds are often overlooked in the literature on resource allocation problems. In this introduction, we have briefly touched this issue, but it remains out of scope of this thesis. However, Papagianni et al. (2013) present a work in this direction, by providing a unified resource allocation framework for both computing and networking resources.

Moreover, the optimization models and methods of this thesis are based around the assumption of the existence of stationary demand periods of a certain length, and the provided solutions are a static deployment for a given period. This work should be complemented with models and methods that, on the contrary, aim at obtaining solutions for cases where the demand is non-stationary. Such dynamic models could take into account migration of VMs from a node to another, and also provide strategies to recover from severe failures.

Another possible extension of this work is the construction of more complex service models. Within the framework provided in Papers I and II, it is possible to adapt the performance and dependability models to consider more complex service types. Furthermore, the inclusion of requirements for geographical distribution of the VM replicas in the models could be valuable. Geographical distribution of replicas fulfill at least two purposes. Firstly, it makes the services more resilient to data center outages, and secondly in cases where the users are in different geographical regions, the users might get serviced from a nearby VM.

Bibliography

M. Al-Kuwaiti, N. Kyriakopoulos, and S. Hussein. A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability. *IEEE Communications Surveys Tutorials*, 11(2):106–124, 2009.

F. Alam, S. Mohan, J. W. Fowler, and M. Gopalakrishnan. A discrete event simulation tool for performance management of web-based application systems. *Journal of Simulation*, 6(1):21–32, 2012.

D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. Energy-aware autonomic resource allocation in multitier virtualized environments. *IEEE Transactions on Services Computing*, 5(1):2–19, 2012.

D. Ardagna, G. Casale, M. Ciavotta, J. Pérez, and W. Wang. Quality-of-service in cloud computing: modeling techniques and their applications. *Journal of Internet Services and Applications*, 5(1):11, 2014.

M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

N. Ashrafi, O. Berman, and M. Cutler. Optimal design of large software-systems using n-version programming. *IEEE Transactions on Reliability*, 43(2):344–350, 1994.

A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

F. Belli and P. Jedrzejowicz. An approach to the reliability optimization of software with redundancy. *IEEE Transactions on Software Engineering*, 17(3): 310–312, 1991.

E. Bin, O. Biran, O. Boni, E. Hadad, E. Kolodner, Y. Moatti, and D. Lorenz. Guaranteeing high availability goals for virtual machine placement. In *2011 31st International Conference on Distributed Computing Systems*, pages 700–709, 2011.

O. Boxma and H. Daduna. Sojourn times in queueing networks. *Stochastic Analysis of Computer and Communication Systems*, pages 401–450, 1990.

D. Breitgand and A. Epstein. SLA-aware placement of multi-virtual machine elastic services in compute clouds. In N. Agoulmine, C. Bartolini, T. Pfeifer, and D. O'Sullivan, editors, *Integrated Network Management*, pages 161–168. IEEE, 2011.

R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.

R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

H. Cambazard, D. Mehta, B. O'Sullivan, and H. Simonis. Bin packing with linear usage costs – an application to energy management in data centres. In C. Schulte, editor, *Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*, pages 47–62. Springer Berlin Heidelberg, 2013.

R. Chakka and I. Mitrani. Approximate solutions for open networks with breakdowns and repairs. *Stochastic Networks, Theory and Applications. Royal Statistical Society Lecture Notes Series*, 4:267–280, 1996.

L. Columbus. Roundup of cloud computing forecasts and market estimates, 2015, 2015. URL `http://www.forbes.com/sites/louiscolumbus/2015/01/24/roundup-of-cloud-computing-forecasts-and-market-estimates-2015/`. Last visited 2015-11-29.

M. J. Csorba, H. Meling, and P. E. Heegaard. Ant system for service deployment in private and public clouds. In *Proceedings of the 2nd Workshop on Bio-inspired Algorithms for Distributed Systems*, pages 19–28, New York, NY, USA, 2010. ACM.

B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174, Berkeley, CA, USA, 2008. USENIX.

J. Dantas, R. Matos, J. Araujo, and P. Maciel. An availability model for eucalyptus platform: An analysis of warm-standy replication mechanism. In *2012 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1664–1669. IEEE, 2012.

T. Distler, R. Kapitza, I. Popov, H. P. Reiser, and W. Schröder-Preikschat. SPARE: Replicas on hold. In *Proceedings of the 18th Network and Distributed System Security Symposium*, Geneva, Switzerland, 2011. The Internet Society.

T. C. Ferreto, M. A. Netto, R. N. Calheiros, and C. A. D. Rose. Server consolidation with migration control for virtualized data centers. *Future Generation Computer Systems*, 27(8):1027 – 1034, 2011.

G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi. Enhanced modeling and solution of layered queueing networks. *IEEE Transactions on Software Engineering*, 35(2):148–161, 2009.

F. C. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31(1):1–26, 1999.

H. Gavranović and M. Buljubašić. An efficient local search with noising strategy for google machine reassignment problem. *Annals of Operations Research*, pages 1–13, 2014.

R. Ghosh, K. Trivedi, V. Naik, and D. S. Kim. End-to-end performability analysis for infrastructure-as-a-service cloud: An interacting stochastic models approach. In *2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing*, pages 125–132, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

H. Goudarzi and M. Pedram. Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 324–331, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

H. Goudarzi and M. Pedram. Energy-efficient virtual machine replication and placement in a cloud computing system. In *2012 IEEE 5th International Conference on Cloud Computing*, pages 750–757, Los Alamitos, CA, USA, 2012. IEEE Computer Society.

M. Grottke, V. Apte, K. Trivedi, and S. Woolet. Response time distributions in networks of queues. In R. J. Boucherie and N. M. van Dijk, editors, *Queueing Networks*, volume 154 of *International Series in Operations Research & Management Science*, pages 587–641. Springer US, New York, USA, 2011.

A. N. Gullhav. Service deployment in heterogeneous cloud-like environments. Master's thesis, Norwegian University of Science and Technology (NTNU), 2011.

L. Heilig and S. Voß. Decision analytics for cloud computing: A classification and literature review. In A. Newman and J. Leung, editors, *Tutorials in Operations Research–Bridging Data and Decisions*, pages 1–26. INFORMS, Cantonsville, 2014.

F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. L. Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 41–50, New York, NY, USA, 2009. ACM.

Hewlett Packard Enterprise. HPE Helion Eucalyptus, 2015. URL `http://www8.hp.com/us/en/cloud/helion-eucalyptus.html`. Last visited 2015/12/09.

B. Jennings and R. Stadler. Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, 23(3):567–619, 2015.

G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. Generating adaptation policies for multi-tier applications in consolidated server environments. In *International Conference on Autonomic Computing, 2008*, pages 23–32, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

H. H. Kramer, V. Petrucci, A. Subramanian, and E. Uchoa. A column generation approach for power-aware optimization of virtualized heterogeneous server clusters. *Computers & Industrial Engineering*, 63(3):652 – 662, 2012.

W. Kuo and R. Wan. Recent advances in optimal reliability allocation. In G. Levitin, editor, *Computational Intelligence in Reliability Engineering*, volume 39 of *Studies in Computational Intelligence*, pages 1–36. Springer Berlin Heidelberg, 2007.

P. A. Lewis and E. J. Orav. *Simulation methodology for statisticians, operations analysts, and engineers*, volume 1. Wadsworth & Brooks/Cole, Pacific Grove, CA, USA, 1989.

S. Loveland, E. Dow, F. LeFevre, D. Beyer, and P. Chan. Leveraging virtualization to optimize high-availability system configurations. *IBM Systems Journal*, 47 (4):591–604, 2008.

F. Machida, M. Kawato, and Y. Maeno. Redundant virtual machine placement for fault-tolerant consolidated server clusters. In *2010 IEEE Network Operations and Management Symposium*, pages 32–39. IEEE, 2010.

S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi. Cloud computing — the business perspective. *Decision Support Systems*, 51(1):176 – 189, 2011.

P. Mell and T. Grance. The NIST definition of cloud computing, 2011. NIST SP 800-145.

C. Papagianni, A. Leivadeas, S. Papavassiliou, V. Maglaris, C. Cervello-Pastor, and A. Monje. On the optimal allocation of virtual resources in cloud computing networks. *IEEE Transactions on Computers*, 62(6):1060–1071, 2013.

J. Perez and G. Casale. Assessing sla compliance from palladio component models. In *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 409–416. IEEE, 2013.

V. Petrucci, O. Loques, and D. Mossé. A dynamic optimization model for power and performance management of virtualized clusters. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 225–233, New York, NY, USA, 2010. ACM.

H. Qian, D. Medhi, and T. Trivedi. A hierarchical model to evaluate quality of experience of online services hosted by cloud computing. In *2011 IFIP/IEEE International Symposium on Integrated Network Management*, pages 105–112. IEEE, 2011.

S. Ramesh and H. G. Perros. A multi-layer client–server queueing network model with non-hierarchical synchronous and asynchronous messages. *Performance Evaluation*, 45(4):223–256, 2001.

ROADEF. ROADEF/EURO challenge 2012: Machine reassignment, 2012. URL `http://challenge.roadef.org/2012/en/`. Last visited 2015/10/01.

M. Rosenblum. The reincarnation of virtual machines. *ACM Queue*, 2(5):34–40, 2004.

C. Sauer and H. Daduna. Availability formulas and performance measures for separable degradable networks. *Economic Quality Control*, 18(2):165–194, 2003.

M. Sedaghat, F. Hernandez-Rodriguez, and E. Elmroth. A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, pages 6:1–6:10, New York, NY, USA, 2013. ACM.

R. Singh, P. Shenoy, M. Natu, V. Sadaphal, and H. Vin. Predico: A system for what-if analysis in complex data center applications. In *Proceedings of the 12th International Middleware Conference*, pages 120–139, Laxenburg, Austria, 2011. International Federation for Information Processing.

B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13:14–22, 2009.

B. Speitkamp and M. Bichler. A mathematical programming approach for server consolidation problems in virtualized data centers. *IEEE Transactions on Services Computing*, 3(4):266–278, 2010.

H. Sun, J. Han, and H. Levendel. A generic availability model for clustered computing systems. In *Proceedings of the 2001 Pacific Rim International Symposium on Dependable Computing*, pages 241–248. IEEE, 2001.

J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Generation Computer Systems*, 28(2):358 – 367, 2012.

K. S. Trivedi. *Probability & statistics with reliability, queuing and computer science applications.* John Wiley & Sons, New York, USA, 2002.

K. S. Trivedi, G. Ciardo, M. Malhotra, and R. A. Sahner. Dependability and performability analysis. In L. Donatiello and R. Nelson, editors, *Performance Evaluation of Computer and Communication Systems*, volume 729 of *Lecture Notes in Computer Science*, pages 587–612. Springer Berlin Heidelberg, 1993.

A. Undheim, A. Chilwan, and P. E. Heegaard. Differentiated availability in cloud computing slas. In *2011 12th IEEE/ACM International Conference on Grid Computing*, pages 129–136, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An analytical model for multi-tier internet services and its applications. *SIGMETRICS Performance Evaluation Review*, 33(1):291–302, 2005.

B. Urgaonkar, A. L. Rosenberg, and P. Shenoy. Application placement on a cluster of servers. *International Journal of Foundations of Computer Science*, 18(05): 1023–1041, 2007.

VMware White Paper. Protecting mission-critical workloads with vmware fault tolerance, 2009. URL `https://www.vmware.com/files/pdf/resources/ft_virtualization_wp.pdf`.

L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl. Scientific cloud computing: Early definition and experience. In *Proceedings of the 10th International Conference on High-Performance Computing and Communications*, pages 825–830. IEEE, 2008.

B. Wickremasinghe, R. Calheiros, and R. Buyya. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 446–452, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

S. P. Woolet. *Performance analysis of computer networks.* PhD thesis, Duke University, Durham, NC, USA, 1993.

L. Wu and R. Buyya. Service level agreement (SLA) in utility computing systems. In *Grid and Cloud Computing: Concepts, Methodologies, Tools and Applications*, pages 286–310. Information Resources Management Association, 2012.

K. Xiong and H. Perros. Service performance and analysis in cloud computing. In *2009 World Conference on Services - I*, pages 693–700, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

Paper I

Anders N. Gullhav, Bjørn Nygreen and Poul E. Heegaard:

Approximating the Response Time Distribution of Fault-tolerant Multi-tier Cloud Services

Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, pp. 287-291

Approximating the Response Time Distribution of Fault-tolerant Multi-tier Cloud Services

Abstract:
Cloud services with a multi-tiered architecture are often difficult to evaluate in terms of performance and dependability. The tiered architecture complicates the resource capacity decisions of the service provider, and makes it more demanding to maintain a good balance between capacity and quality of service. In this work we present an approximation of the response time distribution of a multi-tier service, which should help the providers in their planning and operation. The approximation also takes into account failures in the virtual machines in which the service runs, and acknowledges replication of the tiers. We demonstrate that the approximation can be used as decision support for service providers.

2.1. Introduction

In the resource planning and operation of cloud services, providers need tools which express a relation between resource capacity and quality of service (QoS) of their services (Iyoob et al., 2013). Many cloud services can be modeled as being composed of different functional components running in several virtual machines (VMs), where the components belong to different tiers in the service. When a customer requests the service, the request will be processed at possibly each of the tiers before the response is returned. Hence, one can abstract such a service as an open queuing network where each queue corresponds to a tier. These types of services are often denoted *multi-tier* services, and a typical three-tier service is composed of a web server tier, an application logic tier and a database tier.

In multi-tier services the response time is an important statistic which quantifies the QoS. Typically the mean response time is used as the QoS-metric, but we argue that the percentiles (e.g. 90th or 95th percentiles), or more generally the response time distribution, are of higher importance to the end-users.

In addition to low response time, end-users require that the services they use are reliable. Even if failures happens rarely, a multi-tier service which relies on several VMs might be severely affected by a single failure if proper measures are not taken. The result might be high response times or even downtime. A means to achieve a reliable service on an unreliable infrastructure, like the cloud data centers, is utilization of fault-tolerance techniques such as redundancy (Avižienis et al., 2004). Remus (Cully et al., 2008) is a technique for hot standby replication of VMs where the standby replicas share the backup resources.

For the multi-tier services considered herein we assume that each tier can be replicated into a number of copies, where each run in a separate VM. Henceforth, we denote these VMs running the same service component simply as replicas and assume that all replicas of a tier have equal service capabilities. We distinguish between *active* and *passive* replicas, where active replicas serve demand and passive replicas are hot standbys sharing their resources with other passive replicas. We assume that there can be more than one active replica of a given tier, and apply a static load-balancing scheme with equal probability of selecting an active replica at that tier.

There have been much research on the problem of computing the response time distribution of queuing networks, and a review, now fairly dated, is given by Boxma and Daduna (1990). Determination of the response time distribution in a network is generally difficult, especially when the network is not overtake-free (Boxma and Daduna, 1990). In such cases one usually approaches the problem by making an approximation. One type of approximation is based on an idea to decompose the network into single queues and analyze each queue in isolation. Woolet (1993) presented a method for $M/M/c/b$ queues based on this principle, which again builds on an approach presented in Harrison (1981). Woolet's method is in Grottke et al. (2011) revisited and extended to queues which have phase-type service time distribution. This method is used by Heegaard and Trivedi (2009) in order to model the survivability of a telecommunication network. Our approximation is based the method presented in Woolet (1993) and Grottke et al. (2011), but we also take into account failures in the servers. There also exist research deriving approximate solutions for queuing networks with failures, like Chakka and Mitrani (1996) and Sauer and Daduna (2003). But to our knowledge there are not much work done on computing response time distributions in queuing networks with replicated servers.

The main contribution of this work is to provide a model of a multi-tier cloud service deployed on an unreliable infrastructure, from which one can extract an approximate response time distribution. This approximation takes into account performance degradation as an effect of failures. In addition the approximation gives us the possibility to evaluate and compare the performance of different operational configurations for a given service.

In the next section we present a failure model used to extract the failure probabilities of the tiers, while Section 2.3 details our approximation of the response time distribution. In Section 2.4 we demonstrate how the approximation could be used as decision support for providers, and finally, Section 2.5 concludes the paper.

2.2. Failure Model

Dependability can be defined as the ability of a system to deliver a service which can justifiably be trusted (Laprie, 1992). Moreover, performability is typically defined as a measure of the likelihood that some subset of functions of a system is performed correctly (Al-Kuwaiti et al., 2009), for example that the response time of a service should be less than T seconds with probability P. Performability couples performance and dependability, and is an attribute of fault-tolerance which can be defined as the ability of a system to continue normal operation despite the presence of hardware or software faults (Al-Kuwaiti et al., 2009). Fault-tolerance if often used as a means to achieve a dependable system, and a typical approach is to introduce redundancy via replication of system components, like software and hardware, but also replication of information. For software replication, which is considered herein, there exist different types of approaches, generally classified according to whether the passive replicas are *hot* or *cold*, and have *dedicated* or *shared* access to resources.

A service failure, or failure, can be seen as a transition from correct service to incorrect service (Avižienis et al., 2004). We only consider failures from the physical servers, the software and operation, but exclude the application software failures which take down more than one replica. We assume that the replicas of a service run in different VMs on different physical machines, and thus make the assumption of independent failures, both between replicas of the same tier and between tiers.

When a tier operates correctly, a request from the end-user is transmitted to one of the active replicas of that tier and the selected active replica processes the request. At regular time intervals the state information in the active replicas is transmitted to the passive replicas, such that the passive replicas are ready to take over service delivery in case of a failure. So, when a failure occurs in one of the active replicas, a passive replica is made active and starts to serve the end-users, and the failed replica is being repaired.

Mathematically, we denote the failure rate of active and passive replicas as γ and β, respectively, and assume that the failures occur according to a Poisson process. Furthermore, we let the repair time be exponentially distributed with parameter δ, and assume that the replicas are repaired independently. When an active replica fails, we assume that the time until the passive replica is ready to serve demand is negligible. Thus the passive replica is activated instantly. Moreover, we let k and m be the number of active and passive replicas of a tier, and $n = k + m$.

We represent the failure model described above as a Continuous Time Markov Chain (CTMC). The CTMC is depicted in Figure 2.1, where the state is represented by the tuple (# active not failed, # passive not failed). Note that we assume to have enough "repair men" to repair all replicas simultaneously.

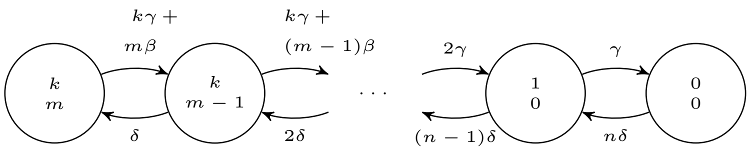

Figure 2.1.: Failure model of a tier represented as a CTMC

Denote the transition rate matrix of the CTMC in Figure 2.1 as G and let $\boldsymbol{\pi}$ denote the vector of stationary probabilities of the CTMC. $\boldsymbol{\pi}$ is found by solving the linear system of equations $G\boldsymbol{\pi} = \mathbf{0}$ with the additional normalizing constraint $\sum_{i\in\mathcal{I}} \pi_i = 1$, where $\mathcal{I}$ is the set of states of the CTMC and $\mathbf{0}$ is a column vector of only zeros. Generally, vectors are columns indicated with bold font, i.e. $\boldsymbol{x}$, and we denote rows as $\boldsymbol{x}^\intercal$.

2.3. Approximation of the Response Time Distribution

Our approximation of the response time distribution of a multi-tier service is based on the response time block method (Woolet, 1993; Grottke et al., 2011) for finding response time distributions in open queuing networks. The method makes the assumption that successive response times of the queues in a path through the network are independent, which is true for some special cases (Boxma and Daduna, 1990). If this is not the case, an approximate distribution is returned. We give an overview of the method below.

2.3.1. The Response Time Block Method

The main idea of the method is to assume that the response time distribution at each queue in the network can be represented by the distribution of time to reach an absorbing state of a CTMC. The CTMCs representing the individual queues are denoted response time blocks, and from these one constructs a total CTMC by "gluing" the CTMCs of the queues together. The response time distribution of the network is then given by the absorption time distribution of the total CTMC.

For an $M/M/1$/FIFO queue with arrival rate λ and service rate μ, the response time distribution is an exponential distribution given as $1 - e^{-(\mu-\lambda)t}$ (Trivedi, 2002, Sect. 8.2). If the queue is stable, i.e. $\lambda < \mu$, the response time block can be depicted as in Figure 2.2. The "In" state is the starting state of this CTMC model, while the "Out" state is either an "In" state for another response time block or the absorbing state in the total CTMC. Similarly one can construct response time blocks for other types of queues. However as we will come to, we model the replicas as $M/M/1$ queues, and refer to Grottke et al. (2011) for description of response time blocks of more complex queues.

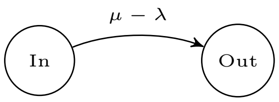

Figure 2.2.: Response time block for an $M/M/1$ queue

Now, considering a network of Q $M/M/1$ queues in series, one could construct a response time block for each of the queues, and glue them together by modeling the "Out" states of a given queue as the "In" state of the next. The "Out" state of the Qth queue corresponds to the absorbing state representing the departure from the network. Figure 2.3 shows the CTMC of this queuing network, where λ_q and μ_q, $q \in \mathcal{Q} = \{1, \ldots, Q\}$ are the respective arrival and service rates of the queues, and "Abs" is the absorbing state.

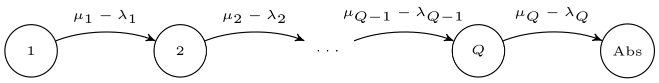

Figure 2.3.: CTMC of Q $M/M/1$ queues in series

To be able to find the distribution of time to reach the absorbing state of the queuing network in Figure 2.3, i.e. the response time distribution, define H as the transition rate matrix on the set of states $\mathcal{Q}' = \{1, \ldots, Q, Q+1\}$, where the last state is the absorbing state. Let $\boldsymbol{p}(t) = [p_1(t), \ldots, p_{Q+1}(t)]^\intercal$ be the vector of probabilities of being in state $q' \in \mathcal{Q}'$ at time t. $p_{Q+1}(t)$ will then correspond to the response time distribution and is found by solving (2.1) with initial state probabilities $\boldsymbol{p}(0) = [1, 0, \ldots, 0]^\intercal$.

$$\frac{d}{dt}\boldsymbol{p}(t) = H\boldsymbol{p}(t) \tag{2.1}$$

2.3.2. Including Failures in the Response Time Block Method

Before we begin to elaborate on how to include failures in the response time blocks, we should first consider our case with one or more active replicas in parallel at each tier under the assumption that no failures occur. We generally assume that a request is processed exactly once at each of the tiers and that the arrival and service rates are Poisson. Thus the multi-tier service can be modeled as a tandem-like queuing network with one or more $M/M/1$ queues at each tier and only feed-forward arcs. Figure 2.4 visualizes a three-tier service with three active replicas at tier 1, two active replicas at tier 2 and two active replicas at tier 3 as a queuing network. Note that the passive replicas do not serve demand in a failure-free situation and hence are not shown in the figure. We assume that the active replicas of the same tier have equal service time distribution and that the arrivals are split uniformly between the active replicas at each tier. Considering the example service in Figure 2.4, the latter means that if the total service demand is λ requests per time unit, the arrival rate at an active replica of tier 1 is $\lambda/3$ and the arrival rate at an active replica of tier 2 and 3 is $\lambda/2$. Assuming that the demand arrives according to a Poisson process and the service times are exponentially distributed with parameter μ_1, μ_2 and μ_3 at each active replica of the three tiers, respectively, the response time of a request at tier 1 will be exponentially distributed with parameter $\mu_1 - \lambda/3$. Likewise, the response time of a request at tier 2 and 3 will be exponentially distributed with parameter $\mu_2 - \lambda/2$ and $\mu_3 - \lambda/2$. Since the active replicas at each tier are equal and the requests to a tier are split equally between the active replicas, the network in Figure 2.4 can be modeled as the CTMC in Figure 2.5.

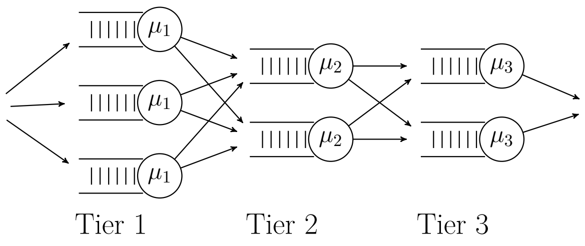

Figure 2.4.: A multi-tier service depicted as a queuing network

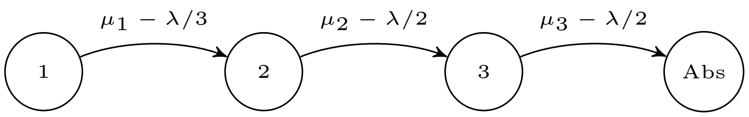

Figure 2.5.: CTMC of the example three-tier service (in Figure 2.4)

Now, let us introduce some more mathematical notation. Let $\mathcal{Q} = \{1, \dots, Q\}$ be the set of tiers in our Q-tier service, and for each tier $q \in \mathcal{Q}$ let $\mathcal{I}_q$ denote the set of states in the CTMC *failure model* of that specific tier (cf. Section 2.2).

Then, denote the stationary probability vector of the failure model of tier q as $\boldsymbol{\pi}_q$, and let a_{qi} be the number of active replicas in failure state i of the failure model of tier q.

To include failure in the CTMC of the response time block we recognize that a failure state i in the failure model of tier q results in a_{qi} active replicas at that tier. So if we treat the CTMC of the failure model of the individual tiers as a Markov reward model (Trivedi, 2002), where the rewards, λ_{qi}, are given by (2.2), we could interpret these rewards as the arrival rate to a queue at tier q when the tier is in failure state i. Based on these rewards we define $\mathcal{I}_q^S = \{i \in \mathcal{I}_q \colon \lambda_{qi} < \mu_q\}$ as the set of states where the resulting $M/M/1$ queue at tier q with arrival rate λ_{qi} would be stable. We also define $\mathcal{I}_q^B = \mathcal{I}_q \setminus \mathcal{I}_q^S$ as the set of non-stable states, and let the blocking probability of a tier be defined as $\pi_{Bq} = \sum_{i \in \mathcal{I}_q^B} \pi_{qi}$. Then we scale the stationary probabilities of the failure model according to (2.3), so that the stationary probabilities of the failure states resulting in a stable tier sum to 1, and compute the expected arrival rate to an active replica in tier q, $\bar{\lambda}_q$ as in (2.4).

$$\lambda_{qi} = \begin{cases} \frac{\lambda}{a_{qi}} & \text{if } a_{iq} > 0 \\ \infty & \text{otherwise} \end{cases} \tag{2.2}$$

$$\pi'_{qi} = \frac{\pi_{qi}}{1 - \pi_{Bq}} \tag{2.3}$$

$$\bar{\lambda}_q = \sum_{i \in \mathcal{I}_q^S} \pi'_{qi} \lambda_{qi} \tag{2.4}$$

When we build the CTMC based on the response time block method, the transition rates from state $q \in \mathcal{Q}$ to $q+1$ will be $\mu_q - \bar{\lambda}_q$, and the response time distribution of the queuing network, given by $p_{Q+1}(t)$, is found by solving (2.1).

In this presentation of our approximation three matters should be noted. Firstly, we return to the assumptions made by the response time method regarding independent response times in successive queues in a path through the network. One of the conditions for this assumption to hold is that the network should be overtake-free (see Boxma and Daduna (1990) for a discussion). The described service in this work can be visualized as the network in Figure 2.4, and hence a request may overtake another by taking a different path through one of the tiers. This means that the queuing network characterizing the multi-tier service violates the assumption about independent response times, and therefore the response time block method will provide an approximate distribution. Secondly, since we assume independent failures and repairs between replicas of different tiers, we can consider the tiers independently in the failure model and hence also compute $\bar{\lambda}_q$ independently. Lastly, the calculation of the approximate response time distribution only reflects the response time of the failure states resulting in stable queues. The reason behind this modeling decision is the fact that even if a tier is not completely down, i.e. all replicas are failed, the resulting response time would probably be too high to be within the tolerance level of the end-users. Therefore we treat all failure states which result in non-stable queues as blocked, which in turn means that the distribution will be a defective distribution (Trivedi, 2002), specifically the response time distribution given that all tiers are stable. Let $\pi_{SB} = 1 - \prod_{q \in \mathcal{Q}} \sum_{i \in \mathcal{I}_q^S} \pi_{qi}$ be the probability that the service is blocked, then the defective response time distribution, $F_T(t)$, is given by (2.5).

$$F_T(t) = (1 - \pi_{SB}) p_{Q+1}(t) \tag{2.5}$$

In order to test the correctness of the approximation, we have compared the approximation with an empirical response time distribution acquired by simulation of several test cases. Because of the page limitation of this paper, we omit the results of these tests, but the tests show that the deviation between the approximation and the simulated response time distributions is small.

2.4. Decision Support for Providers

Having presented our approximation of the response time distribution we will in this section consider an example of a three-tier service, and assume that the service provider presents its end-users a service level agreement (SLA) specifying that the service should provide a response time below T seconds with probability P. Table 2.1a displays the service data, where the service rate vector $\boldsymbol{\mu} = [\mu_1, \mu_2, \mu_3]^\intercal$ gives the service rate for each active replica of each tier. Note that we assume that all active and passive replicas have the same failure rate, γ and β, respectively, and equal repair rate, δ. We should mention that the service data are guesstimates but considered to be realistic.

Table 2.1.: Data and replication patterns for the example service

Parameter	Value
γ	0.041667 hr^{-1}
β	0.020833 hr^{-1}
δ	2.0 hr^{-1}
λ	100 sec^{-1}
$\boldsymbol{\mu}$	$[60, 70, 80]^\intercal$

(a) Service data

RP	$\boldsymbol{k}^\intercal$	$\boldsymbol{m}^\intercal$
1	$[3, 3, 2]$	$[0, 0, 0]$
2	$[3, 3, 2]$	$[0, 0, 1]$
3	$[4, 3, 2]$	$[0, 0, 0]$
4	$[4, 3, 2]$	$[0, 0, 1]$
5	$[3, 3, 3]$	$[0, 0, 1]$

(b) Replication patterns (RP)

To show how a service provider could utilize the approximation as decision support, we will assess the response time distribution of the example service with varying numbers of active and passive replicas at each tier, while keeping the demand, service rates and failure characteristics fixed. We name a combination of the number of active and passive replicas at each tier of the service a *replication pattern*. The number of active replicas of the tiers is given by the vector $\boldsymbol{k} = [k_1, k_2, k_3]^\intercal$, and the number of passive replicas is correspondingly given by $\boldsymbol{m}$, so a replication pattern is defined by the tuple $(\boldsymbol{k}, \boldsymbol{m})$. Table 2.1b presents five replication patterns (labeled RP 1 to RP 5), which we will analyze below.

A plot of the five resulting response time distributions is depicted in Figure 2.6, and Table 2.2 summarizes some important statistics about the distributions. Now, we consider an SLA specifying that the response time should be below 0.2 seconds with probability 0.95, and examine the five replication patterns in order.

- If we start by inspecting RP 1, we see that the resulting blocking probability is too high for it to satisfy the SLA requirement. A simple inspection show that tier three is the weak link, and hence if we add a passive replica to this tier, one should expect the blocking probability to reduce significantly. This is seen in RP 2.

- If we instead of adding a passive replica to tier 3 add another active replica to tier 1, we obtain RP 3. We can see that the blocking probability of RP 2 is much lower than that of RP 3, but regarding the mean response time, we observe the opposite. Nevertheless, considering the SLA requirement, RP 2 is better than RP 3 as the value of the 95th percentile is lower.

- Now combining RP 2 and 3 to obtain the replication pattern denoted RP 4, we now see that the SLA requirement is satisfied, and one might conclude that we should stop investigating more replication patterns.

- However on the basis of RP 4, what if we add one active replica to the third tier and jointly remove one at the first. Now, we end up with RP 5, and we see that they are almost equal performance-wise.

In order to identify if RP 4 is better than RP 5, one could try to compute the cost of both replication patterns and compare them. This might not be straight forward. A service provider might offer several services which interrelate in some way, such that the cost of a replication pattern of one service is dependent on other services. In this case, it might be possible to use an optimization model to simultaneously select the optimal replication pattern for each service in the service portfolio of the provider.

Table 2.2.: Statistics of the replication patterns (times in seconds)

Case	Mean response time	Percentiles		π_{SB}
		90th	95th	
RP 1	0.095998	0.20316	0.29586	0.042767
RP 2	0.099936	0.18002	0.21475	0.0034988
RP 3	0.086743	0.18200	0.25957	0.041618
RP 4	0.090301	0.16164	0.19211	0.0023024
RP 5	0.088178	0.15939	0.19076	0.0024933

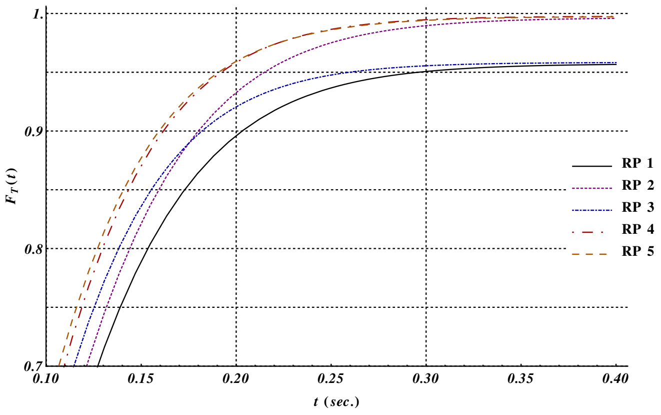

Figure 2.6.: Graphical comparison of the response time distribution of the replication patterns. Note that the origin of the plot is in $(0.1, 0.7)$.

2.5. Conclusions

We have presented a method for finding an approximation of the response time distribution of multi-tiered cloud services. In addition we demonstrated how the approximation could be used as decision support for service providers in their planning process. We also noted that it is not necessarily straight forward to decide which of two or more replication patterns that is the best if all satisfy the SLA requirement. Thus, this approximation might be used in conjunction with an optimization model to find the optimal configuration of a set of services.

Bibliography

M. Al-Kuwaiti, N. Kyriakopoulos, and S. Hussein. A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability. *IEEE Communications Surveys Tutorials*, 11(2):106–124, 2009.

A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

O. Boxma and H. Daduna. Sojourn times in queueing networks. *Stochastic Analysis of Computer and Communication Systems*, pages 401–450, 1990.

R. Chakka and I. Mitrani. Approximate solutions for open networks with breakdowns and repairs. *Stochastic Networks, Theory and Applications. Royal Statistical Society Lecture Notes Series*, 4:267–280, 1996.

B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174, Berkeley, CA, USA, 2008. USENIX.

M. Grottke, V. Apte, K. Trivedi, and S. Woolet. Response time distributions in networks of queues. In R. J. Boucherie and N. M. van Dijk, editors, *Queueing Networks*, volume 154 of *International Series in Operations Research & Management Science*, pages 587–641. Springer US, New York, USA, 2011.

P. G. Harrison. Approximate analysis and prediction of time delay distributions in networks of queues. In *Int. CMG Conference*, pages 70–80, 1981.

P. E. Heegaard and K. S. Trivedi. Network survivability modeling. *Computer Networks*, 53(8):1215–1234, 2009.

I. Iyoob, E. Zarifoglu, and A. B. Dieker. Cloud computing operations research. *Service Science*, 5(2):88–101, 2013.

J.-C. Laprie, editor. *Dependability: Basic Concepts and Terminology.* Springer-Verlag, New York, 1992.

C. Sauer and H. Daduna. Availability formulas and performance measures for separable degradable networks. *Economic Quality Control*, 18(2):165–194, 2003.

K. S. Trivedi. *Probability & statistics with reliability, queuing and computer science applications*. John Wiley & Sons, New York, USA, 2002.

S. P. Woolet. *Performance analysis of computer networks*. PhD thesis, Duke University, Durham, NC, USA, 1993.

Paper II

Anders N. Gullhav, Bjørn Nygreen and Poul E. Heegaard:

Simulation of the Response Time Distribution of Fault-tolerant Multi-tier Cloud Services

submitted to an international journal

Paper III

Anders N. Gullhav and Bjørn Nygreen:

Deployment of Replicated Multi–tier Services in Cloud Data Centres

International Journal of Cloud Computing 4 (2), 2015, pp. 130-149

Deployment of Replicated Multi–tier Services in Cloud Data Centres

Abstract:
A provider of cloud software services is faced with different decisions related to the operation of his services. Firstly, he needs to configure the services such that the quality of service is in accordance with the specified requirements, and secondly, the services need to be deployed in the provider's private cloud in a cost and energy-efficient manner. We argue that these two sets of decisions need to be taken simultaneously in order to make an optimal decision, and we present mixed integer linear programming models optimising these decisions. One of our models also allows the provider to deploy services in a public cloud. We are testing a direct formulation against a reformulation utilising pre-generated node patterns, and observe that the reformulation produces solutions of better quality within a fixed runtime. Furthermore, the effects of bursting services into a public cloud are investigated.

4.1. Introduction

The providers of cloud software services, like web services and business applications, are on a short-term basis faced with different decision problems. In the service delivery, the provider faces the requirements of the end-users towards quality of service (QoS), and such requirements are typically specified in the service level agreements (SLAs). Typical QoS requirements include maximum average response time, maximum downtime or more complex requirements such as an upper bound on the 95th percentile of the response time distribution. In order to satisfy the end-users' requirements for a given service, the provider needs to allocate enough resources to the service. In addition, for business critical applications which are sensitive to downtime or high response times, one also needs to consider the consequences of failures. A means to achieve a service tolerant to failures is to introduce software redundancy through replication of the virtual machines (VMs) involved in the service. In this paper, the services in focus are a collection of collaborating software components, where each component has different functionality and all components are required in the service delivery. This is commonly known as a multi-tier service, and a typical service is a three-tier web service composed of a web server, an application logic and a database server. To increase the fault-tolerance of a multi-tier service, we introduce the option to use passive backup replicas of the different components, i.e. tiers, which are ready to serve the end-users whenever a failure occurs. However, because of the complex nature of multi-tier services, deciding on the appropriate replication level of each component in a multi-tier service and the amount of resources allocated to the replicas to satisfy the QoS requirements while being cost-efficient is generally difficult.

Service providers which operate their own data centre, a private cloud say, also seek to minimize the operational costs of their hardware servers. We denote the servers as nodes in the following. A crucial cost component in the operation of data centres in a short-term perspective is the costs of energy usage needed for running and cooling down the nodes. Although the idle power consumption of CPUs is decreasing, other components in the nodes are still power-inefficient in an idle state, and thus, an idle node might still consume more than 70 per cent of the peak power (Beloglazov et al., 2011). This implies that a means to reduce the power consumption of data centres is to turn off unused nodes or let these enter a low-power mode. In turn, this leads to a strategy where the VMs composing a service are deployed on a minimal subset of the nodes.

We see that we have two different types of decision problems, namely the problem of selecting replication levels of the service components such that the QoS requirements are fulfilled, and the problem of deploying the resulting replicas (i.e. VMs). These two problems are interrelated as the cost of deployment is dependent on the replication levels chosen for the different components of all the services offered by the provider. Therefore, decisions in both problems should be taken simultaneously, and furthermore, in order to make the optimal decisions, all services should be treated together when deciding on the deployment.

The overall problem can be modelled and solved in a static or dynamic way. When solving the problem statically, one sees the demand as stochastic and stationary, and the resulting solution would be a stationary deployment and configuration of the services and the nodes. When the demand is periodic and the periods are sufficiently long, one can find a stationary solution for each demand period and apply a solution when one enters the corresponding period. Such periods might be working hours, evenings, weekends, etc. with different characteristics. On the other hand, a dynamic version of the problem would be solved whenever it is necessary. A condition which will make it necessary to dynamically solve the problem is that the demand leaves its stationary state, e.g. there are non-periodic spikes in the demand, and hence, a new solution is needed quickly. Another dynamic problem is to decide the temporary migration of VMs enforced by failure situations. In this work, we are focusing on modelling and solving the problem statically. However, we take also into account failures in the VMs in the services, although in a static manner. Since we are focusing on finding a stationary solution, the time to find the optimal or near-optimal solution is of less importance than the quality of the solution. Therefore, we emphasise obtaining a solution with high quality above constructing fast algorithms, and we develop mixed integer linear programming (MILP) models considering both the replication of the multi-tier services and the deployment of the replicas. To maintain linearity while including generally non-linear QoS requirements in the models, we utilise pre-generated configurations for each of the multi-tier services, denoted *replication patterns*, which specify replication levels of the different components resulting in satisfactory QoS. In the formulation of the MILPs, we have taken two approaches. Firstly, we present a direct formulation, and secondly, we give a reformulation utilising pre-generated *node patterns*. A node pattern is here a feasible combination of different VMs deployed on a node, e.g. respecting the node capacity and other constraints. In a third model, we also allow the service provider to burst replicas, i.e. VMs, into a public cloud.

In the next section, we present other research related to our work and show how this paper fits into the literature. Then, in Section 4.3, we give a detailed specification of the problem, before we present our mathematical models, in Section 4.4. Section 4.5 contains our results and discussion of these, and lastly, Section 4.6 concludes this paper.

4.2. Related Work

There exist research on several different decision problems in cloud computing systems, and an overview of operations research problems with associated references categorised according to whether the perspective is from a provider, consumer or broker is presented by Iyoob et al. (2013). In this section, we will review some literature on the deployment problems faced by a service provider in a cloud computing context. In addition, we will introduce some work which utilises pre-generated columns, e.g. routes or packing patterns, in mathematical programming models.

4.2.1. Related Work on Deployment Problems

Hermenier et al. (2009) present solution procedures for dynamically allocating VMs to nodes in clusters and creating reconfiguration plans for migration of the VMs, using constraint programming. The objective is to minimise the number of nodes used, in order to save energy, while ensuring that each VM has access to sufficient memory and CPU power. Ferreto et al. (2011) propose heuristic solution methods for a multidimensional bin-packing problem (map VMs to nodes) with migration control. Speitkamp and Bichler (2010) present an algorithm for a static server allocation problem, which consists of finding mappings between services and capacitated servers and minimize the cost of the servers needed. The authors state that the problem is strongly NP-hard, even when only one type of resource is considered and all servers have the same cost and capacity. Another approach looking at the allocation problem statically is presented by Goudarzi and Pedram (2012). They model a problem consisting of decisions related to both the replication and placement of VMs. The objective of their heuristic algorithm used to solve the model is to minimize the cost of energy usage. Petrucci et al. (2010) have taken an approach to dynamically manage the cluster power consumption through a MILP model, where one is to take decisions about which servers that are active, their respective CPU frequencies, and find a mapping between a set of software applications and the server. The objective of the approach is to minimise power consumption while meeting the performance requirements of the applications. The problem is modelled as a variant of the one dimensional variable sized bin-packing problem. Rao et al. (2010) propose a joint load balancing and power control scheme for distributed cloud data centres, where load balancing is conducted on the data centre level and power control is conducted on the individual servers.

There has also been done some work on developing column generation methods for service deployment problems. Breitgand and Epstein (2011) use a combinatorial auctions approach to solve a problem where the objective is to maximize profit by placing VMs on a fixed number of capacitated nodes. On small test instances, the results show that the column generation approach is significantly faster than the direct integer programming (IP) formulation when setting a target optimality gap of 10 per cent. On the larger test instances, the direct IP formulation is not capable of finding a solution within the time limit, but the column generation approach is still able to find reasonably good solutions. Cambazard et al. (2013) develop a column generation method to compute dual bounds of a multi-period bin packing problem with linear usage costs of the bins, reflecting the energy costs of nodes. Their approach computes tight bounds on test instances provided by the industry.

While the literature reviewed above contains models mainly focusing on data centres operated by a single entity, Van den Bossche et al. (2010) introduce a binary integer programming (BIP) model that is used to deploy a set of applications in a hybrid cloud environment. The objective is to minimise the costs of the service provider for executing the applications. The private clouds are capacitated in terms of CPU power and memory, and hence, tasks must be outsourced if there is no free capacity in the private clouds. Tordsson et al. (2012) take the role of a cloud broker and optimise the deployment of VMs among multiple clouds. The authors describe a cloud deployment BIP model minimising the total cost of deployment, while ensuring that the requirements of the customers are fulfilled. They include simple constraints for modelling the customers' requirements, such as a minimum and maximum size of the VMs and a minimum and maximum load to be placed in each cloud.

The work mentioned above do not consider the facts that VMs might be structurally dependent and that the services consist of multiple tiers. When ensuring satisfactory QoS for a multi-tier service, one needs to take into account the total QoS over all tiers, and not treat the VMs individually. Such considerations are included in the resource allocation models of Goudarzi and Pedram (2011) and Ardagna et al. (2012). Both works consider the cost of power usage and models penalties for not satisfying the agreed requirements on the mean response time. Their resulting models are mixed-integer non-linear programs, and they develop heuristic algorithms for solving the problem. Like our work, these two papers model the services as composed of several tiers and consider QoS metrics involving the performance of all VMs of a service. However, while our objective is to provide a stationary solution, Goudarzi and Pedram (2011) and Ardagna et al. (2012) takes a dynamic perspective when modelling and solving the deployment and resource allocation problem.

None of the works reviewed in this section and none to our knowledge considers failures or fault-tolerance techniques together with deployment and resource allocation decisions. In our models, we include passive back-up replicas to improve the fault-tolerance of the services. Another novelty of our work is the introduction of replication patterns which makes it possible to include complex QoS requirements for multi-tier services in linear optimisation models.

4.2.2. Related Work on Pre-generation of Columns

Pre-generation of columns in optimisation models have been utilised in several applications, such as routing, scheduling and bin-packing problems. Fagerholt and Christiansen (2000) and Fagerholt (2001) consider ship scheduling problems, and pre-generate candidate ship schedules for each ship. After the generation of candidate schedules, the optimisation model select one schedule out of the candidates for each ship. They acknowledge that a full enumeration of all feasible candidate schedules may require large computational times, and therefore they reduce the number of schedules heuristically by only allowing schedules with a ship capacity utilisation over a certain threshold. Another application of column pre-generation in a ship routing and scheduling problem is presented by Hennig et al. (2012). Therein, only a subset of the feasible routes are generated, and the candidate routes are selected based on capacity utilisation and sailing cost.

In a bin packing-like problem formulation considering multiple container loading, Eley (2003) uses a greedy heuristic to pre-generate candidate packing patterns. The resulting patterns are all *maximal*, i.e. there is not capacity for adding more items to the packing patterns.

4.3. QoS-aware Deployment of Multi-tier Services

Generally, the problem discussed in this work consists of finding cost-effective deployments of the cloud software services of a single service provider on a given infrastructure, while ensuring satisfactory QoS. The infrastructure is either self-owned, in the following denoted a private cloud, or leased from an infrastructure-as-a-service (IaaS) provider, i.e. a public cloud. The service provider provides a set $\mathcal{S}$ of multi-tier services, and each service $i \in \mathcal{S}$ consists of a set $\mathcal{Q}_i$ of components (tiers). SLAs specify QoS requirements for each service i, and to comply with the requirements the service components might be replicated. The modelling of the decisions concerning replication is explained in Section 4.3.2.

4.3.1. Service Deployment and Resource Consumption

Firstly, considering the deployment in the private cloud of the service provider, we let $\mathcal{N}$ be the set of nodes, indexed by n, which can be used for deployment, and $\mathcal{G}$ be the set of resources provided by the nodes, indexed by g. In this work, we treat all nodes as identical. In the replication of the service components, we distinguish between two types of replicas: replicas which serve demand and backup replicas which do not serve demand unless a failure has occurred. We denote the former type of replica an *active replica* and the latter a *passive replica*. When deploying the replicas, we forbid replicas of the same component to be deployed on the same node, and we denote this as node-disjoint deployment. This rule is employed to ensure that in case a node fails, no more than one replica of a given component is brought down.

We use the binary variables w_{iqn} and v_{iqn} to indicate the deployment of an active and passive replica of component q of service i on node n, respectively. From now on, we will denote the component q of service i as the pair (i, q) for simplicity. In a virtualised data centre, the nodes run a Virtual Machine Monitor (VMM) which is handling the resource management and guaranteeing each replica a minimum amount of the resources, e.g. CPU and memory. The amount of the resources each replica is guaranteed might be modelled as a variable or a fixed parameter. In this work, we have chosen to fix the amount of resources before the optimisation, and in addition, we have let the amount of a given resource guaranteed to a replica be independent of which node the replica runs on. Mathematically, we denote G_{Aiqg} the amount of resource type $g \in \mathcal{G}$ guaranteed to an active replica of the pair (i, q) on the node where the replica is deployed. Considering passive replicas, these should be able to quickly take over service delivery after a failure affecting a corresponding active replica. In order to do so, the passive replicas might need to maintain some state information. This is done by synchronisation with the active replicas, and thus, the passive replicas need to be guaranteed an amount of resources for management purposes. We denote the amount of resource

g guaranteed to a passive replica of the pair (i, q) by G_{Piqg}.

As mentioned, we will also present a model allowing for deployment in a public cloud. This type of deployment, often referred to as cloud bursting, is modelled differently as the service provider has no control over the underlying infrastructure in the public cloud. Specifically, this means that the service provider cannot decide which node in the public cloud should run a given replica. From the service provider's point of view the public cloud is viewed as an infinitely large pool of resources which can be used to run replicas at a cost. This large pool can be provided by several IaaS providers, and can be composed of data centres in different geographical locations. For the purpose of this work, we do not differentiate between different providers or locations, and hence, we treat the public cloud as a generic infinite pool of resources. The cost of deploying a replica of the pair (i, q) in the public cloud is symbolized as C_{Ciq}, and this cost is dependent on the resource usage of the various service components.

4.3.2. QoS and Replication

As stated, there exists SLAs specifying QoS requirements for each service. A typical QoS requirement related to performance may be to define an upper bound on the average response time or an upper bound on given percentile, e.g. the 95th, of the response time distribution. When discussing the response time of a service, we assume it to be the time from a service receives a service request to the time when the service is ready to provide the response, while ignoring the network latency.

Replication of the service components is performed as a means to achieve a QoS according to the requirements specified in the SLA. Increasing the number of active replicas of a component will increase the amount of resources accessible, and thus, the response time of a service request is assumed to decrease. However, the components are assumed to have a probability of failing, and if a failure occurs, fewer VMs will service the end-user's requests, and hence, the response time might increase. To prevent failures from degrading the performance to a non-satisfactory level, one might deploy passive replicas as they will help reducing the performance degradation stemming from failures and consume fewer resources than active replicas.

Since the amount of resources guaranteed to the active replicas, G_{Aiqg}, is fixed, the response time of a service is only dependent on the number of active and passive replicas of each component in the service. The main motivation behind the fixing of the resources guaranteed to the active replicas is that it reduces the complexity of the decisions related to the replication. When the resource amounts of the replicas are fixed, it is possible to derive a finite set of different ways to replicate a given service outside of the optimisation model. We denote the items in this set as replication patterns, which basically are combinations of active and passive replicas for each component of a given service satisfying the QoS requirement. The number of replicas of a component required for satisfying the QoS requirement is of course depending on what value G_{Aiqg} is fixed to, and at the same time the value chosen also influences the average number of replicas deployed on a node. For operational and failure handling purposes, it might be desirable to not run too many replicas on a single node since a node failure will take down many replicas. On the other hand, having fewer and larger replicas will probably make it harder to utilise the capacity efficiently, and hence, reduce the consolidation effect related to energy-efficiency. So, in the models presented in this paper, we let the user decide on the size of the replicas based on his preferences, and the selected size might vary between components.

An approach for approximating the response time distribution of replicated multi-tier services under the existence of failures is presented by Gullhav et al. (2013). Therein the services are modelled as open queuing networks with failing servers, and these queuing networks are mapped to a continuous time Markov chain (CTMC) using a method called the response time block method (Grottke et al., 2011). The approximation of the service response time corresponds to the absorption time distribution in the CTMC. As such, this approach can be used to analyse whether a given replication pattern satisfy the QoS requirement specified in the SLA, or not.

In this paper, we do not go into detail on how to derive feasible replication patterns, but assume that such patterns are given as input to the models, and in the optimisation process, we select one replication pattern for each service i. Note that the derivation of replication patterns for a given service can be done independently of other services. So for now, we consider a case where there exists a set of replication patterns, $\mathcal{R}_i$, for each service i, and we use the binary variables y_{ir} to indicate if replication pattern r is chosen for service i. Each replication pattern r specifies a number of active and passive replicas, R_{Aiqr} and R_{Piqr}, respectively, for each pair (i, q). Figure 4.1 illustrates a replication pattern for a three-tier service consisting of a web server component, an application logic component and a database component, where the number of active replicas of each component is 2, 1 and 1, respectively; and the number of passive replicas is 1, 1 and 2. At first sight, the number of different replication patterns is vast, but one might be able to say something about dominance. Consider a replication pattern, say r_1, specifying a number of active and passive replicas of each component in a service which is satisfactory in terms of the QoS requirement. Then it is not necessary to include a replication pattern of the same service, say r_2, with more active or passive replicas of a given component without removing active or passive replicas from another component since this would lead to a more expensive solution.

Generally, $G_{Piqg} \leq G_{Aiqg}$ since the passive replicas are not supposed to serve demand in a failure-free situation. Provided that a passive replica is to quickly take over service delivery, it will need to have access to G_{Aiqg} resources if a failure

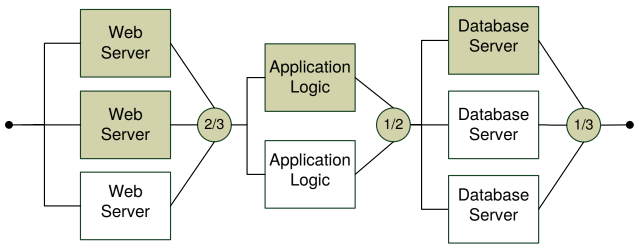

Figure 4.1.: A replication pattern for a three-tier service. The shaded boxes represent active replicas, while the white boxes represent passive replicas.

occurs. If a node is fully loaded with replicas and the passive replicas on the node are only guaranteed access to G_{Piqg} resources, no passive replicas at the node will be able to become active and collaborate in the service delivery when a failure occurs. On the other hand, given that a node is fully loaded and the passive replicas are guaranteed access to G_{Aiqg} resources, all passive replicas on the node can become active at the same time. The second case will be more expensive as more nodes are needed to run the replicas. The first case is less meaningful in our viewpoint since in case of a failure, one must with a high probability migrate a passive replica to a lightly loaded node or possibly start up a new node before letting it become active and serve demand. Our approach takes a direction between these two mentioned cases by ensuring that any passive replica at each node can be activated at a given instant. This is done by guaranteeing that each node running a passive replica has enough free resources such that the passive replica requiring the most resources of type g is able to serve demand if a corresponding active replica fails. Thus, we assume that it is possible to scale up the VMs running on the nodes in the private cloud momentarily. The VMM from Xen, for example, features a mechanism to pause and unpause VMs, which can be used for managing passive replicas in our work. Distler et al. (2011) uses this mechanism in their approach to a cloud-aware fault-tolerant system using passive replicas. However, we acknowledge the occurrence of situations where there are necessary to let more than one passive replica on a given node serve demand. As a means to reduce the probability of such an event, we set an upper bound on the number of passive replicas each node can run, N_P.

Although we are not considering the network latency, we set a bound on the number of different services which can be run from a single node, N_S. By doing this we are implicitly driving different components from the same service to be run on the same node, and thus, the amount of inter-node communication stemming, from the collaboration between components of the same service, is expected to decrease.

4.3.3. Objectives

In the models only considering deployment in the private cloud, the chosen objective is to minimise the number of nodes turned on and running VMs because the nodes consume as much as 70 per cent of the peak power usage in an idle state. Hence, these models have similarities with models of cutting-stock and bin-packing problems. When allowing for deployment in a public cloud as well, the cost of deploying replicas in the public cloud needs to be taken into account. In a model with a long-term perspective, one could have included decisions about infrastructure investments in the private cloud, and weighted the costs of investment and long-term operation to the costs of using public clouds. As our models take a short-term viewpoint, we instead assume that a service provider would utilise the public cloud if and only if there is not enough capacity in the private cloud to run the services. Thus, the objective in the last model consists of minimising the cost of deployment in the public cloud.

4.4. Models of the Service Deployment Problem

This section presents mathematical models of the problem described in Section 4.3, and to ease the understanding of the models all mathematical symbols are summarised in Table 4.1 with a note on where they first appear in Section 4.4. We start by presenting a direct formulation of the deployment problem, and then show how the problem can be formulated in terms of node patterns. Lastly, we extend the reformulated model to allow for deployment in a public cloud in addition to the service provider's private cloud.

4.4.1. Direct Formulation of the Deployment Problem

The direct formulation uses the binary variables w_{iqn} and v_{iqn} to indicate deployment of an active replica and a passive replica of component $q \in \mathcal{Q}_i$ of service $i \in \mathcal{S}$ on node $n \in \mathcal{N}$, respectively. By using these variables, we can formulate the following model.

$$\min z = \sum_{n \in \mathcal{N}} u_n \tag{4.1}$$

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1, \quad \forall i \in \mathcal{S} \tag{4.2}$$

$$\sum_{n \in \mathcal{N}} w_{iqn} - \sum_{r \in \mathcal{R}_i} R_{Aiqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{4.3}$$

$$\sum_{n\in\mathcal{N}} v_{iqn} - \sum_{r\in\mathcal{R}_i} R_{Piqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{4.4}$$

$$w_{iqn} + v_{iqn} - s_{in} \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{4.5}$$

$$\sum_{i\in\mathcal{S}} s_{in} \leq N_S, \quad \forall n \in \mathcal{N} \tag{4.6}$$

$$\sum_{i\in\mathcal{S}} \sum_{q\in\mathcal{Q}_i} v_{iqn} \leq N_P, \quad \forall n \in \mathcal{N} \tag{4.7}$$

$$m_{ng} - (G_{Aiqg} - G_{Piqg}) v_{iqn} \geq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N}, g \in \mathcal{G} \tag{4.8}$$

$$\sum_{i\in\mathcal{S}} \sum_{q\in\mathcal{Q}_i} G_{Aiqg} w_{iqn} + \sum_{i\in\mathcal{S}} \sum_{q\in\mathcal{Q}_i} G_{Piqg} v_{iqn} + m_{ng} - N_{Cng} u_n \leq 0, \ \forall n \in \mathcal{N}, g \in \mathcal{G} \tag{4.9}$$

$$m_{ng} \geq 0, \quad \forall n \in \mathcal{N}, g \in \mathcal{G} \tag{4.10}$$

$$w_{iqn} \in \{0,1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{4.11}$$

$$v_{iqn} \in \{0,1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{4.12}$$

$$s_{in} \in \{0,1\}, \quad \forall i \in \mathcal{S}, n \in \mathcal{N} \tag{4.13}$$

$$u_n \in \{0,1\}, \quad \forall n \in \mathcal{N} \tag{4.14}$$

$$y_{ir} \in \{0,1\}, \quad \forall i \in \mathcal{S}, r \in \mathcal{R}_i \tag{4.15}$$

The objective function (4.1) minimises the number of nodes used for deployment. The equations (4.2) ensure that one replication pattern is selected for each service, and equations (4.3) and (4.4) establish the relation that R_{Aiqr} active and R_{Piqr} passive replicas of the pair (i, q) should be deployed when replication pattern r is selected for service i. The binary variables s_{in} equal 1 if a least one replica of any component of service i is deployed on node n, and so the constraints (4.5) restrict deployment of both an active and a passive replica of the same component on the same node. The inequalities (4.6) and (4.7) set upper bounds, N_S and N_P, on the number of different services and passive replicas deployed on each node. Moreover, the continuous variables m_{ng} denote the amount of resource $g \in \mathcal{G}$ reserved to let any passive replica become active and serve demand at node n. (4.8) sets these variables to be greater than or equal to $(G_{Aiqg} - G_{Piqg})$, i.e. the additional amount of resources required to become active, for the passive replica requiring the most additional resources. The constraints (4.9) define the resource capacities of the nodes, where the first term and second term represents the resources guaranteed to active replicas and passive replicas, respectively, and the third term is the amount

Table 4.1.: Overview of the mathematical symbols in the models and the section where they are used in model for the first time

	Sets	First used
$\mathcal{S}$	Set of services	
$\mathcal{Q}_i$	Set of components of service i	
$\mathcal{N}$	Set of nodes	Sect. 4.4.1
$\mathcal{G}$	Set of resource types	
$\mathcal{R}_i$	Set of replication patterns for service i	
$\mathcal{B}$	Set of node patterns	Sect. 4.4.2
	Parameters	First used
R_{Aiqr}	The number of active replicas of component q of service i in replication pattern r	
R_{Piqr}	The number of passive replicas of component q of service i in replication pattern r	
G_{Aiqg}	The amount of resources of type g guaranteed to an active replica of component q of service i	Sect. 4.4.1
G_{Piqg}	The amount of resources of type g guaranteed to a passive replica of component q of service i	
N_{Cng}	Capacity of resource type g at node n	
N_P	Upper bound on the number of passive replicas deployed on a node	
N_S	Upper bound on the number of different services deployed on a node	
W_{biq}	1, if an active replica of component q of service i is included in node pattern b; 0, otherwise	Sect. 4.4.2
W_{biq}	1, if a passive replica of component q of service i is included in node pattern b; 0, otherwise	
C_{Ciq}	Cost of deploying an active or passive replica of component q of service i in the public cloud	Sect. 4.4.3
N_N	The number of nodes in the private cloud (only binding in the hybrid cloud model)	
	Decision variables	First used
w_{iqn}	1, if an active replica of component q of service i is deployed on node n; 0, otherwise	
v_{iqn}	1, if a passive replica of component q of service i is deployed on node n; 0, otherwise	
s_{in}	1, if at least one replica of service i is deployed on node n; 0, otherwise	
u_n	1, if node n is turned on; 0, otherwise	Sect. 4.4.1
y_{ir}	1, if replication pattern r is chosen for service i; 0, otherwise	
m_{ng}	The amount of resources of type g reserved on node n to allow for activation of a passive replica	
x_b	The number of node patterns of type b selected for deployment	Sect. 4.4.2
w_{Ciq}	The number of active replicas of component q of service i deployed in the public cloud	Sect. 4.4.3
v_{Ciq}	The number of passive replicas of component q of service i deployed in the public cloud	

of resources required for activating at least one passive replica. N_{Cng} denotes node n's capacity of resource g and these constants is multiplied with the variable u_n, so that the node cannot be used for deployment in case it is turned off. Lastly, (4.10) - (4.15) define the variables as continuous or binary.

Since we consider a case where the nodes have identical capacity, a way to reduce the solution space, without deteriorating the objective value, would be to sort the nodes according to whether they are used or not. We have included the constraints (4.16), which force the used nodes to have smaller indices than the unused nodes, when solving the model.

$$u_n - u_{n+1} \geq 0, \quad \forall n \in \mathcal{N} \tag{4.16}$$

4.4.2. Reformulation Based on Node Patterns

Instead of modelling the deployment decisions using the binary variables w_{iqn} and v_{iqn}, one could model these decisions by selection of node patterns. A node pattern is in the context of this work a feasible combination of active and passive replicas of different pairs (i, q) deployed on the same node. A combination of active and passive replicas is feasible if it respects the constraints (4.5) - (4.14) above. In this work, we have chosen to pre-generate feasible node patterns by enumeration, and the enumeration strategy used is elaborated in Section 4.4.4.

We let x_b be integer variables denoting the number of node patterns of type $b \in \mathcal{B}$ selected in the deployment. Note that these variables are general, non-negative integers, and not binary, as there might be optimal to deploy the replicas so that two nodes are identical. Moreover, the parameters W_{biq} indicate whether an active replica of the pair (i, q) is contained in node pattern b. V_{biq} is analogous to W_{biq}, but regards passive replicas instead. When applying a node pattern-based approach the deployment problem can be formulated as follows.

$$\min z = \sum_{b \in \mathcal{B}} x_b \tag{4.17}$$

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1, \quad \forall i \in \mathcal{S} \tag{4.18}$$

$$\sum_{b \in \mathcal{B}} W_{biq} x_b - \sum_{r \in \mathcal{R}_i} R_{Aiqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{4.19}$$

$$\sum_{b \in \mathcal{B}} V_{biq} x_b - \sum_{r \in \mathcal{R}_i} R_{Piqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{4.20}$$

$$x_b \in \mathbb{Z}_+, \quad \forall b \in \mathcal{B} \tag{4.21}$$

$$y_{ir} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, r \in \mathcal{R}_i \tag{4.22}$$

The objective function, (4.17), is still minimising the number of nodes used, but now expressed in terms of the x_b variables. The constraints (4.18) are identical to (4.2), and the equations (4.19) and (4.20) correspond to (4.3) and (4.4). Finally, (4.21) and (4.22) define the x_b and y_{ir} variables as non-negative integers and binary, respectively.

Since each node pattern respects the node-specific constraints (4.5) - (4.14) in the direct formulation, the two formulations are equivalent. If all feasible node patterns, or all maximal node patterns (see Section 4.4.4), are included in $\mathcal{B}$, the formulations will have equal optimal objective function values.

4.4.3. Extending the Reformulated Model to Public Clouds

We are now extending the node pattern-based formulation in order to model a situation where the service provider is allowed to deploy his services in a hybrid cloud, composed of his own private cloud and a public cloud. Deployment in the public cloud is done by leasing VMs from an IaaS provider, and we assume that there exist several different types of VMs with different resource capacities and price. We are only considering a generic public cloud, and only model the decision of how many replicas of (i, q) to deploy in the public cloud, that is, the service provider does not choose between different IaaS providers.

For each replica one can determine which VM type that will be used for deployment in the public cloud, and this is the VM type with the lowest cost and a resource capacity size that guarantees the required resources G_{Aiqg}, i.e. the amount of resources needed for deploying an active replica. This means that both an active and a passive replica of the pair (i, q) will be deployed using the same VM type, and the cost of deploying a replica of (i, q) in the public cloud is denoted C_{Ciq}. The reason for not distinguishing between active and passive replicas in the public cloud is that we assume that the public cloud provider does not support instantaneous scaling of VMs, i.e. the pause/unpause mechanism present in the VMMs in the private cloud.

In the following formulation, we let w_{Ciq} and v_{Ciq} be non-negative integer variables denoting the number of active and passive replicas of (i, q) deployed in the public cloud. In this work, the motivation behind deployment in the public cloud is the occurrence of a situation where the service provider's private cloud does not have enough resources to run the services, that is, the number of nodes available is limited. We use N_N $(= |\mathcal{N}|)$ to symbolise the number of nodes in the private cloud.

$$\min z = \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{Ciq} w_{Ciq} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{Ciq} v_{Ciq} \tag{4.23}$$

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1, \quad \forall i \in \mathcal{S} \tag{4.24}$$

$$\sum_{b \in \mathcal{B}} W_{biq} x_b + w_{Ciq} - \sum_{r \in \mathcal{R}_i} R_{Aiqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{4.25}$$

$$\sum_{b \in \mathcal{B}} V_{biq} x_b + v_{Ciq} - \sum_{r \in \mathcal{R}_i} R_{Piqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{4.26}$$

$$\sum_{b \in \mathcal{B}} x_b \leq N_N \tag{4.27}$$

$$x_b \in \mathbb{Z}_+, \quad \forall b \in \mathcal{B} \tag{4.28}$$

$$w_{Ciq} \in \mathbb{Z}_+, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{4.29}$$

$$v_{Ciq} \in \mathbb{Z}_+, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{4.30}$$

$$y_{ir} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, r \in \mathcal{R}_i \tag{4.31}$$

As explained in Section 4.3.3, when considering deployment in a hybrid cloud, we are minimising the cost of deployment in the public cloud. This is done in (4.23). Again (4.24) is identical to (4.2). In the two sets of constraints (4.25) and (4.26) we have added two terms, w_{Ciq} and v_{Ciq}, in order to allow for deploying active and passive replicas in the public cloud. The inequality (4.27) ensures that the service provider cannot select more node patterns than there are nodes available in the private cloud, and lastly, (4.28) - (4.31) define the variables as either non-negative general integer or binary.

4.4.4. Pre-generation of Node Patterns

Even for relatively small problem sizes, the total number of feasible node patterns is too large for the optimisation software to handle. However, as presented in Section 4.2.2, a typical approach would be to only optimise over a subset of the feasible node patterns. In this work we have applied two measures to limit the number of candidate patterns. Firstly, only maximal node patterns will be selected as candidates. We say that a node pattern is maximal if no further replicas, either passive or active, can be included in the pattern without breaking feasibility according to the constraints (4.5) - (4.9). The number of maximal node patterns will like the total number of node patterns grow exponentially with the problem size, so we also need to reduce the number of candidates in another way. Hence secondly, we include some randomness in the candidate selection, where each maximal node pattern is assigned a score based on a random number and the slack in the constraints (4.9). The number of candidates selected is then controlled by a threshold value. The candidate node patterns are written to disk, and thereafter used as input and read by the optimisation software.

Note that when we do not include all feasible node patterns in the optimisation model, the equalities (4.19) - (4.20) and (4.25) - (4.26) need to be turned into $\geq$-constraints.

4.5. Numerical Results

In this section we present a comparison of the direct formulation and the reformulation based on node patterns. We also investigate the effect of the size of the private cloud, i.e. the number of nodes (N_N), in the hybrid cloud model. Firstly, we will give a brief overview on the setup of the experiments.

4.5.1. Setup of Experiments

In the testing of the models allowing for deployment in the private cloud only, we have used 6 test instances. These test instances vary with respect to the number of different services and the total number of component types, but for all tests presented in this paper, we have only considered one resource type, namely the CPU power. The number of components per service, $|\mathcal{Q}_i|$, is varied between 3 and 5 with an average of 4 for the different services, and we have set N_P to 4 and N_S to 3. In Section 4.5.3, when solving the model allowing for deployment in a public cloud, we have considered 6 different types of VMs, differing in size and price, which can be used for deployment in the public cloud. Table 4.2 gives an overview of the characteristics of the test instances.

Table 4.2.: Overview of the private cloud test instances. The Minimum/maximum number of active replicas on a node refers to the number of active replicas in the maximal packing of a node with the smallest/highest number of active replicas.

Test instance	# services	Total # components types	Minimum/maximum number of active replicas on a node	Avg. # replication patterns per service	Total # maximal node patterns (in 10^6)
P5	5	20	3/6	5.8	0.1
P10	10	40	2/7	7.7	1.7
P20	20	80	2/7	7.7	17.2
P30	30	120	2/7	7.7	54.9
P40	40	160	2/8	7.8	144.3
P50	50	200	2/8	7.7	292.8

It is seen in Table 4.2 that the number of maximal node patterns grows exponentially with the number of services and components, and even for rather small test instances it would not be possible to use all maximal patterns. So, in the models with the node pattern formulation we only optimise over a small fraction of the total number of node patterns, but perform tests with different fractions for each test instance. All node patterns for a given test instance are generated based on the same seed in the random number generation, but with different acceptance thresholds. Since every feasible pattern is found in the same order between different runs of the pre-generation, an increase in the threshold value will generate the same node patterns as with a lower threshold in addition to some new node patterns.

Regarding the distribution of the size of the active replicas, we have expressed this in Table 4.2 as the minimum and maximum numbers of active replicas one can have in a maximal packing of a node. In the largest test instances, one can see that it is possible to have up to eight active replicas on a node. However, the size of the active replicas is also so that it is possible to construct a maximal packing of a node with only one active replica and four passive replicas, which is counted as two active replicas since there is reserved enough capacity to immediately activate a passive replica.

All models are solved using the optimisation software XpressMP 7.4.0, and the tests are run on a CentOS 5.8 machine with a dual core 3.0GHz Intel E5472 Xeon processor and 16 GB of memory. The optimisation software is able to utilise eight threads, and all optimisation runs have a runtime limit of 5 hours.

4.5.2. The Direct Formulation Versus the Reformulation

The uppermost part of Table 4.3 shows the performance of the direct formulation on the six test instances. We can see that we are not able to prove optimality in any of the test instances within the runtime limit, and the gap between the best solution and best bound is quite large. The results of the reformulation, shown in the lower part of the table, list the results of the test instances three times (except for P5), using different fractions of node patterns. For readability we have added an 'a', 'b' and 'c' to the test instance name, with increasing fraction of node patterns included. Because of the large amount of maximal node patterns in the larger test instances, the number of node patterns used in the optimisation is a small fraction of the total number.

For the P5 instance we are able to optimise using all maximal node patterns, and hence, the solution found is optimal. We also notice that the direct formulation is capable of finding this solution, although not being able to prove the optimum. For all the other test instances, we see that the reformulated model produces better solutions than the direct model, even when using a small fraction of the node patterns. In fact, optimising over a small amount of node patterns produces equally good or better solutions than optimising over a large amount. Even though we are not able to prove optimality or find the optimal solution within the maximum run time of 5 hours, it is seen that the reformulation produces fairly good solutions early. The column showing the time to find the first solution proven to be within 10 per cent of the best bound displays that good solutions are found during the first 2 to 3 minutes of optimisation.

It is difficult to compare the best bounds of the direct formulation and the reformulation since the reformulation is based on a heuristic sample of the maximal node patterns. Keep in mind that the best bound is dependent on the node

Table 4.3.: Direct formulation vs. reformulation. The solution times (in seconds) only accounts for the time spent optimising, i.e. not pre-generating columns in the reformulated model. For the reformulation the right-most column indicates the number of node patterns included in percentage of the number of maximal node patterns for each test instance (cf. Table 4.2).

	Test Instance	Best objective value	Best bound	Best sol. found in root node?	Time to first sol. within 10% of best bound**	Solution time	# Node patterns included
Direct Formulation	P5	15	14	No	7	7*	N/A
	P10	37	29	Yes	N/A	2*	
	P20	74	56	Yes	N/A	25*	
	P30	116	85	Yes	N/A	207*	
	P40	156	114	Yes	N/A	1234*	
	P50	193	141	Yes	N/A	2730*	
Reformulation	P5	15	15	Yes	30	30	100%
	P10a	33	33	No	10	382	9.34%
	P10b	33	33	No	22	500	18.70%
	P10c	33	33	No	60	1611	37.37%
	P20a	66	65	No	20	2110*	0.933%
	P20b	66	65	No	38	5677*	1.87%
	P20c	66	65	No	142	10401*	4.68%
	P30a	103	101	No	47	5384*	0.466%
	P30b	103	101	No	109	10742*	0.935%
	P30c	106	100	Yes	262	262*	1.87%
	P40a	139	136	No	69	12950*	0.187%
	P40b	143	136	Yes	161	1861*	0.374%
	P40c	143	135	Yes	499	499*	0.936%
	P50a	173	168	No	88	15396*	0.0938%
	P50b	179	168	Yes	202	202*	0.187%
	P50c	177	167	Yes	596	1066*	0.468%

* Optimal solution not found within 5 hours. Time until best solution found is reported
** This is the time when the particular solution was found, not when it was proved to be within 10 % of the best bound. Note that in the reformulation the best bound depends on the node patterns included

patterns included in the optimisation. But, when optimising over all maximal node patterns in P10, we are able to prove, after about 1000 branches (380 seconds), that the best bound is still 33. So, we can conclude that 33 is the optimal objective value in the P10 case.

Regarding the hardness of solving the test instances, the direct formulation finds the best solution (not optimal) in the root node of the branch and bound tree, except for the smallest instance. That is, the direct formulation is not capable of finding better solutions during the branching. For the reformulation, we observe the same for P30c, P40b, P40c, P50b and P50c, which is due to the large amount of node patterns included. This helps us explaining why optimising over less node patterns gives better solutions.

4.5.3. The Effect of the Size of the Private Cloud

Now, studying the results of the hybrid cloud model, we are focusing on the relation between the size of the private cloud and the amount of capacity required to run the services. We are first considering the test instance with 10 services, and set a limit on the number of nodes in private cloud. In Table 4.3, we saw that the minimum number of nodes required to run the services was 33, and hence, if we set N_N to 33, all service would be run in the private cloud. However, if N_N lies in the interval $[1, \ldots, 32]$, we need to distribute the replicas between the private cloud and the public cloud.

Table 4.4 shows what effect the value of N_N has on the objective value and the hardness of the problem for a case corresponding to P10a (now labelled H10a). In the table, we have also included the trivial case without a private cloud (i.e. 0 nodes), where all replicas need to be deployed in the public cloud. Naturally, the objective value will decrease with an increase in N_N since the objective function only accounts for the cost of deployment in the public cloud, and the objective value reaches a value of 0 when we have 33 nodes in the private cloud.

Table 4.4.: H10a: The effect of the number of nodes in the private cloud on the cost and solution time

# Nodes in private cloud (N_N)	Cost of active replicas in public cloud	Cost of passive replicas in public cloud	Objective value	Best bound	Solution time (sec)
0	7230	7485	14715	14715	1
10	4365	1725	6090	6090	17
15	3375	795	4170	4170	45
20	2415	195	2610	2610	213
25	1410	0	1410	1410	1752
30	405	0	405	405	2018
33	0	0	0	0	484

However, we also observe that the cost of passive replicas decrease more rapidly than the cost of active replicas, and when $N_N = 25$, no passive replicas run in the public cloud. This effect can be explained by the difference in resources guaranteed to passive replicas in the private and public cloud. As explained in Section 4.4.3, when deployed in the public cloud, a passive replica is deployed in the same VM type as a corresponding active replica, even though it requires fewer resources in a failure-free situation. So, when increasing N_N, we would like to move the passive replicas to the private cloud first, until the upper bound on the number of passive replicas on a node is reached or there are no more passive replicas deployed in the public cloud.

Concerning the difficulty of solving the problem, we see that as long as the number of nodes in the private cloud is relatively low, the model is solved in relatively short time. When the number of nodes is increased the model is harder to solve, that is, the solution time increases. However, note that if we set the number of nodes in the private cloud to 33, the problem is seemingly a bit easier to solve.

When studying H50a, corresponding to P50a, we also observe that the problem gets harder as N_N approaches the value where no replicas are deployed in the public cloud. This is seen in Table 4.5. When N_N increases from 50 to 100, we are not able to prove the optimum within 5 hours of runtime, and when increasing N_N further to 150, the optimality gap gets quite large. Moreover, the same effect on the cost of passive replicas as in Table 4.4 is seen; the cost of passive replicas decreases faster than the cost of active replicas.

Table 4.5.: H50a: The effect of the number of nodes in the private cloud on the cost and solution time

# Nodes in private cloud (N_N)	Cost of active replicas in public cloud	Cost of passive replicas in public cloud	Objective value	Best bound	Solution time (sec)
0	37545	38565	76110	76110	1
25	28215	15765	43980	43980	176
50	22635	9465	32100	32100	1938
100	13125	1680	14805	14540	14507*
150	3675	90	3765	2960	13614*

* Optimal solution not found within 5 hours. Time until best solution found is reported

4.6. Conclusions

In this paper, we have developed three different MILP models considering a service provider's decisions related to the deployment and QoS management of his services. To our knowledge, including fault-tolerance techniques in resource allocation and deployment models is new and a novel feature of our work. The two first models considered only deployment in the private cloud of the service provider, while the last model allowed for deployment in a public cloud as well. We utilised pre-generated node patterns in the construction of our reformulated models. Since the number of feasible node patterns grows exponentially with the problem size, we heuristically selected a small sample of the node patterns as candidates in the optimisation models.

The performance of the models where tested on several test instances of different size. We compared the two models concerning deployment only in a private cloud. In all test instances, expect for the smallest instance, the reformulated model produced solutions of higher quality despite that we only considered a small fraction of the total number of feasible node patterns. For the larger test instances we were not able to prove or find the optimal solution using the reformulation, but we noted that high-quality solutions are found early in the optimisation process.

Regarding the model allowing for deployment in a public cloud, we observed that the hardness of the problem grew with the size of the private cloud relative to the capacity required for running all the services, until the size of the private cloud was large enough to run all services. Furthermore, it is shown to be preferable to run all passive replicas in the private cloud if possible.

Acknowledgements

The authors want to thank Telenor's Research and Future Studies Department (Norway) for fruitful discussions and validation of the models, in addition to economical support.

Bibliography

D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. Energy-aware autonomic resource allocation in multitier virtualized environments. *IEEE Transactions on Services Computing*, 5(1):2–19, 2012.

A. Beloglazov, R. Buyya, Y. C. Lee, and A. Y. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, 82(2):47–111, 2011.

D. Breitgand and A. Epstein. SLA-aware placement of multi-virtual machine elastic services in compute clouds. In N. Agoulmine, C. Bartolini, T. Pfeifer, and D. O'Sullivan, editors, *Integrated Network Management*, pages 161–168. IEEE, 2011.

H. Cambazard, D. Mehta, B. O'Sullivan, and H. Simonis. Bin packing with linear usage costs – an application to energy management in data centres. In C. Schulte, editor, *Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*, pages 47–62. Springer Berlin Heidelberg, 2013.

T. Distler, R. Kapitza, I. Popov, H. P. Reiser, and W. Schröder-Preikschat. SPARE: Replicas on hold. In *Proceedings of the 18th Network and Distributed System Security Symposium*, Geneva, Switzerland, 2011. The Internet Society.

M. Eley. A bottleneck assignment approach to the multiple container loading problem. *OR Spectrum*, 25(1):45–60, 2003.

K. Fagerholt. Ship scheduling with soft time windows: An optimisation based approach. *European Journal of Operational Research*, 131(3):559–571, 2001.

K. Fagerholt and M. Christiansen. A combined ship scheduling and allocation problem. *The Journal of the Operational Research Society*, 51(7):834–842, 2000.

T. C. Ferreto, M. A. Netto, R. N. Calheiros, and C. A. D. Rose. Server consolidation with migration control for virtualized data centers. *Future Generation Computer Systems*, 27(8):1027 – 1034, 2011.

H. Goudarzi and M. Pedram. Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 324–331, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

H. Goudarzi and M. Pedram. Energy-efficient virtual machine replication and placement in a cloud computing system. In *2012 IEEE 5th International Conference on Cloud Computing*, pages 750–757, Los Alamitos, CA, USA, 2012. IEEE Computer Society.

M. Grottke, V. Apte, K. Trivedi, and S. Woolet. Response time distributions in networks of queues. In R. J. Boucherie and N. M. van Dijk, editors, *Queueing Networks*, volume 154 of *International Series in Operations Research & Management Science*, pages 587–641. Springer US, New York, USA, 2011.

A. N. Gullhav, B. Nygreen, and P. E. Heegaard. Approximating the response time distribution of fault-tolerant multi-tier cloud services. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 287–291, Los Alamitos, CA, USA, 2013. IEEE Computer Society.

F. Hennig, B. Nygreen, M. Christiansen, K. Fagerholt, K. C. Furman, J. Song, G. R. Kocis, and P. H. Warrick. Maritime crude oil transportation - a split pickup and split delivery problem. *European Journal of Operational Research*, 218(3):764 – 774, 2012.

F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. L. Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 41–50, New York, NY, USA, 2009. ACM.

I. Iyoob, E. Zarifoglu, and A. B. Dieker. Cloud computing operations research. *Service Science*, 5(2):88–101, 2013.

V. Petrucci, O. Loques, and D. Mossé. A dynamic optimization model for power and performance management of virtualized clusters. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 225–233, New York, NY, USA, 2010. ACM.

L. Rao, X. Liu, M. Ilic, and J. Liu. MEC-IDC: joint load balancing and power control for distributed Internet Data Centers. In *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, pages 188–197, New York, NY, USA, 2010. ACM.

B. Speitkamp and M. Bichler. A mathematical programming approach for server consolidation problems in virtualized data centers. *IEEE Transactions on Services Computing*, 3(4):266–278, 2010.

J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Generation Computer Systems*, 28(2):358 – 367, 2012.

R. Van den Bossche, K. Vanmechelen, and J. Broeckhove. Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 228–235, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

Paper IV

Anders N. Gullhav and Bjørn Nygreen:

A Branch and Price Approach for Deployment of Multi-tier Software Services in Clouds

submitted to an international journal

A Branch and Price Approach for Deployment of Multi-tier Software Services in Clouds

Abstract:
This paper considers a combined service placement and replication decision problem in a cloud computing context. The services are composed of multiple tiers which are to be placed on nodes in the private cloud of the service provider or, if the private cloud has limited capacity, partly in a public cloud. In the service delivery, the provider has to take into account the quality of service guarantees offered to his end-users. To solve the problem, we develop a branch and price algorithm, where the subproblems both are formulated as a linear mixed integer program and a shortest path problem with resource constraints (SPPRC), which underlying network has a special structure. The SPPRC can solved by an exact label-setting algorithm, but to speed up the solution process, we develop a heuristic label-setting algorithm based a reduced network and simplified dominance rule. Our results show that using the heuristic subproblem solver is efficient. Furthermore, the branch and price algorithm performs better than a previously developed pre-generation algorithm for the same problem. Furthermore, we analyze and discuss the differences in solutions that utilize resources in a public cloud to different degrees. By conducting this analysis, we are able to identify some essential characteristics of good solutions.

5.1. Introduction

In this work, we are considering a service deployment problem of a software-as-a-service (SaaS) provider that provides a set of services to his end-users. In the provisioning, the SaaS provider (SP) must scale his services according to the performance and availability guarantees specified in the service level agreements (SLAs), contracts that define the services in terms of functionality and quality. Furthermore, the SP also has to decide where to run the services. A typical objective in this problem is to minimize the cost of provision, while fulfilling the service quality guarantees.

In principle, this decision problem can be solved statically or dynamically. Herein, we consider the demand to vary over time, but within certain periods, the demand is stationary in a stochastic sense. When these periods are sufficiently long, that is, they might reflect working hours, evenings, etc., it is possible to compute a stationary deployment solution for each period, and apply the appropriate solution when one enters a new period. If the infrastructure running the services are failure-prone, it is necessary to complement the stationary solution with a strategy to return from a failed state back to the stationary solution. This strategy is found by solving a dynamic problem of another type, which is out of the scope of this paper. However, in cases were the demand is constantly fluctuating and not in a stationary state, it is necessary to solve the former problem dynamically, but such cases are not considered here.

The SaaS services offered by the SP are modeled as multi-tier services, which are services composed of several components collaborating to provide the service to the end-users. An example of a multi-tier service is a three-tier web service composed of a web server tier, an application tier and a database tier. Each of these tiers are often run in several virtual machines (VMs), which in turn run on servers. The SP in focus own and operate a limited set of servers, forming a private cloud, which are capable of running the VMs of the service tiers. An important operational cost component in a data center is the cost of energy, and a strategy used to minimize the cost of energy usage in the VM scheduling is to turn off servers that are not required with the current demand (Iyoob et al., 2013). For the SP, there might be periods were the service demand is too high to be able to provide the services from the private cloud alone. In such cases the SP has the option to lease resources from a public infrastructure-as-a-service (IaaS) provider (e.g. Amazon (Amazon Web Services, 2015)), denoted a public cloud provider. When the infrastructure used by the SP to provide his services is composed of both a private and a public cloud, this infrastructure is referred to as a hybrid cloud (Mell and Grance, 2011). Using a public cloud for permanent service provisioning is costly, and it is often desirable to utilize its own servers fully before leasing capacity from an IaaS provider.

The private and public clouds typically consists of a large amount of cheap servers, hard disks, routers, etc., which make the services prone to failures, and hence make fault tolerance an important consideration in the service deployment. Considering fault tolerance management in cloud systems, there exists several works that present conceptual frameworks and software systems utilizing redundant VMs (Cully et al., 2008; Distler et al., 2011; Jhawar et al., 2013). Cully et al. (2008) describe an approach where a replication engine at each host propagates the state of active VMs to another host, which holds backup VMs ready to execute if an active VM fails. In Distler et al. (2011), an active-passive replication scheme is used to achieve fault tolerance. Therein backup VMs, denoted passive replicas, are put into a paused state, from where they can be unpaused and activated rapidly. Jhawar et al. (2013) propose a framework that aims to facilitate the offering of fault-tolerance as a service to existing applications. The framework supports both active-active and active-passive replication.

In Gullhav and Nygreen (2015), we present a novel optimization model of the service deployment problem of the SP that models decisions both related to the replication of the tiers of multi-tier services and related to the placement of the replicas. In the model, each tier of a service could be replicated into a number of load-balanced replicas, referred to as *active replicas*, and in addition, *passive replicas* are used to improve the fault tolerance of the tier. However, since we are interested in the performance and fault tolerance of the whole service, not only the tiers, the selection of replication levels of the different tiers of a service are linked. Moreover, we argue that in cases where different services interact through their placement (e.g. by running on the same servers), the most cost-efficient way to replicate the tiers of a given service is dependent of the replication of the tiers of other services. This is reflected in the model.

This service deployment problem was modeled as a linear mixed-integer program (MIP), and solved using a commercial MIP solver in Gullhav and Nygreen (2015). We also reformulated the problem and obtained a pattern-based formulation. The linear relaxation of the reformulation was shown to be much stronger than that of the former, *direct MIP formulation*. Nevertheless, the number of variables in the reformulation grew exponentially with the size of the problem, and we could only optimize over a small number of the variables. Since we seek to find a stationary solution, we argue that one can spend some time searching for a near-optimal or optimal solution. If the solution quality is of more importance than the time to find a solution, we suggest using an exact solution method. Here, we propose a branch and price (B&P) algorithm, which master problem is based on the mentioned pattern-based formulation. The subproblem of the B&P is solved using a MIP solver and a label-setting algorithm. The latter seeks to find the shortest path in a network, which to our knowledge has a novel structure. While we develop an exact label-setting algorithm, we early observe that this algorithm has deficiencies related to its dominance rule. A contribution of this paper is an efficient heuristic label-setting algorithm based on a reduced network and simplified dominance rule. Using this heuristic in conjunction with an exact MIP solver speeds up the B&P algorithm. However, in some nodes of the enumeration tree, no improving columns can be found, and hence, using the heuristic algorithm is ineffective. Another contribution of this paper is a simple rule to decide whether the heuristic algorithm should be used in a node, or the exact MIP solver should be called directly. A major question we seek to answer in our computational study is by which methods the subproblem should be solved. The paper also provides a discussion on how the size of the private cloud relative to the service's resource requirements affect the solution structure.

The outline of the paper is as follows. Next, in Section 5.2, we present some works related to the service deployment problem, and in Section 5.3, we describe the problem in more detail. Two variants of the problem are formulated in Section 5.4, before the B&P algorithm is explained in Section 5.5. More details of the algorithm are found in 5.7. The numerical results of our experiments with the algorithm, along with a discussion of the results, are presented in Section 5.6. Finally, we conclude the paper in Section 5.7.

5.2. Related Work

The part of the problem that regards the replication level decisions is related to the redundancy allocation problem (Kuo and Wan, 2007). The redundancy allocation problem consists of allocating parallel components to different subsystems in series with an objective of minimizing the cost, such that the reliability is better than a threshold. Ashrafi et al. (1994) present an application of the redundancy allocation problem where the goal is to optimize the reliability of a software system. More recently, Meedeniya et al. (2010) use multi-objective optimization to explore the trade-off between reliability and energy consumption when building redundancy into an embedded system.

Regarding the part of the problem that relates to the placement of the VMs, there exists a lot of literature on scheduling and placement of VMs in clouds. A recent survey on resource management in clouds is found in Jennings and Stadler (2015). We also review some literature in Gullhav and Nygreen (2015). Nevertheless, a short review follows. Speitkamp and Bichler (2010) presents static optimization models for the problem of placing VMs on servers in enterprise data centers. In contrast to our work, they do not model services as consisting of multiple tiers, and neither do they treat replication of VMs. Dynamic optimization models which investigate the trade-off between energy usage and performance can be found in several works (Petrucci et al., 2010; Ferreto et al., 2011; Ardagna et al., 2012). Google proposed another dynamic optimization model in the ROADEF/EURO challenge 2012 (ROADEF, 2012). Breitgand and Epstein (2011) use column generation to solve a static service placement problem, both with and without considering the cost of migrating VMs. Like us, they require the provider to comply with requirements specified in SLAs, but while we consider the service deployment problem of a SaaS provider, they regard the decision problem of an IaaS provider. Furthermore, Goudarzi and Pedram (2011) and Ardagna et al. (2012) regard a dynamic placement problem for multi-tier services with decisions related to the load balancing and bounds on the performance. However, except our previous work, none of the reviewed placement literature so far models the placement of passive backup replicas. On the other hand, Bin et al. (2011) consider a related placement problem of an IaaS provider where some VMs require one or more backup locations to which they can be migrated in case of a failure.

5.3. Problem Description

We let $\mathcal{S}$ be the set of multi-tier services offered by the SP, and let $\mathcal{Q}_i$, $i \in \mathcal{S}$, be the set of *components*, i.e., tiers, of service i. A component $q \in \mathcal{Q}_i$ runs in a VM, which in turn should run on a server, denoted a *node* in the following. A private cloud placement is an assignment of component q of service i to a node in the private cloud of the SP. The nodes provide resources like CPU power, memory and storage to the VMs, and we let the set $\mathcal{G}$ contain the different types of resources. We assume the nodes to be identical with equal resource capacities N_{Cg} of resource type $g \in \mathcal{G}$. The VMs running in the private cloud are of given sizes, and we let G_{Aiqg} be the resources of type g assigned to component q of service i when running on a node. For simplicity, we will denote component q of service i as the pair (i, q) in the following.

5.3.1. Quality of Service

The QoS levels of the services are stated in SLAs, and common QoS measures for web services include response time, either average or a percentile of the distribution, throughput and downtime (Menasce, 2002). Replication is used to obtain the required QoS, and the SP might choose to replicate a service component into multiple active load-balanced replicas to increase the performance, which means that identical service components can be run in several identical VMs. To make the services fault-tolerant, the SP might also place additional passive back-up replicas on the nodes. The replicas of the same service component should not be placed on the same node, a requirement referred to as node-disjoint placement. This is to prevent that multiple replicas of a single component go down due to a single node failure. The passive replicas does not service demand in a failure-free situation, and consume fewer resources than active replicas (Distler et al., 2011). Therefore, instead of being assigned G_{Aiqg} resources, a passive replica of the pair (i, q) is assigned $G_{Piqg} (< G_{Aiqg})$ resource of type g when placed on a node. Each node that runs at least one passive replica needs to maintain a pool of shared backup resources for activation of passive replicas. The size of this pool is a trade-off between fault tolerance and resource utilization since a small pool size will make it difficult to provide enough resources to a passive replica when it is activated; and a large pool means that a large amount of resources are unused in a failure-free situation. Herein, the pool size is set such that the passive replica requiring the largest increase in resource when being activated can be activated. This means that a node can only guarantee the activation of one passive replica at a time, and thus we will limit the number of passive replicas on a node to N_P, so that the passive replicas are spread on the infrastructure.

Herein, we will not consider a specific QoS measure, but instead assume that there exists a method to check if a service, with given replication levels of its components, satisfies the QoS guarantees. Gullhav et al. (2013) present a method that takes the number of active and passive replicas of each tier and the assigned resources of the replicas as input, and outputs an approximation of the response time distribution of the service. In Gullhav and Nygreen (2015), we introduced *replication patterns* in order to express the relationship between the number of replicas of each component and the QoS guarantees in a linear MIP. Thus, a replication pattern $r \in \mathcal{R}_i$ of service i is a combination of the number of active and passive replicas of each component in the service. We let these numbers be denoted by R_{Aiqr} and R_{Piqr}, which means that if replication pattern r is chosen for service i, one has to deploy R_{Aiqr} active replicas and R_{Piqr} passive replicas of each component $q \in \mathcal{Q}_i$. We assume that the sets of replication patterns $\mathcal{R}_i$ for each service i are given as input, and that all the replication patterns in these sets satisfy the QoS guarantees.

The performance of the underlying network is neglected in our models. Within a single data center, neglecting the network latency might be fair, but between clouds of different geographical locations, one might argue that the network latency plays a role. On the other hand, this negligence simplifies the models considerably. However, there is put a bound on the number of different services that can be run from a single node, N_S. With this constraint, the model will implicitly drive different components from the same service to be run on the same nodes, and so, the amount of inter-node communication, stemming from the collaboration between components of the same service, is expected to decrease.

5.3.2. Placement in Public Clouds

So far, we have taken for granted that the private cloud has enough resources to run all the services, and only discussed the placement of the replicas in the private cloud of the SP. Now, we consider the case that the private cloud does not have enough nodes to run all services, and the SP gets the option, or are rather forced, to lease some extra resources from a public cloud provider. We let N_N denote the number of nodes in the private cloud. In this case, leasing means to run a replica of a pair (i, q) in a VM of an appropriate size in the data centers of the public cloud provider at a cost. We do not assume that the public cloud provider supports the activation of passive replicas, so passive replicas are always run in the private cloud. Since we do not consider network effects explicitly in this work, we can consider the public cloud resources as a generic resource pool, possibly consisting of offers from several public cloud providers. We can also in advance of the optimization determine the cheapest way (at which provider, and in what geographical location, etc.) to place an active replica of the pair (i, q) in the public cloud. The cost of this placement is denoted C_{Ciq}, and it is dependent on the resource requirement of the active replica. Generally, the public cloud providers offer predetermined, fixed types of VMs differing by resource capacity and cost, and typically the resource capacities of the VM types are coarse grained. For example, Amazon (Amazon Web Services, 2015) offers seven general-purpose VM types, where the resource capacities doubles from one VM type to the next. The amount of resource assigned to an active replica of pair (i, q), G_{Aiqg}, is fixed without considering the available VM types offered in the public cloud. When selecting the VM type that is going to be used for placement of an active replica in the public cloud, one has to select a VM type that provides enough resources for each $g \in \mathcal{G}$, and will of course select the cheapest VM type providing this. This means that two pairs (i_1, q_1) and (i_2, q_2) with $G_{Ai_1q_1g} \neq G_{Ai_2q_2g}$ for all $g \in \mathcal{G}$, might have to run in the same type of VM in the public cloud at an identical cost.

5.3.3. Cost Minimization

When only considering deployment in the private cloud of the SP, we consider the cost of energy usage as the sole cost component. Like, several other optimization models for VM placement, we minimize the number of nodes that are turned on. A reason for this choice is that a node that is turned on, but in an idle state, consume as much as 70 per cent of its peak power (Beloglazov et al., 2011), and hence, running the services on as few nodes as possible will reduce the cost substantially. Another situation arises when a hybrid cloud is used for placement. The motivation behind this scenario is that the private cloud does not have enough resources for running all services, and thus the SP has to use a public cloud. In this scenario, the meaningful cost-minimizing objective is to minimize the cost of placing VMs in the public cloud, while fully utilizing the nodes in the private cloud.

5.4. Mathematical Formulations

Two formulations of the service deployment problem are presented next. The first formulates the *private cloud model*, where only placement in the private cloud is considered, while the second formulates the *hybrid cloud model* that also allows for placement in a public cloud. Both formulations were proposed in Gullhav and Nygreen (2015), but are repeated here as they form the master problem of our B&P algorithm. To model the placement of replicas, they use *node patterns*, which represent a feasible assignment of replicas to a node, that is, the assignment must respect the resource capacities of the node, the requirement for shared backup resources, the requirement for node-disjoint placement of replicas, and upper bounds on the number of passive replicas and the number of services on a node. Since the nodes are considered identical, any node pattern can represent any node in the private cloud. In this section, we take the set of node patterns as granted, but in Section 5.5, we will formulate a subproblem for the B&P, which is used to generate node patterns dynamically.

5.4.1. Private Cloud Model

In the formulation below, the integer variables x_b denote the number of times each node pattern $b \in \mathcal{B}$ is used in the solution, where $\mathcal{B}$ represents the set of node patterns. The parameters $W_{biq} \in \{0, 1\}$ and $V_{biq} \in \{0, 1\}$ indicate the placement of an active replica ($W_{biq} = 1$) and a passive replica ($V_{biq} = 1$) of the pair (i, q) in node pattern b. Furthermore, the binary variables y_{ir} indicate the selection of a replication pattern $r \in \mathcal{R}_i$ for service i.

$$\min z_P = \sum_{b \in \mathcal{B}} x_b \tag{5.1}$$

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1 \quad \forall i \in \mathcal{S} \tag{5.2}$$

$$\sum_{b \in \mathcal{B}} W_{biq} x_b - \sum_{r \in \mathcal{R}_i} R_{Aiqr} y_{ir} = 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{5.3}$$

$$\sum_{b \in \mathcal{B}} V_{biq} x_b - \sum_{r \in \mathcal{R}_i} R_{Piqr} y_{ir} = 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{5.4}$$

$$x_b \in \mathrm{Z}_+ \quad \forall b \in \mathcal{B} \tag{5.5}$$

$$y_{ir} \in \{0, 1\} \quad \forall i \in \mathcal{S}, \forall r \in \mathcal{R}_i \tag{5.6}$$

The objective function (5.1) minimizes the number of used node patterns, and thereby also nodes. The equalities (5.2) ensure that one replication pattern is selected for each service. Furthermore, the equalities (5.3) and (5.4) establish the relation between the node pattern variables and the replication pattern variables, and so provide that the correct number of active and passive replicas of each pair (i, q) are placed on the nodes, according the chosen replication pattern. Finally, (5.5) and (5.6) define the node pattern and replication pattern variables as non-negative integer and binary, respectively.

5.4.2. Hybrid Cloud Model

The motivational case of placing replicas in a public cloud is that there exist time periods where the SP does not have enough capacity to run all services in his private cloud. We let w_{Ciq} be integer variables denoting the number of active replicas of (i, q) placed in the public cloud. The mathematical formulation of the hybrid cloud model is given below.

$$\min z_H = \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{Ciq} w_{Ciq} \tag{5.7}$$

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1 \quad \forall i \in \mathcal{S} \tag{5.8}$$

$$\sum_{b \in \mathcal{B}} W_{biq} x_b + w_{Ciq} - \sum_{r \in \mathcal{R}_i} R_{Aiqr} y_{ir} = 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{5.9}$$

$$\sum_{b \in \mathcal{B}} V_{biq} x_b - \sum_{r \in \mathcal{R}_i} R_{Piqr} y_{ir} = 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{5.10}$$

$$\sum_{b \in \mathcal{B}} x_b \leq N_N \tag{5.11}$$

$$w_{Ciq} \in \mathrm{Z}_+ \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{5.12}$$

$$x_b \in \mathrm{Z}_+ \quad \forall b \in \mathcal{B} \tag{5.13}$$

$$y_{ir} \in \{0, 1\} \quad \forall i \in \mathcal{S}, \forall r \in \mathcal{R}_i \tag{5.14}$$

The objective, (5.7), minimizes the cost of placing active replicas in the public cloud. The equalities (5.8) correspond to (5.2) in the private cloud model. The balance equalities (5.9) now include a term for the public cloud placements of active replicas, while (5.10) remains identical to (5.4). The inequality (5.11) reflects that the number of nodes are limited by seting an upper bound on the sum of node patterns. At optimum (5.11) will be satisfied as an equality. At last, (5.12) - (5.14) define the decision variables as integer or binary. It should be noted that the w_{Ciq} variables are naturally integer as long as all x_b and y_{ir} variables are non-fractional.

5.5. The Branch and Price Approach

In this section, we will describe the algorithmic features of our B&P approach, and especially concentrate on the solution methods used for solving the subproblem. In short, B&P is a solution method that uses column generation in a branch and bound (B&B) framework. For the models of Section 5.4, it would be possible in small cases to include all feasible node patterns and solve the models as integer programs (IPs) using B&B. However, for realistic cases, this is not practical, and we propose to solve the model using B&P. In B&P, one generates new node patterns in each B&B node until no further profitable node patterns exist. This is done by alternating between solving the master problem, i.e., the formulations in Section 5.4, as a linear program (LP), and a subproblem for generating new node patterns, until no node patterns with negative reduced cost exists. The subproblem minimizes the reduced cost of a new node pattern by using the dual variables of the master problem as coefficients in the objective function. When no node patterns with negative reduced cost is found, one finishes the current B&B node and uses the same criteria for branching or pruning as in the traditional B&B. However, the branching rules used in B&P applications often differ from the typical branching rule in the B&B framework. The branching rule used herein is discussed in Section 5.5.3. Since the master problem LP only contains a subset of the feasible node patterns, this problem is referred to as the restricted master problem (RMP), and this is solved using a commercial LP solver. To obtain integer solutions by other means than branching, one can solve the RMP as an IP at different points in time. The results of the experiments presented in Section 5.6 show that we find all our best integer solutions this way. This strategy is also used in the literature, e.g., by Gunnerud et al. (2012), which find all their integer solutions by solving an IP.

Algorithm 5.1 Pseudocode of the branch and price algorithm

```
Require: an initial set of node patterns, $\mathcal{B}$, such that feasibility is assured.
 1: Initialize tree by creating a root node
 2: $z^I \leftarrow \infty$ // Initialize the objective value of the current incumbent to $\infty$
 3: while there exists unsolved nodes do
 4:     Get next unsolved B&B node accoring to the best first strategy
 5:     repeat
 6:         $(z, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \eta) \leftarrow$ solveRMP($\mathcal{B}$)
 7:         $(\zeta, \boldsymbol{w}, \boldsymbol{v}) \leftarrow$ solveSubproblem($\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \eta$)
 8:         if $\zeta < 0$ then
 9:             $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\boldsymbol{w}, \boldsymbol{v})\}$ //add the new node pattern, represented by $(\boldsymbol{w}, \boldsymbol{v})$, to $\mathcal{B}$
10:         end if
11:     until $\zeta \geq 0$ //implemented as check against a small positive number
12:     if $z < z^I$ then
13:         if solution $(\boldsymbol{x}, \boldsymbol{y})$ is fractional then
14:             Branch and add new nodes with parent objective value $z$ to the set of unsolved nodes
15:         else
16:             $z^I \leftarrow z$, $(\boldsymbol{x}^I, \boldsymbol{y}^I) \leftarrow (\boldsymbol{x}, \boldsymbol{y})$ // store the new best solution
17:             Remove all unsolved nodes with parent objective value worse than $z^I$ (prune)
18:         end if
19:     end if
20:     if /*any condition*/ then {// specified in Section 5.6.1}
21:         $(z, \boldsymbol{x}, \boldsymbol{y}) \leftarrow$ solveMasterIP($\mathcal{B}$) //Solve the RMP as an IP
22:         if $z < z^I$ then
23:             $z^I \leftarrow z$, $(\boldsymbol{x}^I, \boldsymbol{y}^I) \leftarrow (\boldsymbol{x}, \boldsymbol{y})$ // store the new best solution
24:             Remove all unsolved nodes with parent objective value worse than $z^I$ (prune)
25:         end if
26:     end if
27: end while
28: return $(\boldsymbol{x}^I, \boldsymbol{y}^I)$
```

An overview of the B&P algorithm is shown in Algorithm 5.1. To keep the pseudocode compact we let z, $\boldsymbol{x}$, and $\boldsymbol{y}$, with vectors in bold font, denote the RMP's LP objective value, the node pattern solution vector, and the replication pattern solution vector; and $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, and η be the dual variables of the RMP. $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ refer to dual variables of the balance equations (5.3) and (5.4) (resp. (5.9) and (5.10)), $\boldsymbol{\gamma}$ stems from the branching constraints introduced in Section 5.5.3, while η represent the dual variable of (5.11), and is only present in the hybrid cloud model. Moreover, ζ, $\boldsymbol{w}$, and $\boldsymbol{v}$ denote the reduced cost of the new node pattern, the binary vector indicating the active replicas deployed in the new node pattern, and the binary vector indicating the passive replicas, corresponding to the node pattern coefficients W_{biq} and V_{biq}. In addition, letters with superscript I refer to the current incumbent solution.

In the B&P implementation, we have done two small changes to the master problem formulations. The equalities (5.3)-(5.4) and (5.9)-(5.10) are changed to $\geq$ constraints. This has at least two advantages. Firstly, it makes the dual variables of these constraints non-negative instead of free, which might help to reduce the instability of the dual solutions (Vanderbeck, 2005). Secondly, the feasible region is enlarged which should make it easier to find a feasible solution. However, this also means that one could obtain solutions that deploy more replicas than required. Especially, the passive replicas, which require relatively small amounts of resources, are susceptible to this.

The subproblem that is used to generate new node patterns can be formulated and solved in various ways. In general, subproblems can often be solved as MIPs, and in many applications, the subproblem can be formulated as a shortest path problem with resource constraints (SPPRC), which is solved using a label-setting algorithm (Irnich and Desaulniers, 2005). In addition, heuristic solution methods based on label-setting algorithms have been utilized successfully to solve different types of subproblems (Irnich and Desaulniers, 2005). Solving the subproblem by a label-setting algorithm has an advantage over a MIP solver in that it is relatively easy to extract more than one node pattern in each column generating iteration. Herein, we have formulated the subproblem as a MIP and an SPPRC. For the SPPRC, we develop both an exact and a heuristic label-setting algorithm, outlined in Section 5.5.2. However, when using a heuristic subproblem solver, one must complement the heuristic solver with an exact to maintain an exact B&P algorithm.

5.5.1. Formulating the Subproblem as a MIP

The subproblem can be formulated as the following MIP, where the α_{iq} and β_{iq} are the dual variables of the constraints (5.3) and (5.4) (resp. (5.9) and (5.10)) of the private cloud (resp. hybrid cloud) model. Furthermore, the formulation uses binary placement variables w_{iq} and v_{iq}, corresponding to the node pattern coefficients W_{biq} and V_{biq}; and binary variables s_i to indicate whether at least one replica of service i is placed on the node, or not. The amount of shared backup resources for activation of passive replicas are represented by the variables m_g.

$$\min \zeta = 1 - \Big(\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \alpha_{iq} w_{iq} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \beta_{iq} v_{iq} \Big) \tag{5.15}$$

$$w_{iq} + v_{iq} \leq s_i \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{5.16}$$

$$\sum_{i \in \mathcal{S}} s_i \leq N_S \tag{5.17}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} v_{iq} \leq N_P \tag{5.18}$$

$$m_g - (G_{Aiqg} - G_{Piqg}) v_{iq} \geq 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i, \forall g \in \mathcal{G} \tag{5.19}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} G_{Aiqg} w_{iq} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} G_{Piqg} v_{iq} + m_g \leq N_{Cg} \quad \forall g \in \mathcal{G} \tag{5.20}$$

$$m_g \geq 0 \quad \forall g \in \mathcal{G} \tag{5.21}$$

$$w_{iq} \in \{0, 1\} \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{5.22}$$

$$v_{iq} \in \{0, 1\} \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{5.23}$$

$$s_i \in \{0, 1\} \quad \forall i \in \mathcal{S} \tag{5.24}$$

The objective function (5.15) to be minimized corresponds to the reduced cost of a node pattern in the private cloud model. The expression for the reduced cost in the hybrid cloud model is discussed in Section 5.5.4. The constraints (5.16) ensure node-disjoint placement, and at the same time force s_i to take value 1, as long as there is at least one replica from service i selected in the node pattern. Moreover, (5.17) and (5.18) put upper bounds on the number of different services, and the number of passive replicas that can run on a node, respectively. The resource capacity of the nodes are handled by (5.20), where the first and second term account for the resources assigned to the active and passive replicas, and the third term accounts for the shared backup resources. This term is in turn lower-bounded by the inequalities (5.19). At last, (5.21) - (5.24) define the variables as non-negative or binary, respectively.

5.5.2. Formulating the Subproblem as an SPPRC

The underlying idea of the SPPRC formulation is to let the service components represent the vertices of a graph (two vertices per component, corresponding to an active and a passive replica), and then appropriately connect the vertices with directed arcs, such that the resulting graph is acyclic. The resources of the SPPRC include the reduced cost, the amount of a node's resource assigned to replicas, the amount of a node's resource reserved to let passive replicas to be activated, in addition to the number of passive replicas and the number of different services placed on the node. The goal is to find the minimum reduced cost path from a dummy source vertex to a dummy sink vertex, which is feasible with respect to the other resources. Such a feasible path correspond to a feasible node pattern, and a label-setting algorithm is used to obtain these paths.

The label-setting algorithm runs on a directed acyclic graph $(\mathcal{V}, \mathcal{A})$ which can be viewed as a two-layered network. The upper layer network is illustrated in Figure 5.1, and is composed of the dummy source and sink vertices, σ_0 and τ_0, and several *service blocks*, one for each service in $\mathcal{S}$. From σ_0, there is an arc to every service block; and from each service block, there are arcs to all other service blocks of higher order, in addition to an arc to τ_0. The service blocks can be ordered in different ways, and the ordering can change between separate calls to the label-setting algorithm. Designing the network with the service blocks as central elements is beneficial because of the restricted number of different services that can be visited, cf. constraint (5.17). That is, no more than N_S service blocks can be visited by a path.

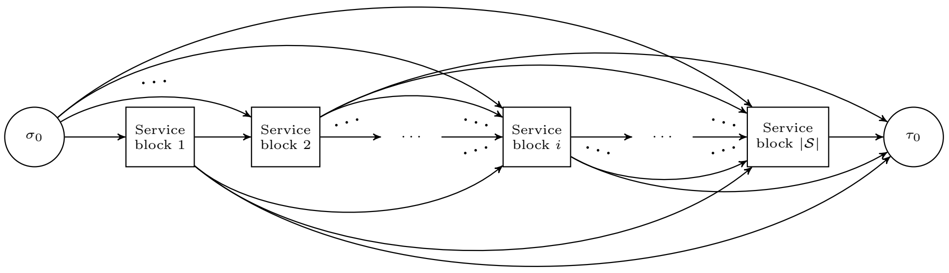

Figure 5.1.: The upper layer network of the SPPRC formulation

Each service block is a network in itself, and the structure of these subnets is depicted in Figure 5.2. The service block is composed of four types of vertices: one σ-vertex, one τ-vertex, one or more a-vertices and one or more p-vertices. The σ-vertex, labelled σ_i in Figure 5.2, is the entrance of the service block and every arc into the service block is directed to this vertex. Similarly, the τ-vertex, labelled τ_i in Figure 5.2, is the exit of the service block, and all arcs from a service block to another service block or to τ_0 are directed out from the τ-vertex. Each a-vertex in the service subnet represents an active replica, i.e. the vertex a_{i1} in

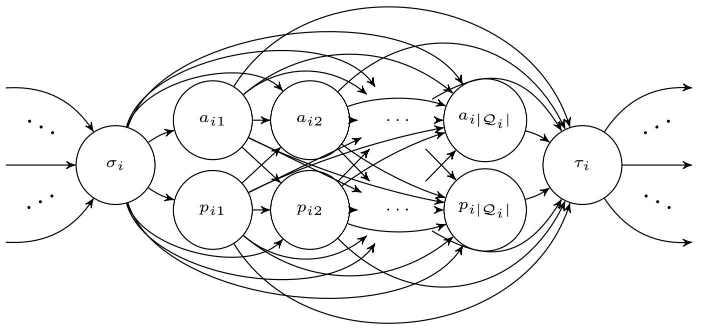

Figure 5.2.: The subnet of a service block in the SPPRC formulation

Figure 5.2 represents an active replica of the first component of service i; and a path visiting this vertex is interpreted as a placement of such a replica in the node pattern corresponding to the path. Analogously, each p-vertex in the service subnet represents a passive replica. Again, the ordering of the components can change between separate calls to the label-setting algorithm. From the σ-vertex, there is an arc to all a-vertices and p-vertices; and from an a-vertex or p-vertex, there are arcs to all a-vertices and p-vertices of higher order, in addition to the τ-vertex. Note that, there are no arcs between a-vertices and p-vertices representing active and passive replicas of the same component, and no arc directly from the σ-vertex to the τ-vertex.

Algorithm 5.2 outlines the basic structure of the label-setting algorithm. The pseudocode is kept compact and general, and thus does not show the special properties of the network. Each vertex ν maintains a set of labels, $\mathcal{L}_\nu$, and the algorithm is initialized by letting the label set of the dummy source vertex σ_0 have one label, with reduced cost 1, representing a path containing only the source vertex σ_0. Then, all vertices are considered in topological order; that is, if $\mathcal{A}$ contain an arc from ν to ω, vertex ν is considered before ω. Before the labels of a vertex ν is extended, all dominated labels in $\mathcal{L}_\nu$ are removed. At the end, when the dummy sink vertex τ_0 is considered, all labels in $\mathcal{L}_{\tau_0}$ with negative reduced cost are returned. The extension and domination of labels is detailed in Appendix 5.A.1 and Appendix 5.A.2, respectively.

Since a label-setting algorithm without a dominance rule essentially is a full enumeration algorithm, a critical algorithmic element is the efficiency of this rule. When conducting experiments with the label-setting algorithm on the SPPRC formulation, it became clear that the interaction between the shared backup re-

Algorithm 5.2 Pseudocode of the label-setting algorithm

```
Require: a directed acyclic graph $(\mathcal{V}, \mathcal{A})$
 1: $\mathcal{L}_{\sigma_0} \leftarrow \{\ell_0\}$
 2: for all vertices $\nu \in \mathcal{V}$ in topological order do
 3:     if domination should be conducted then
 4:         Domination: find and remove all dominated labels in $\mathcal{L}_\nu$
 5:     end if
 6:     for all labels $\ell \in \mathcal{L}_\nu$ do
 7:         for all feasible extensions of label $\ell$ do
 8:             Extension: extend label $\ell$ to obtain label $k$
 9:             $\omega \leftarrow$ vertex of $k$
10:             $\mathcal{L}_\omega \leftarrow \mathcal{L}_\omega \cup \{k\}$
11:         end for
12:     end for
13: end for
14: return all labels (paths) in $\mathcal{L}_{\tau_0}$ with negative reduced cost
```

sources, corresponding to m_g in the subproblem MIP, and the total amount of resources assigned to replicas on the node, made the dominance quite weak. However, fixing m_g before calling the label-setting algorithm would lead to much faster, but heuristic, solution method. There are two implications of this fixing that will increase the speed of the algorithm: the domination becomes more efficient, and one can reduce network size by disregarding the vertices representing passive replicas with $G_{Aiqg} - G_{Piqg} > m_g$ for at least one g. We consider two cases of fixings: m_g fixed to zero for all g, and m_g fixed to positive values. When using this heuristic label-setting algorithm in the B&P, we will in each call to the subproblem solver (Line 7 in Algorithm 5.1) run the label-setting algorithm twice, one with a zero-valued m_g and one with at least one positive m_g. More details on this heuristic dominance and network reduction, in addition to details on how the positive values of the m_g's are set, are found in Appendix 5.A.4.

5.5.3. Branching

In B&P, it is well known that branching directly on the variables generated by the subproblem results in an unbalanced tree. Such a branching rule would also be difficult to implement in the subproblem, and could destroy its structure (Vanderbeck, 2000). While it is not efficient to branch directly on the x_b variables, one can use the traditional branching rules for the binary variables y_{ir}. However, in our B&P algorithm, we prioritize branching on the node patterns as long as there exist fractional x_b variables, and we only describe the branching rule for the node patterns here. Regarding branching on the w_{Ciq} variables of the hybrid cloud model, we pointed out in Section 5.4.2 that these variables are naturally integer.

Several works discuss branching rules which ideas can be applied in our problem (Vanderbeck and Wolsey, 1996; Vance, 1998; Vanderbeck, 2000). Generally, these rules select a subset of the generated columns, such that the sum of the corresponding column variables are fractional. Based on this subset, one adds a constraint to the master problem that either bounds this sum to be less than or equal to the fractional value rounded down, or oppositely, bound this sum to be greater than or equal to the fractional value rounded up. However, the rule to select the subset of columns has to be designed such that it can be handled by the subproblem. For a cutting stock problem, Vance (1998) uses a branching rule that selects a set of rows, denoted a *branching set*, in the master problem, and choose the subset of columns to branch on as the columns that has coefficients greater a certain value in each selected row. We base our rule on this idea, but limit ourselves to consider pairs of rows.

First, we define $\mathcal{J}$ as the set of all branching sets of cardinality two, i.e., an element $j \in \mathcal{J}$ corresponds to a pair of rows in (5.3)-(5.4) (analogously (5.9)-(5.10)), which means that j can either represent two active replicas, two passive replicas, or one active and one passive replica. Because of the node-disjoint placement of replicas of the same component, it is only meaningful that the two rows of a branching set corresponds to two different service components, e.g., (i, q) and (i', q'). We use the symbols $H_{Ajiq} \in \{0, 1\}$ to indicate if the row corresponding to the row (i, q) in (5.3) is included in the branching set j ($H_{Ajiq} = 1$); and likewise $H_{Pjiq} \in \{0, 1\}$ to indicate if the row (i, q) in (5.4) is included in the branching set j ($H_{Pjiq} = 1$). Moreover, let us define the set $\mathcal{B}_j$ as the current set of node patterns that have coefficients one in the two rows represented by the branching set j, cf. (5.25).

$$\mathcal{B}_j = \{b \in \mathcal{B} \mid \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \; (W_{biq} \geq H_{Ajiq} \wedge V_{biq} \geq H_{Pjiq})\} \quad \forall j \in \mathcal{J} \quad (5.25)$$

Using the above definitions, branching is done by selecting a branching set $j \in \mathcal{J}$, such that

$$\sum_{b \in \mathcal{B}_j} x_b = \phi \notin \mathbb{Z}$$

is fractional in the current solution; and then use the following inequalities as branching constraints in the master problem.

$$\sum_{b \in \mathcal{B}_j} x_b \leq \lfloor \phi \rfloor \quad \text{and} \quad \sum_{b \in \mathcal{B}_j} x_b \geq \lceil \phi \rceil$$

Therefore, at a given B&B node t in the tree, one would have a set $\bar{\mathcal{J}}_t$ of active branching sets representing down branches (upper bounds), and a set $\underline{\mathcal{J}}_t$ of active branching sets representing up branches (lower bounds). The branching constraints are written as (5.26) and (5.27), where U_{tj} and L_{tj} are set to the fractional value rounded down and up, respectively.

$$\sum_{b \in \mathcal{B}_j} x_b \leq U_{tj} \quad \forall j \in \bar{\mathcal{J}}_t \tag{5.26}$$

$$\sum_{b \in \mathcal{B}_j} x_b \geq L_{tj} \quad \forall j \in \underline{\mathcal{J}}_t \tag{5.27}$$

We need to point out that it could happen that an active branching set $\hat{\jmath} \in \bar{\mathcal{J}}_t$ (or $\underline{\mathcal{J}}_t$) is used again when a new branching decision is made. In such a situation, the first child of B&B node t, say t_1 will have its current bound $U_{t_1\hat{\jmath}} < U_{t\hat{\jmath}}$ (or $L_{t_1\hat{\jmath}} > L_{t\hat{\jmath}}$), i.e. tightened, while the other child, say t_2 will have $\hat{\jmath}$ as element in both sets of active branching sets, that is, $\hat{\jmath} \in \bar{\mathcal{J}}_{t_2} \cap \underline{\mathcal{J}}_{t_2}$.

Furthermore, the branching rule will not make a node pattern invalid in any B&B node, which means that we do not need to keep track of a set of valid node patterns for each B&B node. In addition, when we solve the RMP as an IP, we disregard all branching constraints and optimize over all node patterns found so far.

A question that is not yet addressed is if this branching rule leads to a complete algorithm, that is, can we guarantee that one could always eliminate fractional x_b variables using this rule? Proposition 5 in Vanderbeck and Wolsey (1996) states that for the cutting stock problem with columns as binary vectors, one might have to use branching sets with cardinality greater than the maximum value of the right hand sides in the master problem. If we had fixed all the replication pattern variables y_{ir} in the enumeration tree to obtain fixed right hand sides in (5.3) - (5.4), we can calculate the maximum right hand side, which could possibly be observed in a B&B node, to

$$\bar{R} = \max\{\max_{i \in \mathcal{S}, q \in \mathcal{Q}_i, r \in \mathcal{R}_i} \{R_{Aiqr}\}, \max_{i \in \mathcal{S}, q \in \mathcal{Q}_i, r \in \mathcal{R}_i} \{R_{Piqr}\}\}.$$

In the test instances used in Section 5.6, this number would be much larger than two. Nevertheless, when setting upper bounds on the run time, as done in Section 5.6, we have not encountered situations where using branching sets with larger cardinality than two have been necessary to eliminate a fractional solution. Yet, we cannot be certain that if the maximum run time is increased, we will still be able to branch successfully on branching sets of cardinality two only.

5.5.3.1. Modifications of the Subproblem Formulations

The dual variables of the branching constraints (5.26) and (5.27), $\bar{\gamma}_j$ for all $j \in \bar{\mathcal{J}}_t$ and $\underline{\gamma}_j$ for all $j \in \underline{\mathcal{J}}_t$, need to be considered in the computation of the reduced cost of a new node pattern. In the MIP formulation of the subproblem this is done by introducing new binary variables, $\bar{p}_j$ for all $j \in \bar{\mathcal{J}}_t$, and $\underline{p}_j$ for all $j \in \underline{\mathcal{J}}_t$, which take value 1 if the branching set j is included in the new node pattern. With this definition the modified objective function is given in (5.28). Moreover, the inequalities (5.29)-(5.30) ensure that the $\bar{p}_j$ and $\underline{p}_j$ variables take the correct value. Note that since $\bar{\gamma}_j \leq 0$ and $\underline{\gamma}_j \geq 0$, the objective function will try to make $\bar{p}_j, j \in \bar{\mathcal{J}}_t$, equal to zero, and hence we only have to force it to take value one in case new node pattern will include the branching set j. For $\underline{p}_j, j \in \bar{\mathcal{J}}_t$, we have the opposite situation as the objective will try to make the variable equal to one, and therefore we have to force the variable to zero as long as the new node pattern will not include the branching set. Hence, each new branching constraint in the master problem will lead to one new binary variable and one new constraint in the MIP formulation of the subproblem.

$$\min \zeta_t = 1 - \Big(\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \alpha_{iq} w_{iq} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \beta_{iq} v_{iq} + \sum_{j \in \bar{\mathcal{J}}_t} \bar{\gamma}_j \bar{p}_j + \sum_{j \in \underline{\mathcal{J}}_t} \underline{\gamma}_j \underline{p}_j \Big) \tag{5.28}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} H_{Ajiq} w_{iq} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} H_{Pjiq} v_{iq} - \bar{p}_j \leq 1 \quad \forall j \in \bar{\mathcal{J}}_t \tag{5.29}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} H_{Ajiq} w_{iq} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} H_{Pjiq} v_{iq} - 2\underline{p}_j \geq 0 \quad \forall j \in \underline{\mathcal{J}}_t \tag{5.30}$$

We emphasize that adding branching constraints to the master problem and implementing the branching decisions in the subproblem formulations make the problems harder to solve. Therefore, it might be desirable to select B&B nodes high up in the tree when selecting new nodes in the B&P algorithm (cf. line 4 of Algorithm 5.1). Hence, we have implemented best first as the node selection strategy of our algorithm.

It is also necessary to modify the SPPRC formulation and label-setting algorithm to account for the branching decisions. Details on this is found in Appendix 5.A.3.

5.5.4. Generating Node Patterns in the Hybrid Cloud Model

In order to generate new improving node patterns in the hybrid cloud model (5.7) - (5.14), we have to make some small modifications to the subproblem. When computing the reduced cost of a new node pattern, we now have to take into account the dual variable η of constraint (5.11). Moreover, the objective coefficient of a new node pattern in the master problem objective (5.7) is zero, not one as in the private cloud model. The modified objective function of the MIP formulation is given in (5.31) below. Note that $\eta \leq 0$, and that the first term of the modified objective function is constant.

$$\min \zeta_t = -\eta - \Big(\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \alpha_{iq} w_{iq} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \beta_{iq} v_{iq} + \sum_{j \in \bar{\mathcal{J}}_t} \bar{\gamma}_j p_j + \sum_{j \in \underline{\mathcal{J}}_t} \underline{\gamma}_j p_j \Big) \quad (5.31)$$

Likewise the modifications in the MIP formulation, the modifications in SPPRC formulation and the label-setting algorithm are only minor. Instead of initializing the label-setting algorithm with a label with reduced cost 1, one initializes the algorithm with a label with reduced cost $-\eta$. Otherwise, the algorithm remain the same.

5.6. Numerical Results and Discussion

The main goal of our experimental study is to evaluate the different solvers for the subproblem. In addition, we are comparing the proposed B&P algorithm with the solution method presented in our earlier work (Gullhav and Nygreen, 2015). The comparisons are done on both the private cloud and hybrid cloud model. At the end of the section, we will also investigate and present some characteristics of the solutions of the hybrid cloud model. Before the results are presented and discussed, we will give a description of the setup of the experiments.

5.6.1. Experimental Setup

We have implemented the B&P and label-setting algorithms in C++, and compiled the code with GCC version 4.8.2 with option `-O3`. The experiments have been run on a CentOS 5.8 machine with a dual core 3.0GHz Intel E5472 Xeon processor and 16 GB of memory. The Xpress-Optimizer version 27.01.02 of the FICO Xpress Optimization Suite version 7.8 is used for solving the master problem and the MIP formulation of the subproblem. The MIP solver has been able to utilize eight threads in the B&B. All experiments have been given a maximum run time of five hours.

In the experiments, we have not used real data. Even so, we believe that the data we have used realistically represent real world cases. The test instances used are of six different sizes, ranging from 5 services to 50 services, and for each instance size, we have five cases. The cases are generated based on ten different dummy services with between three and five components each and different resource requirement characteristics. On average each service consists of four components, and hence, the total number of components in the largest case with 50 services amounts to 200. Each of the dummy services specifies distributions from which the G_{Aiqg} and G_{Piqg} parameters are drawn, and these distributions varies between the components of the same service to imitate real SaaS services. The case generation is controlled by a seed. In all cases, $N_S = 3$ and $N_P = 4$; and we consider the CPU as the only resource type in the experiments. The average value of G_{Aiqg} and G_{Piqg} are 23.0% and 1.60%, given in percent of the CPU capacity. Furthermore, the median of the maximum (minimum) G_{Aiqg} and G_{Piqg} over all cases are 45.0% and 3.00% (8.00% and 0.333%). The replication pattern data are also different among the cases, and the average number of replication patterns per service is 7.51.

Regarding the experiments on the hybrid cloud model, we also have to set a bound on the number of nodes in the private cloud, N_N. For each of the test cases described above, we can construct different cases for the hybrid cloud model by changing the N_N parameter, and we have chosen to give this parameter values based on the best bound obtained on the corresponding test case in the private cloud experiments. Specifically, we have set N_N to 75% or 90% of the best bound, rounded up to the nearest integer. For simplicity, we will refer to these percentages as the *private cloud coverage*. When placed in the public cloud, an active replica, say (i, q), is run in a VM which capacity is at least as large as G_{Aiqg}. Typically, the public cloud providers offer predetermined VM types of different cost and capacity, and most often the cost per resource unit are constant over all VM types. Herein, we have used the hypothetical VM types listed in Table 5.1, and as seen we assume that there exists in total six VM types offered by two different public cloud providers. Since we in our mathematical model do not distinguish between providers, we can in advance of the optimization compute the cost of placing an active replica of each component (i, q) in the public cloud. That is, C_{Ciq} equals the cost of cheapest VM that has larger capacity than G_{Aiqg}. Note that, the cost values in the table are not given units. Since we are not comparing the cost of running services in the private cloud with the cost of placement in the public cloud, the units are not important. However, the relative differences in cost and capacity of the VM types are reflecting the reality.

Table 5.1.: Data of the public cloud VM types used in the hybrid cloud experiments. The capacity is given in percentage of the private cloud node capacity.

Provider 1		Provider 2	
Cost	Capacity	Cost	Capacity
10	10%	15	15%
20	20%	30	30%
40	40%	60	60%

The test cases are primarily labelled based on the number of services, but also given a suffix `a` to `e` distinguishing the five different cases with an equal number of services. E.g., the five private cloud test cases with 20 services are labelled from `P20-a` to `P20-e`. When all cases with 20 services are referred as a whole, we ignore the suffix. Similarly, the hybrid cloud cases with 20 services are labelled `H20-a` to

`H20-e`.

In the B&P algorithm, one has the option of calling an IP solver (the MIP solver of Xpress) on the RMP (cf. line 20 of Algorithm 5.1). In the implementation used in the experiments, we have chosen to call the IP solver directly after the root node is solved, and then after every 100th solved B&B node. In order not to spend too much time on solving the IPs, we have set the maximum run time of the IP solver to 600 seconds. Another parameter, which has to be set, is the maximum number of node patterns that is added to the RMP when the label-setting algorithms are used as solution method for the subproblem. For the exact label-setting algorithm (E-LSA) we have set the maximum number of node patterns to 20. Since the heuristic label-setting algorithm (H-LSA) solves two different networks in each iteration, one with $m_g = 0$ and one with $m_g > 0$, we will add at most 10 node patterns from each of the two runs to the RMP.

5.6.2. Evaluation of the Solution Methods

As we want to maintain an exact solution approach, we have to complement the heuristic solution method with an exact solution method for the subproblem. Therefore, we are interested in assessing the relative performance of the exact solution methods. To do this, we are using two different setups of the B&P algorithm in the tests: one where the subproblem is solved by using the Xpress MIP solver on the MIP formulation in Section 5.5.1; and one where the subproblem is solved using the E-LSA of Section 5.5.2. Table 5.2 displays the results of experiments on the cases with 5 and 10 services. We can see that we are able to solve all of the ten smallest cases to optimality before we reach the maximum run time. Moreover, E-LSA uses fewer iterations but adds more node patterns to master problem. Since we add up to 20 node patterns to the master problem every time the subproblem is solved, we should expect to solve the subproblem fewer times. While the solution times of the E-LSA is on par with the solution times of the MIP in the `P5` cases, we can see that the solution times increase dramatically when studying the `P10` cases. For these cases, the MIP is clearly faster; and, not shown in the table, the differences in solution time continue to increase for the larger cases. By this, we conclude that we prefer to use the MIP as the exact solution method for the subproblem. Moreover, the results disclose that the LP relaxation of the node pattern formulation is very tight. In all of the cases displayed in Table 5.2, the objective value of the optimal solution is equal to the rounded up value of the objective of the RMP when finishing the root node. When using the E-LSA as the subproblem solver, we are able to find the optimal solution by solving the RMP as an IP after finishing the root node. When the MIP is used as the subproblem solver, there are two cases, `P5-a` and `P10-a`, where the IP of the RMP does not give the optimal solution after the root node. Nevertheless, the next time the RMP is solved as an IP, after 100 more B&B nodes, the optimal solutions are found.

Table 5.2.: Comparison of the exact solution methods for the subproblem: MIP vs. Exact label-setting algorithm (E-LSA). The column *np* lists the number of generated node patterns, while the column *iter.* lists the total number of times the subproblem is solved. Solution times are given in seconds.

	MIP						**E-LSA**					
	Best sol.	Best bound	Sol. time	B&B nodes	np	iter.	Best sol.	Best bound	Sol. time	B&B nodes	np	iter.
P5-a	14	14	121	101	299	400	14	14	47	1	500	34
P5-b	17	17	22	1	133	134	17	17	18	1	434	29
P5-c	15	15	30	1	169	170	15	15	29	1	525	35
P5-d	14	14	28	1	155	156	14	14	50	1	519	38
P5-e	18	18	28	1	146	147	18	18	53	1	507	41
P10-a	32	32	329	101	516	617	32	32	4582	1	955	55
P10-b	32	32	103	1	352	353	32	32	838	1	823	50
P10-c	35	35	134	1	363	364	35	35	2182	1	930	61
P10-d	32	32	112	1	363	364	32	32	752	1	854	50
P10-e	37	37	106	1	357	358	37	37	922	1	808	45

In the experiments on the larger cases, where many B&B nodes are explored, we experience B&B nodes where neither the heuristic nor the exact subproblem solver manages to find new improving node patterns. Consequently, in many B&B nodes only one iteration with an exact solution method is necessary to finish the node. On average in the cases with 20, 30, 40 and 50 services, a new improving node pattern is generated by the subproblem in only 25% of the B&B nodes. In a B&B node that does not produce a single node pattern, using the heuristic subproblem solver, which has to be followed by an exact, is clearly a wasted iteration. Ideally, we would like to know in advance if it is possible to generate an improving node pattern in a B&B node; and if so, we want to use the heuristic subproblem solver; otherwise, we will call the exact subproblem solver directly. However, we do not have this type of information; but we try to make a prediction of the possibility of generating an improving node pattern in a B&B node by comparing the final LP objective value of the node's parent and the current LP objective value of the node itself. In a minimization problem, the objective value of a B&B node t, say z_t, can never be less than the final objective value of the parent node, say z_t^{PAR}, i.e., $z_t \geq z_t^{PAR}$; and when node patterns with negative reduced cost are added to the RMP, z_t decreases monotonically. Thus, if the relative difference between these values is small, we suggest that there is less probable to find an improving node pattern in this B&B node. Specifically, we say that if the relative difference is less than a threshold, denoted by Δ, we call the exact subproblem solver next.

This condition is shown in inequality (5.32).

$$\frac{z_t - z_t^{PAR}}{z_t^{PAR}} \leq \Delta \tag{5.32}$$

Table 5.3 shows some results of the experiments with the B&P with the H-LSA as the primary subproblem solver, where the Δ is given different values between zero and one. In the extreme case where $\Delta = 1$, the B&P will never use the H-LSA in other B&B nodes than the root node, and if $\Delta = 0$, the B&P will use the H-LSA in every B&B node until it cannot find an improving node pattern, and then switch to the exact solution method. Moreover, for each of the instance sizes with 30, 40 and 50 services, the table lists the number of solved cases (out of 5 for each size), the average relative gap[1] between the objective value of the best found solution and the best bound after reaching the run time limit, the average percentage of subproblem iterations conducted with the H-LSA, and the average number of B&B nodes. At the bottom of the table, there are also averages over all cases with 30 services or more. Generally, we can see that when Δ is given a large value, the B&P algorithm explores a larger amount of B&B nodes compared to the cases when Δ is given a small value. This result might be explained by, as already discussed, the high amount of B&B nodes where no improving node patterns can be found, and thus smaller values on Δ, implies more wasted calls to the H-LSA. Overall, the results show that using the H-LSA in the B&P tree is valuable, at least if we manage to control when to use the H-LSA and when to call the exact solution method directly. We see that the smallest average gap is achieved when $\Delta = 1 \cdot 10^{-5}$. For this value, we observe that on average the H-LSA is used as the solution method in about 20% of the subproblem iterations, as opposed to about 40% when $\Delta = 0$.

In Gullhav and Nygreen (2015), we compared the performance and solution quality of the direct MIP formulation and a heuristic algorithm that pre-generates a subset of all maximal node patterns, gives this as input to the pattern-based models, and subsequently solves the formulations of Section 5.4 as an IP. The results showed that the pre-generation algorithm outperformed the direct MIP formulation, both in terms of speed and solution quality. Here, we will concentrate on comparing the B&P algorithm with the pre-generation algorithm. Specifically, we will consider two versions of the B&P algorithm: one which only solves the subproblem with the exact MIP model of Section 5.5.1; and one which used the H-LSA as the primary subproblem solver whenever the relative difference between the parent objective and the current objective is less than $1 \cdot 10^{-5}$. If the H-LSA fails to find an improving node pattern, the exact MIP model is used as the solver. We denote these two alternative B&P algorithms for B&P MIP and B&P H-LSA

[1] $\frac{\text{obj. val. of best solution - final best bound}}{\text{final best bound}}$

Table 5.3.: H-LSA results with different thresholds, Δ. *Avg. H-LSA iter.* refers to the percentage of calls to the H-LSA subproblem solver out of the total number of times the subproblem is solved. Run time limit: 5 hours.

		Threshold on relative difference, Δ					
		1	$1 \cdot 10^{-4}$	$5 \cdot 10^{-5}$	$1 \cdot 10^{-5}$	$5 \cdot 10^{-6}$	0
P30	Solved cases	1	0	1	1	1	1
	Avg. gap	1.359%	1.347%	1.158%	0.9559%	1.157%	0.9580%
	Avg. H-LSA iter.	4.468%	5.004%	7.811%	18.17%	23.35%	42.26%
	Avg. B&B nodes	1522	1747	1572	1450	1335	1175
P40	Solved cases	0	0	0	0	0	0
	Avg. gap	3.225%	3.070%	3.216%	2.788%	2.936%	2.935%
	Avg. H-LSA iter.	5.107%	5.845%	9.188%	23.00%	26.46%	40.26%
	Avg. B&B nodes	1535	1507	1500	1230	1204	1099
P50	Solved cases	0	0	0	0	0	0
	Avg. gap	4.013%	3.901%	3.783%	3.554%	3.783%	4.267%
	Avg. H-LSA iter.	7.503%	9.376%	9.579%	18.62%	22.54%	40.02%
	Avg. B&B nodes	1192	1152	1109	955.8	912.0	832.0
P30, P40 & P50	Avg. gap	2.866%	2.772%	2.719%	2.433%	2.625%	2.720%
	Avg. H-LSA iter.	5.692%	6.742%	8.859%	19.93%	24.12%	40.85%
	Avg. B&B nodes	1416	1469	1394	1212	1150	1035

in the following. Regarding the pre-generation algorithm, we have generated node patterns similarly to how it was done in our earlier paper, and for each of the cases in the subsequently presented results we have included a subset of between 250 000 and 300 000 maximal node patterns (out of millions) in the optimization runs. For the P50 cases, this corresponds to about 0.1% of the total number of maximal node patterns. Our experience is that if one includes more node patterns than this, it will not improve the solutions significantly, but instead slow down the MIP solver.

Table 5.4 compares the average gaps of the pre-generation and the B&P algorithms at different time steps for the cases with 20 or more services. The results of the pre-generation is reported without accounting for the time it takes to pre-generate the node patterns, and the gaps are calculated based on the final best bound of the B&P H-LSA, which in all cases equals the final best bound of the B&P MIP. One can see from the columns of Table 5.4 labelled *Final*, that after a maximum of five hours of run time, the B&P H-LSA has the smallest average gaps in all cases. Generally, at a given time step the B&P H-LSA has found better solutions than both the pre-generation and the B&P MIP. Also, the B&P MIP outperforms pre-generation in most cases. For the P50 case we can see that the average gap after 3600 seconds for the B&P MIP is not given. The explanation of this is that the B&P MIP has not yet solved the root node, and subsequently the master problem as an IP, after 3600 seconds in any of the P50 cases. Therefore, no integer solutions are found at this point. Not shown in the table, the magnitude of the final gaps of B&P H-LSA vary from 1 for the unsolved `P20` cases to between 4 and 7 for the `P50` cases. In the latter cases, the best bounds are in the interval $[162, 174]$. Furthermore, both B&P MIP and B&P H-LSA solves three of the five `P20` cases, and B&P H-LSA solves one of the five `P30` cases. The pre-generation cannot find the optimal solution in any case.

Table 5.4.: Average relative gap (in %) at different time steps (seconds): comparison of the pre-generation algorithm and the branch and price algorithms. Run time limit: 5 hours.

	Pre-generation					**B&P MIP**					**B&P H-LSA**				
	3600	7200	10800	14400	Final	3600	7200	10800	14400	Final	3600	7200	10800	14400	Final
`P20`	3.895	3.895	3.592	3.002	3.002	1.824	1.553	1.553	1.553	1.241	1.553	0.924	0.620	0.620	0.620
`P30`	8.196	5.423	4.663	4.663	4.663	2.909	2.320	2.320	1.916	1.732	2.522	1.729	1.343	1.139	0.956
`P40`	7.903	6.566	5.552	5.251	4.674	5.098	3.804	3.509	3.070	3.070	3.958	3.081	2.938	2.788	2.788
`P50`	9.119	9.119	7.224	6.258	5.798	N/A	5.677	4.844	4.614	4.493	5.144	4.494	4.137	3.789	3.554

Now, we are concentrating on experiments with the hybrid cloud model of Section 5.4.2. The B&P H-LSA and pre-generation algorithms is set up equivalent to when it was run on the private cloud cases. That is, Δ is still $1 \cdot 10^{-5}$, and the pre-generation algorithm optimizes a hybrid cloud case over the same node patterns that were used for the corresponding private cloud case.

Table 5.5 compares the gaps at different points in time of the pre-generation algorithm and the B&P H-LSA algorithm for the hybrid cloud cases with 90% private cloud coverage. Again, the gaps of the pre-generation algorithm is computed based on the final best bound of the B&P H-LSA algorithm. Our tests show that the B&P H-LSA produces better results than the B&P MIP, but for presentation purposes, we exclude these results from the tables regarding hybrid cloud cases. However, the table displays that the branch and price algorithm is clearly better than pre-generation for all cases. The gaps of the B&P H-LSA are smaller than those of the pre-generation algorithm. Table 5.5 also shows that the average gaps are larger compared to the private cloud results in all cases. Moreover, the time to solve the root node is longer for both solution methods, and the B&P H-LSA is able to find a solution within 3600 seconds in only one of the five `H50` cases. On the other hand, the pre-generation algorithm finds a solution within 3600 seconds in four of the five `H50` cases, but the gap in one of these cases is as much as 98.40%. When considering the hybrid cloud cases with 75% private cloud coverage, we obtain similar results. As seen in Table 5.6, the B&P H-LSA produce solutions with smaller gaps in all cases. However, the relative gaps are smaller than in the cases with 90% private cloud coverage, and so these cases are seemingly easier to solve. This observation is consistent with the results in (Gullhav and Nygreen, 2015), where we showed that the hybrid cloud model became harder to solve as the number of nodes in the private cloud approached the best found solution of the private cloud model.

Table 5.5.: Average relative gap (in %) at different time steps (seconds): comparison of the pre-generation algorithm and the branch and price algorithm using H-LSA on the hybrid cloud cases with 90% private cloud coverage. Run time limit: 5 hours.

	Pre-generation					**B&P H-LSA**				
	3600	7200	10800	14400	Final	3600	7200	10800	14400	Final
H20	28.99	27.20	25.63	23.58	23.41	8.044	6.775	5.146	5.146	5.146
H30	41.43	39.13	37.28	36.84	34.97	17.64	14.85	12.08	11.49	11.38
H40	107.9	45.14	44.30	43.71	42.45	23.49	18.39	17.64	17.26	16.88
H50	62.38	48.44	46.00	43.54	43.54	25.74	22.39	22.33	20.74	18.66

Table 5.6.: Average relative gap (in %) at different time steps (seconds): comparison of the pre-generation algorithm and the branch and price algorithm using H-LSA on the hybrid cloud cases with 75% private cloud coverage. Run time limit: 5 hours.

	Pre-generation					**B&P H-LSA**				
	3600	7200	10800	14400	Final	3600	7200	10800	14400	Final
H20	10.89	10.17	9.410	9.180	9.180	2.007	1.769	1.709	1.304	1.196
H30	14.34	13.05	12.51	12.47	12.47	4.126	3.933	3.576	3.497	3.497
H40	16.74	14.84	14.73	14.67	14.15	7.208	5.847	5.489	5.899	5.092
H50	37.53	16.13	15.99	17.24	17.24	N/A	7.641	6.738	5.606	5.454

5.6.3. Comparison of the Private Cloud and Hybrid Cloud Solutions

The private cloud and hybrid cloud models are in many ways quite similar, but the different objective functions and the upper bound on the number of nodes in the private cloud of the hybrid cloud model give their respective solutions distinct features. In the private cloud model, the focus is on finding efficient ways to pack a node, while in the hybrid cloud model, one also have to decide which active replicas should be placed in the public cloud. At first, one could think that obtaining efficient packings of the nodes still was the primary success factor for obtaining good solutions in the hybrid cloud model, and that the active replicas which did not fit in the selected efficient packings were moved to the public cloud.

Nevertheless, the solutions of the hybrid cloud model show that there are more aspects than this factor that are important in a good solution.

With the node capacity normalized to 100, Table 5.1 shows that the cost per unit of resource is equal 1 for all VM types. But since the public cloud VM types are offered in pre-defined discrete sizes which might not fit exactly with the resource requirement of an active replica of (i, q), G_{Aiqg}, the cost per unit of resource might be higher. E.g., if $G_{Aiqg} = 32$, this active replica would fit in the largest VM of provider 1 in Table 5.1 and cost 40. The cost per unit of resource is then $40/32 = 1.25$, which is considerable higher than 1. In the private cloud model, this unit cost is of no importance for the solution quality, but in the hybrid cloud model it is a meaningful factor. To compare the solutions of the private cloud cases and the hybrid cloud cases we have computed a *cost-resource-ratio* (CRR) by dividing the total cost of placing a set of active replicas in the public cloud by the sum of resources (G_{Aiqg}) required by the same active replicas. The second column of Table 5.7 gives the CRR of the best solutions of the private cloud cases, i.e., the CRR is computed by dividing the (potential) cost of placing all the active replicas required in the public cloud by the total amount of resource required by the active replicas. The columns labelled *tot* give the same numbers for the hybrid cloud cases with 90% and 75% private cloud coverage. Note that, these numbers are not influenced directly by the placement, but computed based on the replication patterns selected in the solution. The table shows that the CRR computed over all active replicas are often identical. In the cases where they are differing, this must mean that the solutions use different replication patterns. What is more interesting is the CRR computed over all active replicas placed in the public cloud (columns labelled *puc*). These ratios are much smaller than the ratio of the private cloud cases and the ratios over all active replicas in the hybrid cloud cases. This observation is important as it tells us that the optimization process selects the active replicas with relatively low cost per resource unit to be placed in the public cloud. Just for comparison, Table 5.7 also presents the CRR computed over all active replicas placed in the private cloud (cf. column *prc*) and it shows that these ratios are higher than the ratio of the private cloud cases. This is naturally because the active replicas with low cost per resource unit are placed in the public cloud, and so the relatively "expensive" active replicas remain in the private cloud. Furthermore, if one compares the CRR computed over the active replicas in the public cloud one can see that the CRRs for the cases with 90% private cloud coverage are always smaller than the ones with 75% private cloud coverage. This finding supports the interpretation above as it means that if the private cloud coverage is increased, the optimization process will move the active replicas in the public cloud with relatively high cost per resource unit back to the private cloud.

Lastly, Table 5.8 compares the objective values of the best found solutions of the hybrid cloud cases divided by the reduction in the number of nodes compared to

Table 5.7.: The cost-resource-ratio (CRR) of the private cloud solution, and the hybrid cloud solutions with 90% and 75% private cloud coverage (pcc). For the latter, the CRR is computed over all active replicas (*tot*), over the active replicas placed in the private cloud (*prc*), and over the active replicas placed in the public cloud (*puc*).

	Private cloud	**Hybrid cloud: 90% pcc**			**Hybrid cloud: 75% pcc**		
		prc	*tot*	*puc*	*prc*	*tot*	*puc*
H20-a	1.2124	1.2338	1.2123	1.0491	1.2753	1.2123	1.0750
H20-b	1.2234	1.2459	1.2234	1.0680	1.2888	1.2234	1.0810
H20-c	1.2218	1.2444	1.2211	1.0355	1.2825	1.2211	1.0811
H20-d	1.1986	1.2177	1.1986	1.0489	1.2589	1.1983	1.0604
H20-e	1.2197	1.2393	1.2202	1.0676	1.2795	1.2202	1.0788
H30-a	1.2098	1.2336	1.2114	1.0516	1.2725	1.2113	1.0816
H30-b	1.2269	1.2448	1.2269	1.0758	1.2920	1.2269	1.0848
H30-c	1.2240	1.2451	1.2239	1.0647	1.2840	1.2237	1.0848
H30-d	1.1876	1.2059	1.1876	1.0531	1.2459	1.1876	1.0595
H30-e	1.2078	1.2293	1.2091	1.0603	1.2720	1.2091	1.0705
H40-a	1.2023	1.2251	1.2021	1.0603	1.2617	1.2020	1.0796
H40-b	1.2166	1.2403	1.2166	1.0709	1.2801	1.2165	1.0862
H40-c	1.2234	1.2487	1.2234	1.0577	1.2878	1.2234	1.0824
H40-d	1.1966	1.2172	1.1966	1.0535	1.2581	1.1961	1.0646
H40-e	1.2112	1.2322	1.2111	1.0651	1.2706	1.2110	1.0822
H50-a	1.2036	1.2272	1.2036	1.0596	1.2676	1.2034	1.0770
H50-b	1.2150	1.2388	1.2150	1.0668	1.2802	1.2150	1.0736
H50-c	1.2230	1.2494	1.2232	1.0571	1.2892	1.2231	1.0823
H50-d	1.1958	1.2173	1.1958	1.0597	1.2575	1.1954	1.0682
H50-e	1.2099	1.2318	1.2099	1.0727	1.2716	1.2098	1.0789
Average	1.2115	1.2334	1.2116	1.0599	1.2738	1.2115	1.0766

the best found solution of the corresponding private cloud case, that is, $z_H/(z_P - N_N)$. We denote these numbers as the *public cloud node cost*. By regarding the costs of the public cloud VM types in Table 5.1, the minimum cost of moving all active replicas of a fully utilized node to the public cloud is 100. Since the CRR values of the active replicas in the public cloud are considerably higher than 1 (cf. Table 5.7), one should expect a higher cost than 100. However, the public cloud node costs presented in Table 5.8 take (average) values both above and below 100. The latter can be explained by the fact that in many cases the average node utilization in the private cloud cases is significantly less than 100%. Hence, when $z_P - N_N$ nodes are removed from the solution of a private cloud case, one does not have to compensate by leasing resources from a public cloud provider equivalent to $z_P - N_N$ fully utilized nodes. Furthermore, Table 5.8 also illustrates that the public cloud node cost is consistently less when the private cloud coverage is 90% compared to 75%, which is natural because the average utilization of the N least utilized nodes is reduced as N_N is increased, and thus fewer resources per node has to be compensated for by using the public cloud. Another feature of the numbers in the table is that the public cloud node costs seems to decrease with the size of the test cases, with the H30 cases being exceptions. This can be explained by the fact that the gaps between the best found solution and the best bound in the private cloud cases increase with the problem size, and as a consequence, the average node utilization is lower in the larger cases.

Table 5.8.: Public cloud node cost. Comparison between the best solutions of the hybrid cloud cases with 90% and 75% private cloud coverage (pcc).

	90% pcc	**75% pcc**
H20	96.24	104.1
H30	98.15	104.7
H40	90.73	100.1
H50	87.75	98.12

5.7. Conclusions

We have studied two versions of the service deployment problem proposed in Gullhav and Nygreen (2015): one for periods where the private cloud of the service provider has enough resource capacity to run all services, and one for periods where a public cloud is used for placement as well, forming a hybrid cloud. A B&P algorithm was proposed to solve the problems. The subproblem of the B&P was solved by a MIP solver and a label-setting algorithm, which was run on a network designed to exploit the special problem structure. Firstly, we developed an exact label-setting algorithm, but due to a weak dominance rule, the run time of the algorithm did not scale well to larger problems. One contribution of this paper is a heuristic label-setting algorithm (H-LSA), which based on the problem structure, heuristically reduce the underlying network and simplify the dominance rule. The H-LSA was complemented with the exact MIP solver to obtain an exact and complete B&P algorithm. Our experiments showed that using the H-LSA sped up the solution process. However, we also observed that in many B&B nodes, no node patterns with negative reduced cost did exist, and calling the H-LSA in such a case results in a wasted iteration. Another contribution of this paper is a novel strategy to predict the likeliness of finding node patterns with negative reduced cost in a B&B node, and in nodes where this seems unlikely, we directly call the MIP solver.

We also compared the B&P algorithm with the pre-generation algorithm presented in Gullhav and Nygreen (2015), and the results showed that the B&P

outperformed the other algorithm. Moreover, the B&P managed to solve all test cases with 10 services or less, but as the problem size grew the relative gap between the objective value of the best-found solution and the best bound increased when a maximum run time was set. For the cases with 50 services, the gap was over five percent on average after five hours of run time. The results of the experiments on the hybrid cloud model indicate that this model is more difficult to solve than the private cloud model; the relative gaps were larger and it took longer time to solve the root node. We also obtained results showing that the hybrid cloud model becomes harder to solve when the private cloud coverage approaches 100%. Since the hybrid cloud case with equally many services and service components as a private cloud case, but with fewer nodes, are more difficult to solve than the private cloud case, this must mean that it is not necessarily easy to select the active replicas for placement in the public cloud.

By analyzing the solutions of the hybrid cloud model, we found as expected that the active replicas placed in the public cloud fit better in the VM types offered in the public cloud, i.e., they have less cost per unit of resource required, than the active replicas placed in the private cloud. This is a critical factor in a good solution.

Bibliography

Amazon Web Services. Amazon Web Services (AWS) - Cloud Computing Services, 2015. URL `http://aws.amazon.com/`. Last visited 2015/03/04.

D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. Energy-aware autonomic resource allocation in multitier virtualized environments. *IEEE Transactions on Services Computing*, 5(1):2–19, 2012.

N. Ashrafi, O. Berman, and M. Cutler. Optimal design of large software-systems using n-version programming. *IEEE Transactions on Reliability*, 43(2):344–350, 1994.

A. Beloglazov, R. Buyya, Y. C. Lee, and A. Y. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, 82(2):47–111, 2011.

E. Bin, O. Biran, O. Boni, E. Hadad, E. Kolodner, Y. Moatti, and D. Lorenz. Guaranteeing high availability goals for virtual machine placement. In *2011 31st International Conference on Distributed Computing Systems*, pages 700–709, 2011.

D. Breitgand and A. Epstein. SLA-aware placement of multi-virtual machine elastic services in compute clouds. In N. Agoulmine, C. Bartolini, T. Pfeifer, and D. O'Sullivan, editors, *Integrated Network Management*, pages 161–168. IEEE, 2011.

B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174, Berkeley, CA, USA, 2008. USENIX.

T. Distler, R. Kapitza, I. Popov, H. P. Reiser, and W. Schröder-Preikschat. SPARE: Replicas on hold. In *Proceedings of the 18th Network and Distributed System Security Symposium*, Geneva, Switzerland, 2011. The Internet Society.

T. C. Ferreto, M. A. Netto, R. N. Calheiros, and C. A. D. Rose. Server consolidation with migration control for virtualized data centers. *Future Generation Computer Systems*, 27(8):1027 – 1034, 2011.

H. Goudarzi and M. Pedram. Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 324–331, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

A. N. Gullhav and B. Nygreen. Deployment of replicated multi–tier services in cloud data centres. *International Journal of Cloud Computing*, 4(2):130–149, 2015.

A. N. Gullhav, B. Nygreen, and P. E. Heegaard. Approximating the response time distribution of fault-tolerant multi-tier cloud services. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 287–291, Los Alamitos, CA, USA, 2013. IEEE Computer Society.

V. Gunnerud, B. A. Foss, K. I. M. McKinnon, and B. Nygreen. Oil production optimization solved by piecewise linearization in a branch & price framework. *Computers & Operations Research*, 39(11):2469 – 2477, 2012.

S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 33 – 65. Springer, New York, USA, 2005.

I. Iyoob, E. Zarifoglu, and A. B. Dieker. Cloud computing operations research. *Service Science*, 5(2):88–101, 2013.

B. Jennings and R. Stadler. Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, 23(3):567–619, 2015.

M. Jepsen, B. Petersen, S. Spoorendonk, and D. Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511, 2008.

R. Jhawar, V. Piuri, and M. Santambrogio. Fault tolerance management in cloud computing: A system-level perspective. *IEEE Systems Journal*, 7(2):288–297, 2013.

W. Kuo and R. Wan. Recent advances in optimal reliability allocation. In G. Levitin, editor, *Computational Intelligence in Reliability Engineering*, volume 39 of *Studies in Computational Intelligence*, pages 1–36. Springer Berlin Heidelberg, 2007.

I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Architecture-driven reliability and energy optimization for complex embedded systems. In G. T. Heineman, J. Kofron, and F. Plasil, editors, *Research into Practice – Reality and Gaps*, volume 6093 of *Lecture Notes in Computer Science*, pages 52–67. Springer Berlin Heidelberg, 2010.

P. Mell and T. Grance. The NIST definition of cloud computing, 2011. NIST SP 800-145.

D. Menasce. QoS issues in web services. *Internet Computing, IEEE*, 6(6):72–75, 2002.

V. Petrucci, O. Loques, and D. Mossé. A dynamic optimization model for power and performance management of virtualized clusters. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 225–233, New York, NY, USA, 2010. ACM.

ROADEF. ROADEF/EURO challenge 2012: Machine reassignment, 2012. URL `http://challenge.roadef.org/2012/en/`. Last visited 2015/10/01.

B. Speitkamp and M. Bichler. A mathematical programming approach for server consolidation problems in virtualized data centers. *IEEE Transactions on Services Computing*, 3(4):266–278, 2010.

P. H. Vance. Branch-and-price algorithms for the one-dimensional cutting stock problem. *Computational Optimization and Applications*, 9:211–228, 1998.

F. Vanderbeck. On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48 (1):111–128, 2000.

F. Vanderbeck and L. A. Wolsey. An exact algorithm for ip column generation. *Operations Research Letters*, 19(4):151 – 159, 1996.

F. Vanderbeck. Implementing mixed integer column generation. In G. Desaulniers, J. Desrosiers, and M. Solomon, editors, *Column Generation*, pages 331–358. Springer US, 2005.

Appendix 5.A. SPPRC Details

Appendix 5.A.1. Labels and Label Extension

Based on the network description in Section 5.5.2, we let $\mathcal{V}_A$ ($\mathcal{V}_P$) be the set of all a-vertices (p-vertices) in all service subnets; and $\mathcal{V}_\sigma$ and $\mathcal{V}_\tau$ be the sets of all σ-vertices and τ-vertices, respectively. Note that σ_0 and τ_0 of the upper layer network (Figure 5.1) are not contained in the latter sets. Moreover, since a visit to an a-vertex or p-vertex is analogous to a placement of an active or passive replica, we have to assign the dual variables, which are used in the mathematical formulation of the subproblem in Section 5.5.1, to the respective vertices. Therefore, a vertex $\nu \in \mathcal{V}_A$ is assigned a dual variable α_ν corresponding to the dual variable α_{iq} of the service-component pair (i, q) that the vertex represents. Likewise, each vertex $\nu \in \mathcal{V}_P$ is assigned a dual variable β_ν corresponding to the dual variable β_{iq}. Lastly, we also keep track of the resource usage of a node pattern by assigning each $\nu \in \mathcal{V}_A$ resource consumption values $G_{A\nu g}$ for all resource types $g \in \mathcal{G}$, resembling the resource consumption G_{Aiqg} of (i, q) which ν represents. Similarly, each $\nu \in \mathcal{V}_P$ is assigned the resource consumption values $G_{P\nu g}$ for all $g \in \mathcal{G}$.

To conduct the label-setting algorithm, each label stores data according to Table 5.9. We use the notation $\psi(\ell)$ to refer to the vertex of label ν, and similarly $\lambda(\ell)$, $\zeta(\ell)$, $f_g(\ell)$, $m_g(\ell)$, $\pi(\ell)$ and $\xi(\ell)$ to refer to the rest of the data listed in the table.

Table 5.9.: The data stored for each label

Symbol	Description
ψ	pointer to the vertex of the label
λ	pointer the predecessor label
ζ	the reduced cost of the label
m_g	the amount of resource of type $g \in \mathcal{G}$ reserved for activation of passive replicas
f_g	the accumulated resource of type $g \in \mathcal{G}$ (i.e. CPU, memory, etc.), including m_g
π	the accumulated number of passive replicas
ξ	the accumulated number of different services

The path represented by a label ℓ is feasible if all of the following conditions hold:

$$f_g \le N_{Cg}, \quad \forall g \in \mathcal{G} \tag{5.33}$$

$$\pi \le N_P \tag{5.34}$$

$$\xi \le N_S \tag{5.35}$$

These conditions are equivalent to the constraints (5.20), (5.18) and (5.17), respectively.

When a label ℓ is extended to a vertex ν to form a new label k, the data stored for the new label is constructed as shown below. (5.38) - (5.42) are denoted resource extension functions.

$$\psi(k) = \nu \tag{5.36}$$

$$\lambda(k) = \ell \tag{5.37}$$

$$\zeta(k) = \begin{cases} \zeta(\ell) - \alpha_\nu & \text{if } \nu \in \mathcal{V}_A \\ \zeta(\ell) - \beta_\nu & \text{if } \nu \in \mathcal{V}_P \\ \zeta(\ell) & \text{otherwise} \end{cases} \tag{5.38}$$

$$m_g(k) = \begin{cases} m_g(\ell) + \\ \quad \max\{G_{A\nu g} - G_{P\nu g} - m_g(\ell), 0\} & \text{if } \nu \in \mathcal{V}_P \\ m_g(\ell) & \text{otherwise} \end{cases} \quad \forall g \in \mathcal{G} \tag{5.39}$$

$$f_g(k) = \begin{cases} f_g(\ell) + G_{A\nu g} & \text{if } \nu \in \mathcal{V}_A \\ f_g(\ell) + G_{P\nu g} + \\ \quad \max\{G_{A\nu g} - G_{P\nu g} - m_g(\ell), 0\} & \text{if } \nu \in \mathcal{V}_P \\ f_g(\ell) & \text{otherwise} \end{cases} \quad \forall g \in \mathcal{G} \tag{5.40}$$

$$\pi(k) = \begin{cases} \pi(\ell) + 1 & \text{if } \nu \in \mathcal{V}_P \\ \pi(\ell) & \text{otherwise} \end{cases} \tag{5.41}$$

$$\xi(k) = \begin{cases} \xi(\ell) + 1 & \text{if } \nu \in \mathcal{V}_\sigma \\ \xi(\ell) & \text{otherwise} \end{cases} \tag{5.42}$$

Functions (5.36) and (5.37) set the vertex and predecessor, respectively, and (5.38) computes and stores the accumulated reduced cost, based on to which vertex set ν belongs. Note that the label constructed in the initialization in Algorithm 5.2 has an accumulated reduced cost of 1, corresponding to the constant term in the objective function (5.15) in the MIP formulation ($-\eta$ in the hybrid cloud model). Furthermore, (5.39) computes the amount of resources reserved for activation of passive replicas. If the vertex ν represents a passive replica and $G_{A\nu} - G_{P\nu} > m_g(\ell)$, $m_g(k)$ is increased from $m_g(\ell)$, otherwise $m_g(k)$ remains at the level of $m_g(\ell)$. (5.40) computes the accumulated resources. If vertex ν represents an active replica, the extension comprises adding the resource usage of ν to the accumulated resources of ℓ. On the other hand, if ν represents a passive replica, one also have to account from a possible increase in the resources reserved for activation of passive replicas, likewise done in (5.39). If ν neither represents an active nor passive replica, $f_g(k)$ takes the same value as $f_g(\ell)$. Lastly, (5.41) and (5.42) increment the number of passive replicas and different services if ν represents a passive replica or is an entrance of a service block, respectively.

Appendix 5.A.2. Dominance Criteria

The goal of implementing the dominance step of Algorithm 5.2 is to reduce the number of extended labels, and thereby speed up the solution procedure. It should be noted that without the dominance step, Algorithm 5.2 would have been a procedure enumerating all feasible paths.

In order to describe our dominance criteria, we define the set of path extensions for ℓ, $\mathcal{E}(\ell)$, as the partial paths, starting in vertex $\psi(\ell)$ and ending in the sink vertex τ_0, that can be concatenated with the partial path represented by ℓ, and result in a resource-feasible path from σ_0 to τ_0. Then, domination is defined as follows; label ℓ_1 *dominates* label ℓ_2 if:

$$\psi(\ell_1) = \psi(\ell_2) \tag{5.43}$$

$$\zeta(\ell_1) \leq \zeta(\ell_2) \tag{5.44}$$

$$\mathcal{E}(\ell_1) \supseteq \mathcal{E}(\ell_2) \tag{5.45}$$

If conditions (5.43) - (5.45) hold, it is certain that any path being constructed by concatenating the partial path represented by ℓ_1 and any partial path $e \in \mathcal{E}(\ell_2)$ cannot have a worse reduced cost than the path constructed from the partial path represented by ℓ_2 and the same e. Therefore, ℓ_1 dominates ℓ_2, and the label ℓ_2 can be removed from the set of labels at vertex $\psi(\ell_2)$, i.e. $\mathcal{L}_{\psi(\ell_2)}$.

Unfortunately, in most cases, ours including, it is not practical to check condition (5.45) directly. Nevertheless, since the resource extension functions (5.39) - (5.42) of the SPPRC are non-decreasing, the conditions (5.46) - (5.49) below imply condition (5.45) (Irnich and Desaulniers, 2005).

$$m_g(\ell_1) \geq m_g(\ell_2) \quad \forall g \in \mathcal{G} \tag{5.46}$$

$$f_g(\ell_1) \leq f_g(\ell_2) \quad \forall g \in \mathcal{G} \tag{5.47}$$

$$\pi(\ell_1) \leq \pi(\ell_2) \tag{5.48}$$

$$\xi(\ell_1) \leq \xi(\ell_2) \tag{5.49}$$

The domination is performed in line 4 of Algorithm 5.2, and it consists of checking if labels dominate each other, and if so remove the dominated label. If the domination is implemented naïvely, one has to compare every label with all other labels, in total $|\mathcal{L}_\nu|^2$ comparisons at vertex ν. However, if one maintain the set of labels at a node ordered according to the reduced cost ζ, one only has to compare every label with other labels with worse (i.e., higher) reduced cost (cf. domination criterion (5.44)).

In line 3 of Algorithm 5.2, we have the option to decide whether the domination step should be conducted in a given vertex or not. We have conducted the domination step in every vertex, except the dummy source and sink, σ_0 and τ_0; and at the source vertex σ_1 of the first service.

Appendix 5.A.3. Implementing the Branching in the SPPRC

The basic idea on how to deal with the branching decisions is to keep track of which branching sets a label has visited, and if visited for a second time, i.e., both replicas corresponding to a branching set are included in the label, we have to correct the reduced cost of the label with the dual variable of the branching set. First define $\mathcal{V}_{Cj} \subset (\mathcal{V}_A \cup \mathcal{V}_P)$ for each $j \in \bar{\mathcal{J}}_t \cup \underline{\mathcal{J}}_t$ as the set of vertices representing the service component pairs (i, q) with $H_{Ajiq} = 1$ or $H_{Pjiq} = 1$. Note that, the cardinality of $\mathcal{V}_{Cj}$ is always two. In addition to the data in Table 5.9, every label ℓ has to keep track of the number of times the vertices related to branching set $j \in \bar{\mathcal{J}}_t$ ($j \in \underline{\mathcal{J}}_t$) is visited. We denote these counters as $\bar{\theta}_j$ ($\underline{\theta}_j$). When extending a label ℓ to label k at vertex ν, these counters are updated according to (5.50)-(5.51). At every extension, we also need to monitor if the label visits a branching set for the second time, and if so subtract the corresponding dual variable from the accumulated reduced cost. To do this, we build the sets $\bar{\mathcal{F}}$ and $\underline{\mathcal{F}}$ as the set of branching sets that are visited for the second time at the current vertex, ν. After updating $\bar{\theta}_j$ and $\underline{\theta}_j$, the sets are built as shown in (5.52) and (5.53) for each label k. By using these sets, the resource extension function for updating the reduced costs, (5.38), are updated to (5.54).

$$\bar{\theta}_j(k) = \begin{cases} \bar{\theta}_j(\ell) + 1 & \text{if } \nu \in \mathcal{V}_{Cj} \\ \bar{\theta}_j(\ell) & \text{otherwise} \end{cases} \quad \forall j \in \bar{\mathcal{J}}_t \tag{5.50}$$

$$\underline{\theta}_j(k) = \begin{cases} \underline{\theta}_j(\ell) + 1 & \text{if } \nu \in \mathcal{V}_{Cj} \\ \underline{\theta}_j(\ell) & \text{otherwise} \end{cases} \quad \forall j \in \underline{\mathcal{J}}_t \tag{5.51}$$

$$\bar{\mathcal{F}}(k) = \{j \in \bar{\mathcal{J}}_t \mid \nu \in \mathcal{V}_{Cj} \wedge \bar{\theta}_j(k) = 2\} \tag{5.52}$$

$$\underline{\mathcal{F}}(k) = \{j \in \underline{\mathcal{J}}_t \mid \nu \in \mathcal{V}_{Cj} \wedge \underline{\theta}_j(k) = 2\} \tag{5.53}$$

$$\zeta(k) = \begin{cases} \zeta(\ell) - \alpha_\nu - \sum_{j \in \bar{\mathcal{F}}(k)} \bar{\gamma}_j - \sum_{j \in \underline{\mathcal{F}}(k)} \underline{\gamma}_j & \text{if } \nu \in \mathcal{V}_A \\ \zeta(\ell) - \beta_\nu - \sum_{j \in \bar{\mathcal{F}}(k)} \bar{\gamma}_j - \sum_{j \in \underline{\mathcal{F}}(k)} \underline{\gamma}_j & \text{if } \nu \in \mathcal{V}_P \\ \zeta(\ell) & \text{otherwise} \end{cases} \tag{5.54}$$

The introduction of branching constraints in the master problem also influences how the domination can be conducted. When comparing the reduced costs of two labels to check if ℓ_1 dominates ℓ_2, one can have that label ℓ_1 has visited one vertex in a branching set $\bar{j}$, while ℓ_2 has not. If $\bar{j}$ is an element in $\bar{\mathcal{J}}_t$, another visit by an extension of ℓ_1 to this branching set will incur a *penalty* of $\bar{\gamma}_{\bar{j}}$, while an extension of ℓ_2 to the same vertex will not incur a penalty. Moreover, if ℓ_2 has visited one vertex, not visited by ℓ_1, in a branching set $\underline{j} \in \underline{\mathcal{J}}_t$, the extension of label ℓ_2 will receive a *bonus* of $\underline{\gamma}_{\underline{j}}$ if the partial path, represented by ℓ_2, is extended to the second vertex of the branching set, while an extension of ℓ_1 to this vertex will not get this bonus. A way to overcome this issue is to compensate the reduced cost of ℓ_1 by the potential penalties which can be observed by an extension of ℓ_1, but not by an extension of ℓ_2; and compensate the reduced cost of ℓ_2 by the potential bonuses which can be observed by an extension of ℓ_2, but not by an extension of ℓ_1. Jepsen et al. (2008) uses this idea in order to account for the dual variables stemming from subset-row (SR) inequalities in the master problem of a vehicle-routing problem. Since the SR inequalities are $\leq$-type of constraints, they only have to compensate the reduced cost with potential penalties in their dominance criteria. Now we define $\bar{\mathcal{P}}$ (and $\underline{\mathcal{P}}$) as the set of branching sets $j \in \bar{\mathcal{J}}_t$ (and $\underline{\mathcal{J}}_t$) that are already visited once and has the potential to be visited once more. After extension from label ℓ to label k at vertex ν, the sets are built according to (5.55) and (5.56), where $\omega \succ \nu$ reads as ω succeeds ν in the topological order of the acyclic network, i.e., ω can potentially be visited after the current vertex ν. Using these definitions, we rewrite the dominance criterion (5.44) to (5.57). That is, label ℓ_1 dominates label ℓ_2 if (5.43), (5.46)-(5.49) and (5.57) holds.

$$\bar{\mathcal{P}}(k) = \{j \in \bar{\mathcal{J}}_t \mid \bar{\theta}_j(k) = 1 \wedge \exists \omega \in \mathcal{V}_{Cj}\ (\omega \succ \nu)\} \tag{5.55}$$

$$\underline{\mathcal{P}}(k) = \{j \in \underline{\mathcal{J}}_t \mid \underline{\theta}_j(k) = 1 \wedge \exists \omega \in \mathcal{V}_{Cj}\ (\omega \succ \nu)\} \tag{5.56}$$

$$\zeta(\ell_1) - \sum_{j \in (\bar{\mathcal{P}}(\ell_1) \setminus \bar{\mathcal{P}}(\ell_2))} \bar{\gamma}_j \quad \leq \quad \zeta(\ell_2) - \sum_{j \in (\underline{\mathcal{P}}(\ell_2) \setminus \underline{\mathcal{P}}(\ell_1))} \underline{\gamma}_j \tag{5.57}$$

Appendix 5.A.4. Details of the Heuristic Label-setting Algorithm

The idea of the heuristic label-setting algorithm is to fix the m_g's in advance of calling Algorithm 5.2. We consider two cases: $m_g = 0$ for all $g \in \mathcal{G}$, in which no passive replicas can be placed in the current node pattern; and $m_g > 0$ for at least one resource type g, which means that passive replicas with $G_{Aiqg} - G_{Piqg} > m_g$ cannot be placed in the node pattern. It is very easy to adapt the SPPRC formulation and the label-setting algorithm to the former case. All vertices $\nu \in \mathcal{V}_P$, i.e., vertices representing passive replicas, together with all arcs connected to ν are removed from the SPPRC graph, and the dominance criteria (5.46) and (5.48) are disregarded. Otherwise, the algorithm works as before. Certainly, this network reduction and simplification of the dominance rule will have considerable impact on the run time of one pass of the algorithm. In the latter case, with non-zero m_g, we choose to pre-select $|\mathcal{G}|$ passive replicas for placement, assuming that $|\mathcal{G}| \leq N_P$ holds in the following. Each of these replicas will function as a benchmark for the m_g (for different resources g). That is, $m_g = (G_{A\hat{\imath}_g\hat{q}_g g} - G_{P\hat{\imath}_g\hat{q}_g g})$ for all $g \in \mathcal{G}$ where the service components $(\hat{\imath}_1, \hat{q}_1), \ldots, (\hat{\imath}_g, \hat{q}_g), \ldots, (\hat{\imath}_{|\mathcal{G}|}, \hat{q}_{|\mathcal{G}|})$ are the pre-selected passive replicas. To implement this pre-selection and fixing in the SPPRC formulation and label-setting algorithm, more changes has to be done, but the changes are still simple. Firstly, we remove the vertices representing the active replicas of service components $(\hat{\imath}_g, \hat{q}_g)$ for all $g \in \mathcal{G}$ in addition to the connected arcs. Furthermore, all vertices $\nu \in \mathcal{V}_P$ with $(G_{A\nu g} - G_{P\nu g}) > m_g$ for at least one g, and the arcs connected to these vertices, are removed. Then, the order of the services is slightly changed to facilitate the forced placement of the pre-selected passive replicas. Firstly, the service blocks of the pre-selected replicas, denoted *pre-selection blocks*, are ordered as the first service blocks (cf. Figure 5.1), and the arcs from σ_0 to all service blocks except the first are removed. From all the pre-selection blocks, except the last one in the ordering, there are only arcs from the τ-vertex to the σ-vertex of the next pre-selection block. From the last pre-selection block, there are arcs to all forward service blocks and the τ_0 vertex. Moreover, the subnet of the pre-selection blocks (cf. Figure 5.2) is rearranged by letting the $p_{\hat{\imath}_g\hat{q}_g}$ vertices be ordered as the first. All arcs out of the σ-vertices are removed except the one arc to the first of possibly $|\mathcal{G}|$ pre-selected passive replicas of the service. In case there are several pre-selected replicas in a subnet, each pre-selected replica is linked by an arc, while the last pre-selected replica has arcs to every other a-vertex and p-vertex of higher order. Lastly, the dominance rule will disregard criterion (5.46).

A fundamental observation about good solutions of the problem is that node patterns typically contain either zero or N_P passive replicas, and in the latter case, the passive replicas have quite similar resource requirements for activation $(G_{Aiqg} - G_{Piqg})$. We use this observation when deciding the pre-selected passive replicas, that is, we try to pick $|\mathcal{G}|$ passive replicas which together with $N_P - |\mathcal{G}|$ other passive replicas make up a set of passive replicas with large dual variables compared to their total resource usage if placed together on a node. For presentation purposes, we have omitted the details of the procedure used to pre-select passive replicas.

Paper V

Anders N. Gullhav, Jean-François Cordeau, Lars Magnus Hvattum and Bjørn Nygreen:

Adaptive Large Neighborhood Search Heuristics for Multi-tier Service Deployment Problems in Clouds

submitted to an international journal

Adaptive Large Neighborhood Search Heuristics for Multi-tier Service Deployment Problems in Clouds

Abstract:
This paper proposes adaptive large neighborhood search (ALNS) heuristics for two service deployment problems in a cloud computing context. The problems under study consider the deployment problem of a provider of software-as-a-service applications, and include decisions related to the replication and placement of the provided services. A novel feature of the proposed algorithms is a local search layer on top of the destroy and repair operators. In addition, we use a mixed integer programming-based repair operator in conjunction with other faster heuristic operators. Thus, the proposed algorithms can be classified as matheuristics. Because of the different time consumption of the repair operators, we need to account for the time usage in the scoring mechanism of the adaptive operator selection. The computational study investigates the benefits of implementing a local search operator on top of the standard ALNS framework. Moreover, we also compare the proposed algorithms with a branch and price (B&P) approach previously developed for the same problems. The results of our experiments show that the benefits of the local search operators increase with the problem size. We also observe that the ALNS with the local search operators outperforms the B&P on larger problems, but it is also comparable with the B&P on smaller problems with a short run time.

6.1. Introduction

An increasing proportion of enterprise and business software, such as customer management systems, email systems and time management applications, are run as web services in clouds through the software-as-a-service (SaaS) model. However, Marston et al. (2011) identify the lack of quality of service and availability guarantees as one of the major weaknesses of adopting cloud software services. Even though frameworks and software systems offering fault tolerance management in clouds have been proposed (Cully et al., 2008; Distler et al., 2011; Jhawar et al., 2013), there exist very few optimization models considering fault tolerance by introduction of redundancy (Avižienis et al., 2004) in the literature. Distler et al. (2011) present a fault tolerance approach based on active-passive replication, where passive backup replicas are run in a paused state, from which they can be activated rapidly. The passive replicas do not serve demand while being paused, and the replicas consume considerably less resources than corresponding demand-serving active replicas. We take these ideas into account when regarding the service deployment problem of a SaaS provider (SP). In this problem we consider decisions related to the replication of the SaaS services simultaneously with placement decisions. In previous work (Gullhav and Nygreen, 2015a), we presented two mathematical models for the problem: one that considers service placement in a hybrid cloud, and another one that only considers placement in the private cloud of the service provider. We refer to Mell and Grance (2011) for definitions of the different types of clouds.

The SP offers a set of SaaS services, modeled as multi-tier services, to its clients. A typical example of a multi-tier service is a three-tier web service composed of a web server, an application server and a database server. When deployed in a cloud environment, each of the tiers, referred to as components of the service, run in separate virtual machines (VMs). In turn, the VMs are placed on physical machines, which we refer to as nodes. In our work, we assume that the service provider owns and operates one or more data centers forming a private cloud, and can lease additional VMs in a public cloud when needed. Furthermore, the services of the SP are required to have a certain level of quality of service (QoS), as specified in service level agreements (SLAs), i.e., contracts between the SP and its clients. The QoS might be specified in terms of bounds on the performance, e.g., the response time, and bounds on the dependability, e.g., the availability or downtime. To achieve the guaranteed QoS, each service component can be replicated into a number of load-balanced replicas, and additional backup replicas of the components might be deployed to achieve an appropriate fault tolerance. We denote the former type of replicas as active and the latter as passive. The overall objective of the problem is to find the minimum cost deployment while respecting the QoS requirements of the SLA and other technical requirements, such as node resource capacities. When a service provider offers multiple services, and these services interact through their placement (i.e., run on the same nodes), we argue that the cheapest way to replicate the tiers of a service is dependent of how other services are replicated. Therefore, it is valuable for the SP to consider the replication and placement of the services simultaneously.

The literature proposing related placement problems is discussed in our previous paper (Gullhav and Nygreen, 2015a) and a recent survey on resource management in clouds is given by Jennings and Stadler (2015). Goudarzi and Pedram (2011) and Ardagna et al. (2012) propose resource allocation models for deployment of QoS-constrained multi-tier services. Their models do not concern backup replication and placement of backup replicas as very few models in the service placement literature do. However, Bin et al. (2011) propose a solution method for a placement problem of an infrastructure-as-a-service (IaaS) provider, where some VMs require one or more backup locations to which they can be migrated in case of a failure. Another problem related to ours is the redundancy allocation problem, where the goal is to find the minimum cost allocation of parallel components to different subsystems in series, while maintaining a reliability higher than a given level (Kuo and Wan, 2007).

In Gullhav and Nygreen (2015a), we modeled the problem as a direct mixed-integer program (MIP). In addition, the problem was reformulated as a stronger pattern-based model. The latter formulation was solved by an a priori column generation algorithm, also called pre-generation, where a subset of the feasible patterns were given to the master problem in advance of the optimization. Furthermore, in Gullhav and Nygreen (2015b), we proposed a branch and price (B&P) algorithm, where patterns were generated dynamically instead of a priori. The B&P algorithm outperformed the pre-generation algorithm.

The contribution of this paper is to introduce two novel adaptive large neighborhood search (ALNS) algorithms for two variants of the service deployment problem. In addition to destroy and repair operators, which are part of the standard ALNS framework, a novel feature of the algorithms is a local search layer on top of the repair operators. A key question we seek to answer in the computational study of this paper is what benefits the local search operators could bring to the ALNS. Another special feature of the proposed algorithms is a MIP-based repair operator, in addition to other heuristic insertion operators. Since the repair operators vary with respect to their time-performance trade-off, we score the operators according to both their performance and time consumption in the adaptive operator selection. Furthermore, the computational study also compares the speed and solution quality of the ALNS algorithms and the previously proposed B&P algorithm for the two versions of the service deployment problem.

The outline of the paper is as follows. In the next section, we present a description of the service deployment problem, and in Section 6.3, we repeat the direct MIP formulation of Gullhav and Nygreen (2015a). In Section 6.4, we give a description of the components of the proposed ALNS algorithms. The computational study is presented and discussed in Section 6.5, before Section 6.6 concludes the paper.

6.2. Problem Description

Let $\mathcal{S}$ be the set of multi-tier services, and let $\mathcal{Q}_i$ denote the set of components of service $i \in \mathcal{S}$. For brevity, we will denote component $q \in \mathcal{Q}_i$ of service i as the pair (i, q). The VMs running the service components might run in a public cloud or on the set of nodes, $\mathcal{N}$, in the private cloud of the service provider, and each node has a set of limited resources, $\mathcal{G}$, e.g, CPU, memory, and storage. The nodes are assumed to be identical, and have resource capacities N_{Cg} for all resources $g \in \mathcal{G}$. When placed on a node, an active replica of the pair (i, q) consumes G_{Aiqg} resources of type g. The public cloud IaaS providers offer different VM types of a fixed capacity and cost to run the service components in the public cloud. As an example, Amazon Web Services (2015) offers several general purpose VM types, ranging from `micro` to `10xlarge`, with stepwise increases in capacity and cost. When placing an active replica of the pair (i, q) in the public cloud, this replica is run in the VM type that offers at least G_{Aiqg} resources for all g, and the cost of this VM type is denoted by C_{Ciq}. However, while active replicas can be run in the public cloud, we assume that support for passive replicas are only present on the nodes in the private cloud, and when run on the nodes in a passive state, the passive replicas require G_{Piqg} ($< G_{Aiqg}$) resources. To ensure that passive replicas can be activated, each node that runs one or more passive replicas must maintain an unassigned pool of resources. One has to make a trade-off between cost and fault tolerance when setting the size of this pool since a small pool might make it impossible to activate a passive replica on the node, while a large pool will result in a large amount of unused resources in a failure-free situation. Here, the size of the pool of shared backup resources is set to be larger than the resources required to activate any of the passive replicas running on the node. In addition, the number of passive replicas run on a node is limited to N_P. Moreover, the replicas of the same service component are required to be run on different nodes. This policy is referred to as node-disjoint placement. Otherwise, a single node failure could bring down several replicas of the same component. In the following, N_{Cg} is assumed to be normalized to 1 for all resources and, hence, G_{Aiqg} and G_{Piqg} are fractions of the node resource capacities.

The replication of the service components is done to obtain a certain level of performance and make the service fault tolerant. We do not consider a specific QoS measure, but instead assume that there exists a method to check whether given replication levels of all the components of a service result in a tolerable QoS according to the SLA. Gullhav et al. (2013) propose a method that takes the number of active and passive replicas of each component of a service and the component's assigned resources as input, and outputs an approximate response time distribution. The method also assumes that the VMs fail according to a Poisson process, and takes this into account when computing the approximation. This method can in principle be used here if the SLA specifies bounds on the mean or a percentile of the response time distribution of a service. Moreover, Gullhav and Nygreen (2015a) have introduced a modeling structure, called *replication patterns*, that specifies the number of active and passive replicas of all components of a service. With this structure, we manage to maintain a linear MIP model of the problem. We let $\mathcal{R}_i$ be the set of replication patterns of service i, and let R_{Aiqr} and R_{Piqr} denote the number of active and passive replicas of the pair (i, q)

in replication pattern $r \in \mathcal{R}_i$.

In the QoS guarantees, we do not account for the network latency. When the VMs of a service are placed in the same data center, or the private cloud, this latency can be neglected. When VMs placed in different data centers or clouds communicate, the latency should ideally be accounted for. However, doing this simplification makes our models much less complex. Nevertheless, we set an upper bound on the number of different services on a node to N_S, which drives different components of the same service to be run on the same nodes. In turn, this will reduce the amount of inter-node communication in the private cloud.

6.3. Direct MIP Formulation

The direct MIP formulation (Gullhav and Nygreen, 2015a) uses binary variables w_{iqn} and v_{iqn} to indicate the placement of an active replica and a passive replica of component $q \in \mathcal{Q}_i$ of service $i \in \mathcal{S}$ on node $n \in \mathcal{N}$ in the private cloud, respectively. Furthermore, the integer variables w_{Ciq} are used to keep track of the number of active replicas placed in the public cloud. The binary variables y_{ir} indicate the selection of a replication pattern $r \in \mathcal{R}_i$ for service i, and we use the variables m_{ng} to represent the amount of resource of type g that is reserved for activation of passive replicas on node n. Lastly, the binary variables s_{in} indicate whether a replica belonging to service i is placed on node n, or not. With these definitions, the service deployment problem can be formulated as follows:

$$\min z = \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{Ciq} w_{Ciq} \tag{6.1}$$

subject to:

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1 \quad \forall i \in \mathcal{S} \tag{6.2}$$

$$\sum_{n \in \mathcal{N}} w_{iqn} + w_{Ciq} - \sum_{r \in \mathcal{R}_i} R_{Aiqr} y_{ir} = 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{6.3}$$

$$\sum_{n \in \mathcal{N}} v_{iqn} - \sum_{r \in \mathcal{R}_i} R_{Piqr} y_{ir} = 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{6.4}$$

$$w_{iqn} + v_{iqn} - s_{in} \leq 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i, \forall n \in \mathcal{N} \tag{6.5}$$

$$\sum_{i \in \mathcal{S}} s_{in} \leq N_S \quad \forall n \in \mathcal{N} \tag{6.6}$$

$$\sum_{i\in\mathcal{S}}\sum_{q\in\mathcal{Q}_i} v_{iqn} \leq N_P \quad \forall n \in \mathcal{N} \tag{6.7}$$

$$m_{ng} - (G_{Aiqg} - G_{Piqg})v_{iqn} \geq 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i, \forall n \in \mathcal{N}, \forall g \in \mathcal{G} \tag{6.8}$$

$$\sum_{i\in\mathcal{S}}\sum_{q\in\mathcal{Q}_i} G_{Aiqg}w_{iqn} + \sum_{i\in\mathcal{S}}\sum_{q\in\mathcal{Q}_i} G_{Piqg}v_{iqn} + m_{ng} \leq 1 \quad \forall n \in \mathcal{N}, \forall g \in \mathcal{G} \tag{6.9}$$

$$m_{ng} \geq 0 \quad \forall n \in \mathcal{N}, \forall g \in \mathcal{G} \tag{6.10}$$

$$w_{iqn} \in \{0,1\} \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i, \forall n \in \mathcal{N} \tag{6.11}$$

$$v_{iqn} \in \{0,1\} \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i, \forall n \in \mathcal{N} \tag{6.12}$$

$$s_{in} \in \{0,1\} \quad \forall i \in \mathcal{S}, \forall n \in \mathcal{N} \tag{6.13}$$

$$y_{ir} \in \{0,1\} \quad \forall i \in \mathcal{S}, \forall r \in \mathcal{R}_i \tag{6.14}$$

$$w_{Ciq} \in \mathbb{Z}_+ \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{6.15}$$

The objective function (6.1) minimizes the total cost of placing replicas in the public cloud, and the equalities (6.2) ensure that one replication pattern is selected for each service. The two sets of equalities (6.3) and (6.4) establish the relation between the placement variables, w_{iqn}, w_{Ciq} and v_{iqn}, and the replication pattern variables y_{ir}, and so ensure that the correct number of active and passive replicas of each pair (i, q), according the chosen replication pattern, are placed on the nodes or in the public cloud. The rest of the constraints model the technical requirements related to the placement in the private cloud. Specifically, the inequalities (6.5) take care of the requirement specifying that the replicas of the same pair (i, q) should be placed on different nodes, and at the same time force s_{in} to take value 1, as long as there is at least one replica from service i deployed on n. Moreover, constraints (6.6) and (6.7) put upper bounds on the number of different services, and the number of passive replicas on each node, respectively. The resource capacities of the nodes are handled by constraints (6.9), where the first and second terms account for the resources assigned to the active and passive replicas deployed on the node, and the third term accounts for the resources reserved for activation of passive replicas, which is set by the inequalities (6.8).

We now want to consider the special case where the number of nodes in the private cloud is large enough to run all replicas privately. As opposed to the *hybrid cloud model* (6.1) - (6.15) above, we refer to this model as the *private cloud model.* The objective function for this case might consist of several cost components, but we have chosen to focus on the power consumption of the nodes in the private cloud. In data centers, the power consumption of a node in an idle state is significant and can be as large as 70 % of the peak power consumption (Beloglazov et al., 2011). Hence, a common strategy, also applied in VM placement models in the literature, is to minimize the number of nodes used for placement. Herein, we also implement this objective, and introduce the binary variables u_n indicating whether node n is turned on and used for placement, or not. The hybrid cloud model (6.1)-(6.15) is then modified by replacing the objective function by (6.16). In addition, the active deployment equalities (6.3) are reduced to the equalities (6.17). We also need to prevent nodes that are turned off from being used, which is achieved by replacing constraints (6.9) by constraints (6.18). Finally, the variable definitions (6.19) are also added to the private cloud model, such that the private cloud model is formulated as the problem of minimizing (6.16) subject to (6.2), (6.17), (6.4)-(6.8), (6.18), (6.10)-(6.14), and (6.19).

$$\min z = \sum_{n \in \mathcal{N}} u_n \tag{6.16}$$

$$\sum_{n \in \mathcal{N}} w_{iqn} - \sum_{r \in \mathcal{R}_i} R_{Aiqr} y_{ir} = 0 \quad \forall i \in \mathcal{S}, \forall q \in \mathcal{Q}_i \tag{6.17}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} G_{Aiqg} w_{iqn} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} G_{Piqg} v_{iqn} + m_{ng} - u_n \leq 0 \quad \forall n \in \mathcal{N}, \forall g \in \mathcal{G} \tag{6.18}$$

$$u_n \in \{0,1\} \quad \forall n \in \mathcal{N} \tag{6.19}$$

6.4. The Adaptive Large Neighborhood Search

The Adaptive Large Neighborhood Search (ALNS) is an extension of the Large Neighborhood Search (LNS) metaheuristic framework presented by Shaw (1997), and is also related to the ruin and recreate principle of Schrimpf et al. (2000). The ALNS was first proposed by Ropke and Pisinger (2006), and has been used in many applications since then. Pisinger and Ropke (2010) give an overview of the LNS and ALNS algorithms, and review some of the literature on applications of the algorithms.

The basic principle of the ALNS and LNS is to iteratively destroy and repair a solution, and accept the new solution as the incumbent if some criterion is met. A simple acceptance criterion is to only accept improving solutions, while a criterion commonly used in the literature is the simulated annealing (SA) acceptance criterion. The ALNS extends and deviates from the LNS by including several destroy operators and repair operators in the search for improving solutions. That is, in each iteration one destroy operator and one repair operator are chosen as the neighborhood operators. The algorithms proposed in this paper also adopt the ideas of the Iterated Local Search (ILS) paradigm (Lourenço et al., 2010). The search process of ILS is based on iteratively improving a solution by local search until a local optimum is found, and then perturbing the solution before continuing with local search from the modified solution. The perturbation works as the diversification mechanism, and should be designed so that the search process manages to escape local optima. The local search works as the intensification mechanism. To increase the intensification of the ALNS, we have implemented different local search operators that are called after the solution is repaired.

Algorithm 6.1 shows a pseudocode that gives an overview of the proposed ALNS metaheuristics. The sets of destroy, repair and local search operators are denoted $\mathcal{O}_D$, $\mathcal{O}_R$, and $\mathcal{O}_L$, respectively. In Line 3, the destroy operator θ, repair operator ρ, and local search operator λ to be used in the current iteration are biasedly selected using the vectors of weights $\mathbf{\Psi}_D$, $\mathbf{\Psi}_R$, and $\mathbf{\Psi}_L$. In Line 12, the weights of the operators are updated based on quality of the new solution, σ. Lines 3 and 12 are discussed in more detail in Section 6.4.5, while the different destroy, repair and local search operators are described in Sections 6.4.2, 6.4.3, and 6.4.4, respectively.

Algorithm 6.1 Pseudocode of the ALNS metaheuristic

```
Require: a feasible solution $\sigma$ //$\bar{\sigma}$ is the incumbent solution and $\hat{\sigma}$ is the best
    found solution
 1: $\eta = 0$; $\bar{\sigma} = \sigma$; $\hat{\sigma} = \sigma$; $\mathbf{\Psi}_D = [1, \dots, 1]^\intercal$; $\mathbf{\Psi}_R = [1, \dots, 1]^\intercal$; $\mathbf{\Psi}_L = [1, \dots, 1]^\intercal$
 2: repeat
 3:   select destroy and repair operators $\theta \in \mathcal{O}_D$ and $\rho \in \mathcal{O}_R$, and local search
      operator $\lambda \in \mathcal{O}_L$ using scores $\mathbf{\Psi}_D$, $\mathbf{\Psi}_R$ and $\mathbf{\Psi}_L$
 4:   $\eta = \eta + 1$ //increment iteration counter
 5:   $\sigma = \lambda(\rho(\theta(\bar{\sigma})))$
 6:   if accept($\sigma$,$\bar{\sigma}$) then
 7:     $\bar{\sigma} = \sigma$
 8:     if $\sigma$ is better than $\hat{\sigma}$ then
 9:       $\hat{\sigma} = \sigma$
10:     end if
11:   end if
12:   update($\mathbf{\Psi}_D$,$\mathbf{\Psi}_R$,$\mathbf{\Psi}_L$)
13: until $\eta = I$ //stop when reaching the maximum number of iterations, I
14: return $\hat{\sigma}$
```

6.4.1. Cost Functions and Acceptance Criteria

The acceptance criterion used in Line 6 of Algorithm 6.1 is based on the classical SA criterion: with a temperature τ and cost function $c(\sigma)$, a new solution σ is accepted with probability $\min\{1, e^{\frac{-(c(\bar{\sigma})-c(\sigma))}{\tau}}\}$, where $\bar{\sigma}$ is the current solution. Starting from an initial temperature τ_0, the temperature is gradually reduced every iteration by multiplying τ by a factor $\alpha \in (0, 1)$. Thus, the probability of accepting a new solution that is worse than the incumbent is gradually reduced.

In the ALNS for the hybrid cloud model, the objective function (6.1) is a natural choice as a cost function. However, the objective function of the private cloud model (6.16) leads to a fitness landscape consisting of large plateaus, and it would not be effective to use this function to drive the heuristic search. This problem is also observed in vehicle routing problems that minimize the number of vehicles as the primary objective. Pisinger and Ropke (2007) allow their ALNS for vehicle routing problems to work with infeasible solutions when minimizing the number of vehicles. Specifically, they allow some requests not to be served and penalize the unserved requests in the cost function. If a new solution with no unserved requests is found, the number of vehicles used in the future solutions is reduced by one. Similarly, we allow the ALNS to examine infeasible solutions by relaxing the node resource constraints (6.18), and penalize the surplus resource usage as the only term in the cost function. Thus, the set of nodes that can be used for placement is gradually reduced, and we denote the set of nodes available for placement for a solution σ as $\mathcal{N}(\sigma)$. Formally, with e_{ng} defined as the surplus usage of resource type $g \in \mathcal{G}$ on node $n \in \mathcal{N}(\sigma)$, the cost function of the ALNS in the private cloud case is given by:

$$c(\sigma) = \sum_{n \in \mathcal{N}(\sigma)} \sum_{g \in \mathcal{G}} e_{ng} \tag{6.20}$$

If a solution with $c(\sigma) = 0$ is found, the number of nodes allowed in the next solution is reduced by one. We denote solutions of the private cloud model that violate only the node resource constraints (6.18) as *over-utilized*, and include them in the set of feasible solutions.

6.4.2. Destroy Operators

We describe seven methods to destroy a solution. The first five consider only destruction of parts of the private cloud, the sixth method considers destruction of parts of both the private and public cloud, while the last method only removes replicas from the public cloud. The destruction of the private cloud is done in several ways, either by removal of all replicas on a subset of the nodes, removal of all active replicas on a subset of the nodes, removal of all passive replicas on a subset of the nodes, or removal of all replicas of a subset of the services. All methods take the current solution σ and a parameter $\delta \in (0, 1]$, the degree of destruction, as input. In addition, all but the first method (Random Node Removal) have a parameter $\gamma \in \mathbb{R}_+$ that controls the degree of randomization in the destruction. For brevity, we define $\mathcal{Q}_A(n)$, respectively $\mathcal{Q}_P(n)$, as the set of pairs (i, q) which have an active replica, respectively a passive replica, placed on node $n \in \mathcal{N}(\sigma)$ in the current solution σ.

To ease the exposition, we use the terms *displace* and *displacement* with the meaning of moving a replica either from a node in the private cloud or the public cloud to a set of unplaced active replicas or unplaced passive replicas.

6.4.2.1. Random Node Removal

This method randomly selects h nodes for removal, that is, all active and passive replicas at these h nodes are displaced. The number of nodes h to destroy is randomly drawn according to a uniform distribution among the integers in the interval $[2, \lfloor \delta |\mathcal{N}(\sigma)| \rfloor]$.

6.4.2.2. Active Replica Removal

The idea of this operator is to select h nodes and displace all active replicas on these nodes, while preserving the passive replicas. Instead of selecting the h nodes randomly, we want to bias the selection towards nodes having a specific feature, denoted by ϕ. The feature can either be low resource utilization as defined in equation (6.21) below, or low average public cloud cost of the active replicas running on the node as defined in equation (6.22). Only the first of these features is used in the ALNS for the private cloud model. Feature ϕ_A only considers the active part of the resource utilization of the nodes, that is, keeping all else fixed, it measures how well the node is utilized by active replicas.

$$\phi_A(n) = |\mathcal{G}|^{-1} \sum_{g \in \mathcal{G}} \frac{\sum_{(i,q) \in \mathcal{Q}_A(n)} G_{Aiqg}}{1 - \sum_{(i,q) \in \mathcal{Q}_P(n)} G_{Piqg} - m_{ng}} \tag{6.21}$$

$$\phi_C(n) = |\mathcal{Q}_A(n)|^{-1} \sum_{(i,q) \in \mathcal{Q}_A(n)} C_{Ciq} \tag{6.22}$$

The pseudocode of the Active Replica Removal, where the node selection is biased towards nodes with feature ϕ, is shown in Algorithm 6.2. The randomness of the algorithm is controlled by γ, and setting $\gamma = 1$ makes the node selection in Line 4 completely random. The parameter h is randomly and uniformly drawn among the integers in $[2, \lfloor 2\delta |\mathcal{N}(\sigma)| \rfloor]$.

6.4.2.3. Passive Replica Removal

The Passive Replica Removal operator only differs from the Active Replica Removal in Algorithm 6.2 at Line 5 where it displaces all passive replicas of node n, instead of all active replicas. In addition, the node vector is sorted in ascending

Algorithm 6.2 Pseudocode of the Active Replica Removal operator

```
Require: a solution $\sigma$, $h \in \mathbb{N}$, $\gamma \in \mathbb{R}_+$
1: Let $\boldsymbol{N}$ be a vector of the nodes in solution $\sigma$, sorted in ascending order of feature $\phi$
2: while $h > 0$ do
3:     $\pi$ = random number $\in [0, 1)$
4:     Select an element $n$ at position $\lfloor \pi^\gamma |\boldsymbol{N}| \rfloor$ of $\boldsymbol{N}$
5:     displace all active replicas of node $n$, and remove $n$ from $\boldsymbol{N}$
6:     $h = h - 1$
7: end while
8: return $\sigma$
```

order of the feature represented by ϕ_P, as defined in equation (6.23) below. Nodes running passive replicas with diverging requirements for the amount of the shared backup resource m_{ng}, and nodes running few passive replicas get a low value on ϕ_P. The rationale of this destroy operator is that it is beneficial that as many passive replicas as possible (bounded by N_P) share the reserved backup resources, and that the passive replicas running on the same node have quite similar resource requirements when being activated. The number of nodes, h, is drawn in the same way as in the active replica removal method.

$$\phi_P(n) = |\mathcal{G}|^{-1} \sum_{g \in \mathcal{G}} \frac{\sum_{(i,q) \in \mathcal{Q}_P(n)} (G_{Aiqg} - G_{Piqg})}{m_{ng} N_P} \tag{6.23}$$

6.4.2.4. Worst Node Removal

Like Random Node Removal, the Worst Node Removal operator selects h nodes and displaces all replicas running on the nodes. Thus, Algorithm 6.2 is adapted by changing Line 5 such that all replicas on the selected node are displaced. In this operator, the vector of nodes is ordered according to the feature ϕ_W as defined in (6.24) below. As long as there are passive replicas on the node, ϕ_W is a weighted combination of ϕ_A and ϕ_P, using weight parameter $\beta_W \in (0, 1)$. In the private cloud model, for a node n with over-utilized resources, i.e., $e_{ng} > 0$ for at least one g, ϕ_A takes a value larger than 1. In order not to favor this, we put an upper bound of 1 on ϕ_A.

$$\phi_W(n) = \begin{cases} (1 - \beta_W) \min\{1, \phi_A(n)\} + \beta_W \phi_P(n) & \text{if } \mathcal{Q}_P(n) \neq \emptyset \\ \phi_A(n) & \text{otherwise} \end{cases} \tag{6.24}$$

6.4.2.5. Over-utilized Node Removal

This operator is quite similar to Worst Node Removal, but is specifically designed for use in the private cloud model. It biases the node selection primarily towards nodes with large over-utilization, and secondarily towards nodes with large under-utilization. This is achieved by sorting the node vector according to feature ϕ_O, given in (6.25), where we define $G_g(n) = \sum_{(i,q)\in\mathcal{Q}_A(n)} G_{Aiqg} + \sum_{(i,q)\in\mathcal{Q}_P(n)} G_{Piqg} + m_{ng}$, that is, the current usage of resource g at node n. Observe that the expression is negated, and that the first term, the over-utilization, is given a large weight Ω_O, such that a node, say n_1, with the smallest possible over-utilization is ordered before a node, say n_2, with the largest possible under-utilization, i.e., $\phi_O(n_1) < \phi_O(n_2)$.

$$\phi_O(n) = -\sum_{g\in\mathcal{G}} \Big(\Omega_O \max\{0, G_g(n) - 1\} + \max\{0, 1 - G_g(n)\}\Big) \tag{6.25}$$

6.4.2.6. Related Service Removal

All the destroy operators considered so far select nodes, randomly or according to some feature, and displace all or some of the replicas on the selected nodes. The idea of the related service removal is to look for similarities among the worst nodes, specifically by considering the services that are present on the worst nodes. The operator is illustrated in Algorithm 6.3. Similar to the worst node removal, the vector of nodes is sorted according to feature ϕ_W, and h_N nodes are selected and added to a set of the worst nodes. However, instead of displacing the replicas of these nodes, the operator finds the h_S services that are represented with the most replicas at the worst nodes. Then all replicas of the h_S services are displaced. The parameters h_N and h_S are randomly and uniformly drawn among the integers in the intervals $[2, \lfloor\delta|\mathcal{N}(\sigma)|\rfloor]$ and $[1, \lfloor\delta|\mathcal{S}|\rfloor]$, respectively.

Since this operator displaces complete services, it is simple for the subsequent repair operator to change the replication pattern of the services involved. While this might be possible for repair operators following other destroy operators as well, we have not considered this. Therefore, the replication patterns can only change after calling the Related Service Removal.

6.4.2.7. Public Cloud Destruction

This operator, outlined in Algorithm 6.4, is the sole operator that only considers destruction of the public cloud placement. The parameter h, the number of active replicas to displace, is uniformly drawn among the integers in the interval $[5, \lfloor\delta_C|\boldsymbol{Q}_C(\sigma)|\rfloor]$, where δ_C is the degree of destruction, and $\boldsymbol{Q}_C(\sigma)$ is a vector of the active replicas currently placed in the public cloud. The displacement of repli-

Algorithm 6.3 Pseudocode of the Related Service Removal operator

```
Require: a solution $\sigma$, $h_N \in \mathbb{N}$, $h_S \in \mathbb{N}$, $\gamma \in \mathbb{R}_+$
 1: Let $\boldsymbol{N}$ be a vector of the nodes in solution $\sigma$, sorted in ascending order of
    feature $\phi_W$
 2: Let $\mathcal{N}_W = \emptyset$ be the set of the worst nodes
 3: while $h_N > 0$ do
 4:     $\pi$ = random number $\in [0, 1)$
 5:     Select an element $n$ at position $\lfloor \pi^\gamma |\boldsymbol{N}| \rfloor$ of $\boldsymbol{N}$
 6:     remove $n$ from $\boldsymbol{N}$, and add $n$ to $\mathcal{N}_W$
 7:     $h_N = h_N - 1$
 8: end while
 9: Let $\mathcal{S}_W$ be the set of services with cardinality $h_S$ that are represented with
    most active and passive replicas on the nodes in $\mathcal{N}_W$
10: Displace all replicas of the services in $\mathcal{S}_W$, from both the private cloud and
    the public cloud
11: return $\sigma$
```

cas is biased towards replicas with high cost, C_{Ciq}, and the bias is still controlled by the parameter γ.

Algorithm 6.4 Pseudocode of the Public Cloud Destruction operator

```
Require: a solution $\sigma$, $h \in \mathbb{N}$, $\gamma \in \mathbb{R}_+$
 1: Let $\boldsymbol{Q}_C$ be a vector of the active replicas $(i, q)$ currently placed in the public
    cloud in solution $\sigma$, sorted in descending order of public cloud cost $C_{Ciq}$
 2: while $h > 0$ do
 3:     $\pi$ = random number $\in [0, 1)$
 4:     Select an element $(i, q)$ at position $\lfloor \pi^\gamma |\boldsymbol{Q}_C| \rfloor$ of $\boldsymbol{Q}_C$
 5:     displace active replica $(i, q)$ from the public cloud
 6:     $h = h - 1$
 7: end while
 8: return $\sigma$
```

This operator is only used in the ALNS for the hybrid cloud model, and it is used in combination with one of the other operators that consider destruction of the private cloud, except from the Related Service Removal. When this operator is called in combination with the Random Node Removal, $\gamma = 1$, that is, the displacement of replicas from the public cloud is completely random; otherwise γ takes the same value as the other destroy operators.

6.4.3. Repair Operators

We use three types of repair operators. The first two are fast and insert unplaced replicas one at a time, based on simple measures. They do not concentrate on rebuilding the nodes individually, like a strategy that solves several knapsack problems in sequence, but consider every node as a potential point of placement, as long as the insertion of a replica at the node is feasible. The last repair operator is based on solving a reduced version of the direct MIP models, where a large number of the variables is fixed to integer values. Even though the MIP is greatly reduced, the operator is more time-consuming than the insertion operators. However, if one gives the operator enough time, it will find the optimal way to repair the destroyed solution.

6.4.3.1. Greedy Insertion

The Greedy Insertion operator is a fast and simple constructive heuristic, which iteratively places the unplaced replicas on the nodes guided by a cost measure. Specifically, for the ALNS of the hybrid cloud model, if no more active replicas can be placed on the nodes, the rest of the unplaced active replicas are placed in the public cloud. However, if there are remaining unplaced passive replicas, the operator returns without any feasible solution. Even though the ALNS of the private cloud model allows over-utilized nodes in the solutions, it is possible that the greedy insertion cannot find a feasible placement for all unplaced active or passive replicas, and thus, returns without any feasible solution.

The cost measures used to guide the greedy insertion heuristic in the ALNS for the hybrid cloud model and the private cloud model differ. This is natural since the objective functions in the mathematical formulations also differ. Furthermore, the cost measures used to guide the insertion of active replicas and passive replicas differ, and they are not directly comparable. Therefore, the Greedy Insertion heuristic first tries to insert all passive replicas, and then focuses on inserting all active replicas.

In the hybrid cloud model, we let $\xi_P(i, q, n)$, defined in (6.26), denote the cost of inserting a passive replica of service component pair (i, q) at node n. The cost accounts for the absolute deviation between the current reserved backup resources, m_{ng}, at the node and the replica's requirement for backup resources. However, if the insertion is infeasible, we set $\xi_P(i, q, n) = \infty$. Analogously, we define $\xi_A(i, q, n)$, according to (6.27), as the cost of inserting an active replica of pair (i, q) at node n. The first term of (6.27), which is given a large weight Ω_C, accounts for the public cloud cost that would incur if the replica is not placed in the public cloud, while the second term accounts for the residual resource slack at the node after insertion (less is better). Since active replicas with large public cloud cost should have a smaller insertion cost $\xi_A(i, q, n)$, we subtract C_{Ciq} from a constant $\bar{C}_C$ which is greater than $\max_{(i,q)} C_{Ciq}$.

$$\xi_P(i,q,n) = \sum_{g \in \mathcal{G}} |m_{ng} - (G_{Aiqg} - G_{Piqg})| \tag{6.26}$$

$$\xi_A(i,q,n) = \Omega_C(\bar{C}_C - C_{Ciq}) + \sum_{g \in \mathcal{G}} \Big(1 - G_g(n) - G_{Aiqg}\Big) \tag{6.27}$$

The cost measures are adapted to (6.28) and (6.29) in the ALNS for the private cloud model. Since the overall cost function used by the acceptance criteria (see Section 6.4.1) concerns the amount of over-utilized resources, both cost measures below take this into account. We let $\Delta e_{Pg}(i,q,n)$, respectively $\Delta e_{Ag}(i,q,n)$, denote the increase in over-utilization (of resource type g) when a passive, respectively an active, replica of pair (i,q) is inserted on node n; this increase in over-utilization is given a large weight Ω_I in the cost measures.

$$\xi_P(i,q,n) = \Omega_I \sum_{g \in \mathcal{G}} \Delta e_{Pg}(i,q,n) + \sum_{g \in \mathcal{G}} |m_{ng} - (G_{Aiqg} - G_{Piqg})| \tag{6.28}$$

$$\xi_A(i,q,n) = \Omega_I \sum_{g \in \mathcal{G}} \Delta e_{Ag}(i,q,n) + \sum_{g \in \mathcal{G}} \Big(1 - G_g(n) - G_{Aiqg}\Big) \tag{6.29}$$

In a greedy fashion the operator first selects the passive replica among all unplaced replicas with minimum cost, and inserts this replica on its minimum cost node. This insertion process repeats until all passive replicas are inserted, or there are no more feasible insertions to do. In the latter case, the operator would return without a feasible solution. Assuming that all passive replicas were placed, the operator considers the unplaced active replicas in the same way. In the ALNS for the hybrid cloud model, if there are unplaced active replicas left after the insertion process, these replicas are placed in the public cloud. On the other hand, in the private cloud model the operator would return without a feasible solution unless all active replicas are inserted.

6.4.3.2. Regret Insertion

The Regret Insertion is designed to be less myopic than the Greedy Insertion in the insertion strategy. The operator extends the Greedy Insertion by being guided by a regret value, or look-ahead value, instead of the cost measure directly. For a passive (active) replica of pair (i,q), let n_{iqk} be the node with kth lowest cost according to cost measure $\xi_P(i,q,n)$ ($\xi_A(i,q,n)$). Moreover, the regret-k value $\zeta_{iq}(k)$ for a passive replica is defined as in (6.30). The regret-k value for active replicas is defined analogously using the cost measures $\xi_A(i,q,n)$ instead.

$$\zeta_{iq}(k) = \sum_{j=2}^{k} \Big(\xi_P(i,q,n_{iqj}) - \xi_P(i,q,n_{iq1})\Big) \tag{6.30}$$

The insertion procedure of the Regret Insertion is similar to that of the Greedy Insertion, except that instead of selecting the replica with lowest cost of insertion in each iteration, the replica with the *maximum* regret value is selected. If a replica has fewer than k nodes where an insertion is feasible, the regret value of the replica will be ∞. Hence, the replica is selected for insertion early in the process. This means that regret insertion operators with high k have greater probability of finding a feasible solution than both regret insertion operators with low k and the Greedy Insertion.

6.4.3.3. MIP Insertion

The MIP Insertion is based on the idea of calling a general-purpose solver on the MIP models presented in Section 6.2, where a part of the variables is fixed. In a given iteration of the ALNS, the fixed variables correspond to the non-destroyed part of the solution. Thus, when the MIP Insertion is used to repair the solution, one optimizes the MIP model over the variables of the replicas in the sets of unplaced active and passive replicas. If the Related Service Removal is used for destroying the solution prior to calling the MIP Insertion, the latter also optimizes over the replication pattern variables, y_{ir}, of the displaced services.

For the hybrid cloud MIP model of Section 6.2, the objective function of the MIP Insertion is changed to (6.31) by adding a second term considering the shared resources reserved for activation of passive replicas, m_{ng}. This is done to ensure that the repair operator inserts passive replicas wisely, even if the insertion does not affect the public cloud cost. However, the public cloud cost term is given a large weight, Ω_C, such that it dominates the expression.

$$\min z = \Omega_C \sum_{i\in\mathcal{S}} \sum_{q\in\mathcal{Q}_i} C_{Ciq} w_{Ciq} + \sum_{n\in\mathcal{N}(\sigma)} \sum_{g\in\mathcal{G}} m_{ng} \tag{6.31}$$

The MIP Insertion used in the ALNS for the private cloud model has to consider that the resource constraints of the nodes are relaxed. Therefore, the original node resource constraints (6.18) are rewritten as (6.33), where e_{ng} accounts for the resource usage surplus at the nodes, and defined in (6.34). Similar to the overall objective of the ALNS, the MIP model seeks to minimize the total resource usage surplus of the nodes, hence the objective of the MIP Insertion is written as (6.32).

$$\min z = \sum_{n\in\mathcal{N}(\sigma)} \sum_{g\in\mathcal{G}} e_{ng} \tag{6.32}$$

$$\sum_{i\in\mathcal{S}} \sum_{q\in\mathcal{Q}_i} G_{Aiqg} w_{iqn} + \sum_{i\in\mathcal{S}} \sum_{q\in\mathcal{Q}_i} G_{Piqg} v_{iqn} + m_{ng} - e_{ng} \leq 1 \quad \forall n \in \mathcal{N}(\sigma), \forall g \in \mathcal{G} \tag{6.33}$$

$$e_{ng} \geq 0 \quad \forall n \in \mathcal{N}(\sigma), \forall g \in \mathcal{G} \tag{6.34}$$

6.4.4. Local Search Operators

Two classes of local search operators are used in the ALNS: a swap of H_1 replicas from node n_1 with H_2 replicas from node n_2, denoted as the inter-node swap; and a swap of H_1 replicas from node n with H_2 replicas currently placed in the public cloud, denoted as the inter-cloud swap. By using different values for the operator parameters H_1 and H_2, several different swap operators are considered by the ALNS.

6.4.4.1. Inter-node Swap

The Inter-node Swap operator either swaps H_1 active replicas and H_2 active replicas between two nodes, or swaps two sets of H passive replicas between two nodes. An inter-node swap is defined by the tuple $(\bar{\mathcal{Q}}_1, \bar{\mathcal{Q}}_2, n_1, n_2)$, where $\bar{\mathcal{Q}}_1$ and $\bar{\mathcal{Q}}_2$ are sets of replicas and n_1 and n_2 are the nodes where the replicas $\bar{\mathcal{Q}}_1$ and $\bar{\mathcal{Q}}_2$, respectively, currently are placed. After performing the swap, the replicas in $\bar{\mathcal{Q}}_2$ run on n_1, while the replicas in $\bar{\mathcal{Q}}_1$ run on n_2.

The operator iteratively searches for swaps that directly or indirectly improve the solution quality. In each iteration, the operator performs an improving swap, and then continues the search. The improvement of a swap is measured by a cost function, ψ. Searching through all feasible swaps in each iteration would make the operator computationally expensive. Therefore, in each iteration, the operator stops the search if an improving swap is found after having searched through all feasible swaps between a given node and any other node. At the end of each iteration, the operator performs the most improving swap found in this iteration. Algorithm 6.5 outlines the operation of the inter-node swap operator. In Line 8, all swaps between nodes n_1 and n_2 that lead to a feasible solution are evaluated by a cost function, and in Line 10, the most improving swap is saved. When $H_1 = H_2$, we exploit the symmetry by letting n_2 iterate over the nodes that are ordered after n_1 in a fixed ordering (see Line 7). However, when $H_1 \neq H_2$, we also need to consider swaps of H_2 replicas from lower-ordered nodes with H_1 from higher-ordered nodes.

Algorithm 6.5 Pseudocode of the inter-node swap operator

```
Require: a solution $\sigma$, $H_1 \in \mathbb{N}$, $H_2 \in \mathbb{N}$
 1: //let $(\hat{\mathcal{Q}}_1, \hat{\mathcal{Q}}_2, n_1, n_2)$ denote the most improving swap in an iteration, and let
    $\hat{\psi}$ denote the minimum cost
 2: let the set $\tilde{\mathcal{N}}$ contain all nodes in the solution $\sigma$, $\mathcal{N}(\sigma)$
 3: while true do
 4:   $\hat{\psi} = 0$ // set the minimum cost to zero
 5:   while $\tilde{\mathcal{N}} \neq \emptyset$ do
 6:     remove a random element $n_1$ from $\tilde{\mathcal{N}}$
 7:     for all $n_2 \in \mathcal{N}(\sigma) : n_2$ ordered higher than $n_1$ do
 8:       find the most improving swap $(\bar{\mathcal{Q}}_1, \bar{\mathcal{Q}}_2, n_1, n_2)$ of $H_1$ replicas from $n_1$
          with $H_2$ replicas from $n_2$
 9:       if $\psi(\bar{\mathcal{Q}}_1, \bar{\mathcal{Q}}_2, n_1, n_2) < \hat{\psi}$ then
10:         $\hat{\psi} = \psi(\bar{\mathcal{Q}}_1, \bar{\mathcal{Q}}_2, n_1, n_2)$ and $(\hat{\mathcal{Q}}_1, \hat{\mathcal{Q}}_2, n_1, n_2) = (\bar{\mathcal{Q}}_1, \bar{\mathcal{Q}}_2, n_1, n_2)$
11:       end if
12:     end for
13:     if $\hat{\psi} < 0$ then break //improving swap found
14:   end while
15:   if $\hat{\psi} = 0$ then break //no improving swap found
16:   perform swap $(\hat{\mathcal{Q}}_1, \hat{\mathcal{Q}}_2, n_1, n_2)$ //update $n_1, n_2 \in \mathcal{N}(\sigma)$
17:   add nodes $n_1$ and $n_2$ to $\tilde{\mathcal{N}}$ //update $n_2$ if $n_2$ already exists in $\tilde{\mathcal{N}}$
18: end while
19: return $\sigma$
```

We use different cost functions based on whether the operator swaps active or passive replicas, and on whether the ALNS considers the hybrid or the private cloud model. In the ALNS for the hybrid cloud model, we only consider inter-node swaps of passive replicas, and use the cost function ψ_P defined in (6.35). The functions $\Delta m_g(\bar{\mathcal{Q}}_1, \bar{\mathcal{Q}}_2, n_1)$ account for the increase or decrease in the shared backup resources reserved for passive replicas (of resource type g) at node n_1, when the set $\bar{\mathcal{Q}}_1$ of passive replicas are removed from the node and the set $\bar{\mathcal{Q}}_2$ are placed on the node. Thus, ψ_P in (6.35) computes the total change in the resources reserved for passive replicas. In the ALNS for the private cloud model, it is more natural to look at the increase or decrease in over-utilization of the resources incurred by the swap. Thus, the cost functions measuring the improvement of a swap is simply the difference in the cost function of the ALNS, given in (6.20), before and after the swap.

$$\psi_P(\bar{\mathcal{Q}}_1, \bar{\mathcal{Q}}_2, n_1, n_2) = \sum_{g \in \mathcal{G}} \Big(\Delta m_g(\bar{\mathcal{Q}}_1, \bar{\mathcal{Q}}_2, n_1) + \Delta m_g(\bar{\mathcal{Q}}_2, \bar{\mathcal{Q}}_1, n_2) \Big) \tag{6.35}$$

6.4.4.2. Inter-cloud Swap

The behavior of the Inter-cloud Swap operator is similar to the Inter-node Swap operator, but it considers swaps of replicas between a node in the private and the public cloud, instead of swaps between two nodes. Therefore, this operator is solely used in the ALNS for the hybrid cloud model. Moreover, since only active replicas run in the public cloud, this operator considers exclusively swaps of H_1 active replicas with H_2 active replicas. Like the Inter-node Swap, the Inter-cloud Swap tries to limit the search in each iteration by stopping after all feasible swaps between a given node n and the public cloud have been evaluated, as long as one or more improving swaps are found, and then performs the most improving swap. The cost function ψ_A for evaluating the quality of a swap is defined in (6.36). The function computes the change in public cloud cost when a set $\bar{\mathcal{Q}}_1$ of active replicas at a node n is swapped with a set $\bar{\mathcal{Q}}_2$ of active replicas currently placed in the public cloud. An improving swap is identified by a negative cost.

$$\psi_A(\bar{\mathcal{Q}}_1, \bar{\mathcal{Q}}_2) = \sum_{(i,q) \in \bar{\mathcal{Q}}_1} C_{Ciq} - \sum_{(i,q) \in \bar{\mathcal{Q}}_2} C_{Ciq} \tag{6.36}$$

6.4.5. Adaptive Operator Selection

In the previous sections, we have defined several destroy operators, repair operators and local search operators. In a given iteration of the ALNS, we only use one operator of each type, and we need a way to decide which operators to use in each iteration (see Line 3 of Algorithm 6.1). The traditional way of selecting the operators in an ALNS is to select the destroy and repair operators independently by using a roulette wheel selection strategy where each of the operators is assigned weights.

In applications where the repair operators are similar in complexity and time consumption, independent selection of the operators seems to be unproblematic. However, in our ALNS where a relatively expensive repair operator, i.e., the MIP insertion, and other less accurate, but faster, repair operators are used side by side, there might be something to gain by coupling the selection of destroy and repair operators. We first select the destroy operator based on the traditional method in ALNS, i.e., a destroy operator $\hat{\theta}$ is selected with a probability according to (6.37), where $\Psi_{D\theta}$ represents the weight of the destroy operator $\theta \in \mathcal{O}_D$. Moreover, each destroy operator, θ, is associated with a set of repair operators that may be called successively; we denote this set by $\mathcal{O}_{R\theta} \subseteq \mathcal{O}_R$. The idea is to couple the selection of destroy and repair operators, so that some destroy operators have to be followed by a call to the MIP insertion operator, while other operators have to be followed by a less accurate greedy or regret insertion operator. The probability of selecting a repair operator $\hat{\rho} \in \mathcal{O}_{R\hat{\theta}}$ to follow the destroy operator $\hat{\theta}$ is given in (6.38). At last, the local search operator is chosen in the same way as the destroy operator, that is, operator $\hat{\lambda}$ is selected with a probability according to (6.39).

$$\frac{\Psi_{D\hat{\theta}}}{\sum_{\theta \in \mathcal{O}_D} \Psi_{D\theta}} \tag{6.37}$$

$$\frac{\Psi_{R\hat{\rho}}}{\sum_{\rho \in \mathcal{O}_{R\hat{\theta}}} \Psi_{R\rho}} \tag{6.38}$$

$$\frac{\Psi_{L\hat{\lambda}}}{\sum_{\lambda \in \mathcal{O}_L} \Psi_{L\lambda}} \tag{6.39}$$

Generally, the underlying adaptiveness of the ALNS is provided by the *adaptive weight adjustment* principle (Ropke and Pisinger, 2006). The intent is that one could include several destroy and repair operators that function well for different problem instances and different problem structures in the sets of operators, and adjust their weights dynamically based on their past performance. In each iteration of the ALNS, the quality of the solution found and the time spent in the iteration, that is essentially the time spent in Line 5 of Algorithm 6.1, are used to score the operators used in that iteration. The new solution is classified according to whether it is *a new best solution*, *better than the incumbent solution*, *accepted as a new incumbent*, or *not accepted*; based on this, the operators get a *basic score* ν_1, ν_2, ν_3, or ν_4, where $\nu_1 > \nu_2 > \nu_3 > \nu_4$. The lowest score, ν_4, is defined as 1. Since the different repair operators might use a significantly different amount of time to repair the destroyed solution, we have chosen to consider the time spent in each iteration in the scoring of the operators. Especially, we are interested in reducing the scores in iterations which are time-consuming. Therefore, we compute a *time-normalized score* shown in (6.40), where $j \in \{1, 2, 3, 4\}$ according to the classification of the new solution, T_σ denotes the time spent in the iteration, and $\bar{T}$ refers to the average time spent in an iteration since the beginning of the search. Moreover, the normalization can be controlled by the parameter ω. The outer max-term in the normalization ensures that no operators are given a score worse than ν_4, while the inner min-term makes the scoring only penalize long iterations, and not iterations that are shorter than the average. Moreover, if the repair method is not capable of finding a feasible solution, the operator gets score ν_4.

$$\max\{\nu_4, \nu_j \min\{1, \bar{T}/(\omega T_\sigma)\}\} \tag{6.40}$$

The search is divided into segments that correspond to 100 iterations. At the beginning of each segment, the scores of all operators are set to one, and throughout the segment, the operators accumulate the time-normalized scores from the iterations where they were used. At the end of a segment, the weights of all operators are updated based on their accumulated scores (Line 12 of Algorithm 6.1). The destroy operator's weights are updated as shown in (6.41), where $\varsigma_{D\theta}$ represents the accumulated scores of destroy operator θ, and $\Phi_{D\theta}$ is a count of the number of times operator θ was called in the last segment. The parameter β_R determines how quickly the past performance is reset in the adaptive weight adjustment method. The repair and local search operators are updated in the same manner.

$$\Psi_{D\theta} = (1 - \beta_R)\Psi_{D\theta} + \beta_R \frac{\varsigma_{D\theta}}{\Phi_{D\theta}} \tag{6.41}$$

6.4.6. Initial Solution

The construction of the initial solution is done in two steps. First, a replication pattern is selected for each service. Then, the Regret Insertion with $k = 3$ is called to insert all active and passive replicas. Several heuristic rules can potentially be used to select the replication patterns. In the implementation, we have selected the replication patterns that minimize the total resource assigned to all the active replicas in each service. Moreover, for the private cloud model, unless a strict natural upper bound on the number of nodes exists, one has to set a bound. Setting this bound too low might result in a situation where the regret insertion cannot find a feasible solution. If this is the case, one can increase the number of nodes and call the regret insertion until a feasible solution is found.

6.4.7. Summary of the Operators

Several destroy and repair operators were presented in Sections 6.4.2 and 6.4.3, respectively. However, not all of them can be or are used in both the ALNS for the hybrid cloud model and the ALNS for the private cloud model. Moreover, the selection of a repair operator is presented as coupled to the selected destroy method. To achieve the coupled selection, we have grouped destroy and repair operators together, in two groups. The groups for both the hybrid cloud and private cloud cases are shown in Table 6.1. Keep in mind that the ALNS for the hybrid cloud model also calls the Public Cloud Destruction operator in each iteration, except when the Related Service Removal is used. These groups imply that if Related Service Removal is selected as the destroy operator, the MIP Insertion will be selected as the repair operator. In method group (ii), there is the Greedy Insertion operator and one or more Regret Insertion operators with different k. The different values of k used are decided in the tuning of the algorithm, which is discussed in Section 6.5.1. Moreover, in the private cloud case, we use two instances of the Over-utilized Node Removal, one for the MIP Insertion and one for the other methods. The two instances might consider different degrees of destruction δ.

Table 6.1.: Grouping of destroy and repair operators

	Hybrid Cloud		Private Cloud	
Method Group	**Repair Operators**	**Destroy Operators**	**Repair Operators**	**Destroy Operators**
(i)	MIP Insertion	Related Service Removal	MIP Insertion	Related Service Removal
		Worst Node Removal		Over-utilized Node Removal
(ii)	Greedy Insertion	Random Node Removal	Greedy Insertion	Random Node Removal
		Passive Replica Removal		Passive Replica Removal
	Regret Insertion	Active Replica Removal-ϕ_A	Regret Insertion	Active Replica Removal-ϕ_A
		Active Replica Removal-ϕ_C		Over-utilized Node Removal

Regarding the local search operators, the ALNS for the hybrid cloud model uses one or more Inter-node Swap operators for passive replicas, and one or more Inter-cloud Swap operators for active replicas, where the values of H_1 and H_2 vary. The ALNS for the private cloud model uses Inter-node Swap operators for both active and passive replicas with different values of H_1 and H_2. The combinations of H_1 and H_2 used in the swaps are decided in the tuning process, which is described in Section 6.5.1.

6.5. Computational Study

The purpose of the computational study is twofold. We present the results of an evaluation of the ALNS with and without local search (LS) operators. In addition, we perform a comparison of the ALNS with the branch and price (B&P) algorithm proposed by Gullhav and Nygreen (2015b). Before presenting the results, we give some details on the setup of the experiments.

6.5.1. Setup of the Experiments and Tuning

The ALNS was implemented in C++, and compiled with GCC 4.8.2 with optimization option `-O3`. We have run all our experiments on a CentOS 5.8 machine with a dual core 3.0 Ghz Intel E5472 Xeon processor and 16 GB of memory. The MIP Insertion operator calls the Xpress-Optimizer version 27.01.02 of the FICO

Xpress Optimization Suite 7.8. The MIP solver of Xpress has utilized up to eight threads in the tree search.

We have designed an instance generator which constructs test cases that aim to be as realistic as possible. The test cases used range from 20 to 70 services, and each service is composed of four components on average, which implies that the largest cases contain a total of 280 components. The instance generator is based on ten different dummy services composed of between three and five components each. The different services have a structure reflecting real-world services with different resource requirements for the different components. In short, each component of the ten services is given a resource requirement distribution, and the case generation is based on drawing resource requirements from these distributions using different seeds. For each instance size, we have generated five cases for testing, and for the instances with 40 and 50 services, we have generated another five for tuning purposes. In all cases, $N_S = 3$ and $N_P = 4$, and we consider CPU resources only. The minimum, average, and maximum values on the G_{Aiqg} parameters are 8.0%, 23.0%, and 45.0%, respectively, while the same values for the G_{Piqg} parameters are 0.33%, 1.6%, and 3.0%. Over all cases, the average number of replication patterns per service is 7.5.

A difference between the test cases for the private cloud model and hybrid cloud model is that the latter have an (effective) upper bound on the number of nodes in the private cloud, N_N. This upper bound is based on a lower bound (LB) on the number of nodes needed to place all services in the private cloud. This lower bound is computed as the best bound provided by the B&P algorithm (Gullhav and Nygreen, 2015b). Using this LB, we constructed two types of test cases for the hybrid cloud: one with $N_N = \lceil 0.75\text{LB} \rceil$, and one with $N_N = \lceil 0.9\text{LB} \rceil$. We say that these two types of test cases have 75% and 90% *private cloud coverage*. Furthermore, the placement of active replicas in the public cloud is done at a cost, and the offered VM types used are presented in Table 6.2. In the table, we list two providers. Since we model the public cloud as a generic pool of resources, we select the VM types for the different components as the cheapest one with enough capacity. The costs are synthetic and not given units. However, the relative cost and size between the VM types reflect the real world.

Table 6.2.: Data of the public cloud VM types used in the hybrid cloud experiments. The capacity is given in percentage of the private cloud node capacity.

Provider 1		**Provider 2**	
Cost	**Capacity**	**Cost**	**Capacity**
10	10%	15	15%
20	20%	30	30%
40	40%	60	60%

The test cases are named based on the number of services they contain. The hybrid cloud test cases with 20 services are referred to as `H20`, while the corresponding test cases with no restriction on the number of nodes are labeled `P20` and, hence, used for testing the private cloud model. Whenever it is necessary to identify the five different test cases with a given number of services, the cases are appended a letter from 'a' to 'e', i.e., `H20-a` to `H20-e`.

The ALNS has several parameters that can be tuned for the problem structure. We used SMAC (Hutter et al., 2011) to tune the parameters listed in Table 6.3. The tuning was done on separate test cases with 40 and 50 services, and we set the parameters of the ALNS for the hybrid cloud model and the ALNS for the private cloud model independently. Thus, for the hybrid cloud model we tuned the parameters over 20 cases: ten with 40 services and ten with 50 services. For each size, five of the cases had a private cloud coverage of 75%, while the other five had a private cloud coverage of 90%. The tuning was set up to evaluate the relative gap between the best LB produced by the B&P on the respective test case and the objective value of ALNS after 900 seconds. Moreover, SMAC was run with eight processes in parallel. The processes shared data during the run, but outputted eight different parameter combinations.

Most of the parameters in Table 6.3 are defined in Section 6.4. However, the parameters of the acceptance criteria, τ_0 and α, are not tuned directly. These two parameters are set based on the tuned parameters τ_C and P_E, in addition to the maximum number of iterations I and the initial solution σ_0 of a given run. In the ALNS for the hybrid cloud model, the initial temperature τ_0 is set so that a solution with τ_C% higher cost than σ_0 would be accepted with 50% probability. The cooling rate α is set so that after cooling the temperature I times, a solution with τ_C% higher cost than σ_0 would be accepted with probability P_E. For the private cloud model, the temperature and cooling rate is reset in every iteration following a new best solution, since a new best solution has zero cost, i.e., no over-utilization. The reset procedure is similar to the procedure setting the initial temperature and cooling rate for the hybrid cloud model, but when called, it uses the cost of the current solution and the remaining number of iterations.

Regarding the tuning of the Regret Insertion operators, Table 6.3 shows that the maximum k of the operators is 4. This means that we use three Regret Insertion operators with k equal to 2, 3 and 4. The tuning of the swap operators is not shown in the table. However, for both the hybrid cloud and private cloud model, the best combination of Inter-node Swaps of passive replicas were $(H_1, H_2) \in \{(1,1),(2,2)\}$, i.e., swap of one replica with another and swap of two replicas with two other replicas. For the Inter-cloud Swap of active replicas in the hybrid cloud case, the best swap types were $(H_1, H_2) \in \{(1,1),(1,2),(2,1),(2,2)\}$. In the private cloud case, the best Inter-node Swaps of active replicas were $(H_1, H_2) \in \{(1,1),(1,2),(2,2)\}$. In this case, the swaps $(1,2)$ and $(2,1)$ are identical.

Table 6.3.: Overview of the tuned parameters and their respective values in the algorithm for the private cloud model (PCM) and hybrid cloud model (HCM).

Parameter	**Description**	**Parameter domain**	**Value in**	
			PCM	**HCM**
δ^M	Deg. of destruction before calling MIP Insertion	$\{0.05, 0.10, 0.15, \dots, 0.5\}$	0.05	0.05
δ	Deg. of destruction before calling other repair operators	$\{0.05, 0.10, 0.15, \dots, 0.5\}$	0.05	0.1
δ_C^M	Deg. of destr. (public cloud) before calling MIP Insertion	$\{0.1, 0.2, \dots, 1\}$	N/A	0.5
δ_C	Deg. of destr. (public cloud) before calling other repair ops.	$\{0.1, 0.2, \dots, 1\}$	N/A	1.0
γ	Randomness in destroy operators	$\{2, 3, \dots, 8\}$	3	4
β_W	Weight parameter in the Worst Node Removal	$\{0.1, 0.2, \dots, 0.9\}$	0.4	0.1
K	Maximum value of k in the Regret Insertion operators	$\{2, 3, 4, 5\}$	4	4
(ν_1, ν_2, ν_3)	Basic scores in the adaptive operator selection	$\{1, 2, \dots, 50\}$	(37,32,9)	(42,31,22)
ω	Time normalization parameter in the adaptive op. selection	$\{0.5, 0.6, \dots, 1.5\}$	0.6	1.1
β_R	Weight put on the past performance in the scoring	$\{0.05, 0.10, 0.15, \dots, 0.5\}$	0.15	0.5
τ_C	Parameter controlling the temperature initialization	$\{0.1, 0.5, 1, 2.5, 5, 10, 20\}$	1	0.1
P_E	Prob. (in %) of accepting a sol. τ_C% worse than the initial sol.	$\{0.01, 0.05, 0.1, 0.25, 0.5\}$	0.1	0.1

6.5.2. Results

First, we are interested in evaluating the effect of using an LS operator on top of the repair operators in the ALNS. In the evaluation, we use the best LB obtained by the B&P algorithm to compute relative gaps between the LB and the objective value of the best solution obtained by the ALNS with and without LS operators. The LBs for the private cloud cases have been shown to be very tight in the cases that are solved to optimality. The private cloud model has structural similarities with the cutting stock problem, for which column generation provides very tight bounds (Vanderbeck, 1999). However, we cannot be certain about the quality of the LBs in the hybrid cloud cases. Table 6.4 gives the average relative gaps at different points in time, after 5, 10 and 15 minutes, for the hybrid cloud cases with 75% and 90% private cloud coverage. We see that on the smaller cases, the performance of the two versions of the ALNS is quite similar. However, when the problem size grows, there seems to be a benefit in using the LS on top of the repair operators. In addition, the benefit is more pronounced for the cases with 90% private cloud coverage, than for the cases with 75% private cloud coverage. As previous studies have shown, the cases with higher private cloud coverage are more difficult to solve and have larger gaps (Gullhav and Nygreen, 2015a,b). This might imply that the LS becomes more valuable as the difficulty of the problem increases. Table 6.5 displays the gaps for the private cloud cases at 5, 10 and 15 minutes of run time. We can see that the relative gaps are smaller than in the hybrid cloud cases, but the ALNS with the LS operators still gives the best results. Furthermore, we also see that when the problem size grows, the LS becomes more beneficial.

Table 6.4.: Average relative gap (in %) between best solution found and best bound at different points in time (seconds): comparison of the ALNS with and without local search (LS) operators on the hybrid cloud cases.

	75% private cloud coverage						**90% private cloud coverage**					
	ALNS w/o LS			**ALNS with LS**			**ALNS w/o LS**			**ALNS with LS**		
	300	600	900	300	600	900	300	600	900	300	600	900
H20	7.940	6.672	5.938	8.001	6.448	5.639	20.62	18.03	16.42	20.47	17.53	16.41
H30	10.79	8.998	8.013	10.02	8.632	7.859	28.35	24.69	22.41	26.32	22.24	21.07
H40	12.75	11.11	9.781	11.75	10.26	9.397	36.23	29.58	26.92	27.97	24.24	22.49
H50	17.33	14.59	13.27	15.82	12.95	12.17	45.09	36.99	33.53	38.05	31.73	29.20
H60	19.12	15.91	14.63	17.63	14.56	13.52	55.93	43.37	38.11	47.62	37.40	32.86
H70	22.68	19.64	17.29	21.47	17.93	15.66	60.51	49.82	44.31	55.53	42.92	37.26

While Tables 6.4 and 6.5 present average values for each instance size, Table 6.6 presents the p-values of the Wilcoxon signed-rank test (Hollander et al., 2013).

Table 6.5.: Average relative gap (in %) between best solution found and best bound at different points in time (seconds): comparison of the ALNS with and without local search (LS) operators on the private cloud cases.

	ALNS w/o LS			**ALNS with LS**		
	300	600	900	300	600	900
P20	2.102	2.102	2.102	2.111	1.798	1.528
P30	3.113	2.339	2.339	2.339	1.953	1.953
P40	4.395	3.662	3.074	2.784	2.341	2.341
P50	5.913	5.085	4.260	3.551	3.197	2.959
P60	6.932	5.841	5.349	4.260	3.468	3.266
P70	8.142	6.954	6.277	5.092	4.243	3.987

This non-parametric statistical hypothesis test is used to compare the relative gaps of the ALNS with and without the LS operators, with the aim to reject the null hypothesis. The null hypothesis states that the difference between the algorithms follows a symmetric distribution around zero, and we test this against the two-sided alternative hypothesis. Moreover, in the test, we have grouped the H20 and H30 cases, the H40 and H50 cases, and the H60 and H70 cases together (analogously for the private cloud cases) to obtain larger test samples. For the hybrid cloud cases, we have jointly performed the test over the cases with 75% and 90% private cloud coverage. With a significance level of 5%, we can reject the null hypothesis in all cases in Table 6.6 where the p-value is less than 0.05. From these results, we can see that we cannot reject the null hypothesis with a significance level of 5% for the H20 and H30 cases. However, when the instance size increases, the differences between the two algorithms are significant and in favor of the ALNS with the LS operators. For the private cloud cases, we can see that the difference between the algorithms is significant at a level of 1%, also for the smaller cases. These results are consistent with the averages presented in Table 6.4 and 6.5.

When comparing the ALNS with the previously developed B&P algorithm, we need to perform the comparison on a different time scale. Except for the small cases, the B&P use more than 15 minutes to obtain a first feasible integer solution. However, in situations where the service demand has sufficiently long periods of stationarity, one can spend more than 15 minutes optimizing the service deployment, and therefore it is interesting to see the performance of the ALNS heuristic in comparison with the exact B&P on a longer time scale. We are now only presenting a comparison between the B&P and the ALNS with the LS operators, but the results of our experiments show that the ALNS with the LS operators still produces better solutions than the ALNS without the LS. In the following comparisons, we need to underline that the B&P algorithm is designed

Table 6.6.: P-values of Wilcoxon signed-rank test: comparison of the relative gaps of the ALNS with and without local search operators at different points in time (seconds).

	Cases	300	600	900
Hybrid cloud cases	`H20-H30`	0.067	0.222	0.266
	`H40-H50`	< 0.001	< 0.001	< 0.001
	`H60-H70`	< 0.001	< 0.001	< 0.001
Private cloud cases	`P20-P30`	0.002	0.009	0.002
	`P40-P50`	0.002	0.002	0.002
	`P60-P70`	0.002	0.006	0.002

with the main focus on finding solutions of high quality, not on finding solutions quickly. Therefore, the root node is solved to LP optimality, before a MIP solver is called to optimize over the columns found up to this point, and then branching is performed. To reduce the time to obtain the first integer solution, it would be possible to call a MIP solver before the root node is finished. Nevertheless, this was not considered in the design of the B&P.

Table 6.7 presents the comparison on the hybrid cloud cases with 75% and 90% private cloud coverage for run times up to three hours. For the cases with 50 services or less and with 75% private cloud coverage, the B&P algorithm performs better than the ALNS on a long time scale. However, in all of the cases with 50 services or more, the B&P spends longer than an hour to solve the root node of the B&B tree. For all `H70` cases, the B&P spends over three hours solving the root node, and does not find a solution within the maximum run time. Moreover, it manages to find a solution in only two of the five H60 cases within three hours. With a private cloud coverage of 90%, the right part of Table 6.7 shows that the ALNS performs better than the B&P in all cases with 30 services or more. Still, the B&P spends longer than three hours to solve the root node in all of the `H70` cases. Nevertheless, it finds a solution in one of the `H50` cases within one hour, in two of the `H60` cases within two hours, and another two of the `H60` cases within three hours.

Table 6.8 presents a comparison between the B&P and the ALNS with the LS operators on the private cloud cases. Except for the `P20` cases, the ALNS produces the best solutions after one, two and three hours of run time. For the `P20` cases, the B&P beats the ALNS when given three hours. While none of the solution approaches managed to solve any of the hybrid cloud cases to optimality, the B&P finds and proves the optimal solution in three of the `P20` cases and two of the P30 cases. In comparison, the ALNS manages to find the optimal solution in two of the `P20` cases. Even though the private cloud cases are easier, in terms of lower gaps and more cases solved to optimality, the B&P uses more than an

Table 6.7.: Average relative gap (in %) between best solution found and best bound at different points in time (seconds): comparison of the B&P and the ALNS with local search (LS) operators on the hybrid cloud cases.

	75% private cloud coverage								**90% private cloud coverage**							
	B&P				**ALNS with LS**				**B&P**				**ALNS with LS**			
	1800	3600	7200	10800	1800	3600	7200	10800	1800	3600	7200	10800	1800	3600	7200	10800
H20	2.875	2.007	1.769	1.709	5.066	4.579	3.765	3.380	11.16	8.044	6.775	5.607	14.02	11.93	10.81	10.01
H30	5.349	4.508	3.933	3.854	6.933	5.718	5.030	4.385	20.09	17.74	14.85	12.08	18.28	16.04	13.42	11.63
H40	N/A	6.733	5.902	5.489	7.942	6.832	6.246	5.270	24.24*	23.44	18.39	17.64	20.28	17.85	14.87	13.36
H50	N/A	N/A	7.663	6.738	10.33	9.247	8.444	7.588	N/A	25.74*	22.45	22.39	25.12	21.90	18.79	17.05
H60	N/A	N/A	N/A	8.151*	11.78	10.53	9.156	8.379	N/A	N/A	32.53*	28.04*	27.41	24.17	20.51	18.58
H70	N/A	N/A	N/A	N/A	13.85	12.14	10.96	10.28	N/A	N/A	N/A	N/A	30.33	26.26	22.73	20.87

* The algorithm has found the first integer solution in only some of cases within the specified amount of time

hour to find its first solution in all of the P70 cases. If one compares the results in Table 6.8 with Table 6.5 for the test cases with 40 services or more, one can observe that the ALNS with LS operators produces, on average, solutions with smaller gaps within five minutes than the B&P within three hours. Thus, the ALNS can be said to be much more scalable than the B&P algorithm.

Table 6.8.: Average relative gap (in %) between best solution found and best bound at different points in time (seconds): comparison of the B&P and the ALNS with local search (LS) operators on the private cloud cases.

	B&P				**ALNS with LS**			
	1800	3600	7200	10800	1800	3600	7200	10800
P20	1.831	1.553	0.924	0.620	0.933	0.933	0.933	0.933
P30	3.113	2.522	1.916	1.343	1.362	1.160	1.160	0.977
P40	4.377*	3.958	3.081	2.938	2.054	1.760	1.465	1.465
P50	N/A	5.144*	4.494	4.137	2.482	2.130	1.777	1.653
P60	N/A	5.335	4.546	4.352	2.772	2.479	2.081	1.782
P70	N/A	N/A	5.183	5.183	3.395	2.799	2.546	2.204

* The algorithm has found the first integer solution in only some of cases within the specified amount of time

Similar to the statistical tests performed for the ALNS with and without LS operators, we have performed the Wilcoxon signed-rank test in order to compare the B&P and the ALNS with LS operators on a longer time scale. The null hypothesis is still that the differences between the algorithms are symmetrically distributed around zero. The most interesting comparisons are on the cases where the B&P is able to find an integer solution within a given time limit. However, to make the comparison complete, we have assigned a high cost to the cases where the B&P did not manage to find the first integer solution within the specified time. The tests are performed with the same grouping of the test cases as before, and the results are shown in Table 6.9. With a significance level of 5%, we cannot reject the null hypothesis for the `H20` and `H30` cases at time 1800 and 3600 seconds. For two and three hours of run time, there is a significant difference in favor of the B&P algorithm. For the group `H40-H50`, the difference is significant at 5% and in favor of the ALNS for all run times, except for after 2 hours. Considering the private cloud cases, the difference is significant in most cases, even at a level of 1% for the larger cases. However, after two and three hours of run time for the group `P20-P30`, the difference is not significant.

Table 6.9.: P-values of Wilcoxon signed-rank test: comparison of the relative gaps of the B&P and the ALNS with local search operators at different points in time (seconds).

	Cases	1800	3600	7200	10800
Hybrid cloud cases	`H20-H30`	0.053	0.053	0.012	0.015
	`H40-H50`	< 0.001	< 0.001	0.073	0.014
	`H60-H70`	< 0.001	< 0.001	< 0.001	< 0.001
Private cloud cases	`P20-P30`	0.014	0.023	0.419	1
	`P40-P50`	0.002	0.002	0.006	0.002
	`P60-P70`	0.002	0.002	0.006	0.002

Lastly, to illustrate the evolution of the objective function values in the search process, Figure 6.1 compares the progress of these values of the B&P and the two ALNS versions on the `H40-a` case with 75% private cloud coverage. We can see that the objective function value of the ALNS with LS operators drops quicker than the ALNS without LS operators. This effect is observed in almost all cases, and can be explained by the intensifying effect of the LS operators. Moreover, the figure shows that B&P finds a solution after about 40 minutes, and this solution is, in this case, better than the best solution found by the ALNS. While the objective function value of the ALNS algorithm drops gradually, and in small steps, this value drops only two times for the B&P. The B&P finds its best solutions by regularly solving an IP over all columns found up to certain points in time, and this explains the few and distinct drops in the objective function value. Similarly, Figure 6.2 shows the progress of the cost on the `P50-d` case. We still see that the ALNS with LS operators drops faster in the beginning of the search, compared to the ALNS without LS operators. The ALNS algorithm produces solutions of higher quality than the B&P in this case, even after three hours. Furthermore, we also observe that the objective function value of the ALNS algorithm now drops fewer times compared to the values in Figure 6.1. Since the objective function corresponds to the number of nodes required in the solution, one should expect to obtain larger plateaus in the evolution of the cost and, typically, the time spent on a given level increases as the objective function value approaches the optimal solution.

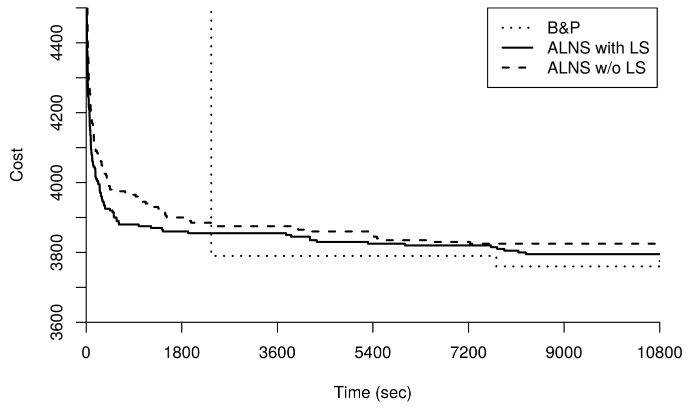

Figure 6.1.: Evolution of objective function value: comparison of the B&P algorithm and the ALNS with and without local search (LS) operators on the `H40-a` case with 75% private cloud coverage

6.6. Conclusions

In this paper, we have presented a novel ALNS for the service deployment problem proposed by Gullhav and Nygreen (2015a). The ALNS implements a LS layer on top of the repair operators, and the different LS operators are selected dynamically based on their past performance. Furthermore, the ALNS includes a MIP-based repair operator, in addition to faster heuristic insertion operators. Since the MIP Insertion is slow, but produces solutions of better quality compared to the fast heuristic insertion operators, it is necessary to take the time consumption into account in the operator scoring mechanism.

The results of our experiments show that the ALNS benefits from the LS operators on the larger hybrid cloud cases and on all private cloud cases. The results also show that the impact of the LS operators are especially prominent in the

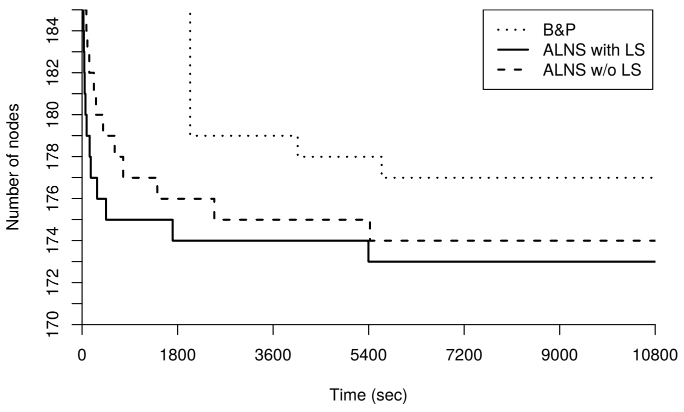

Figure 6.2.: Evolution of objective function value: comparison of the B&P algorithm and the ALNS with and without local search (LS) operators on the `P50-d` case

first minutes of the search, which can be explained by the operators' intensifying effect.

On a longer time scale, we see that the ALNS with LS operators performs significantly better than a previously proposed B&P algorithm on the larger test cases. However, on the smaller cases, the differences between the algorithms are not significant or in favor of the B&P. On the larger private cloud cases, the ALNS with LS operators produces after five minutes of run time as good solutions as the B&P manages to produce after three hours.

Bibliography

Amazon Web Services. Amazon Web Services (AWS) - Cloud Computing Services, 2015. URL `http://aws.amazon.com/`. Last visited 2015/03/04.

D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. Energy-aware autonomic resource allocation in multitier virtualized environments. *IEEE Transactions on Services Computing*, 5(1):2–19, 2012.

A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

A. Beloglazov, R. Buyya, Y. C. Lee, and A. Y. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, 82(2):47–111, 2011.

E. Bin, O. Biran, O. Boni, E. Hadad, E. Kolodner, Y. Moatti, and D. Lorenz. Guaranteeing high availability goals for virtual machine placement. In *2011 31st International Conference on Distributed Computing Systems*, pages 700–709, 2011.

B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174, Berkeley, CA, USA, 2008. USENIX.

T. Distler, R. Kapitza, I. Popov, H. P. Reiser, and W. Schröder-Preikschat. SPARE: Replicas on hold. In *Proceedings of the 18th Network and Distributed System Security Symposium*, Geneva, Switzerland, 2011. The Internet Society.

H. Goudarzi and M. Pedram. Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 324–331, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

A. N. Gullhav and B. Nygreen. Deployment of replicated multi–tier services in cloud data centres. *International Journal of Cloud Computing*, 4(2):130–149, 2015a.

A. N. Gullhav and B. Nygreen. A branch and price approach for deployment of multi-tier software services in clouds. Technical Report IOT-B-15-02, Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology, NO-7491, Trondheim, Norway, 2015b.

A. N. Gullhav, B. Nygreen, and P. E. Heegaard. Approximating the response time distribution of fault-tolerant multi-tier cloud services. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 287–291, Los Alamitos, CA, USA, 2013. IEEE Computer Society.

M. Hollander, D. A. Wolfe, and E. Chicken. *Nonparametric statistical methods*. John Wiley & Sons, Somerset, NJ, USA, 3rd edition, 2013.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In C. A. C. Coello, editor, *Learning and Intelligent Optimization*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer Berlin Heidelberg, 2011.

B. Jennings and R. Stadler. Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, 23(3):567–619, 2015.

R. Jhawar, V. Piuri, and M. Santambrogio. Fault tolerance management in cloud computing: A system-level perspective. *IEEE Systems Journal*, 7(2):288–297, 2013.

W. Kuo and R. Wan. Recent advances in optimal reliability allocation. In G. Levitin, editor, *Computational Intelligence in Reliability Engineering*, volume 39 of *Studies in Computational Intelligence*, pages 1–36. Springer Berlin Heidelberg, 2007.

H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search: Framework and applications. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, pages 363–397. Springer, Boston, 2010.

S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi. Cloud computing — the business perspective. *Decision Support Systems*, 51(1):176 – 189, 2011.

P. Mell and T. Grance. The NIST definition of cloud computing, 2011. NIST SP 800-145.

D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403 – 2435, 2007.

D. Pisinger and S. Ropke. Large neighborhood search. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, pages 399–419. Springer, Boston, 2010.

S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171, 2000.

P. Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. Technical report, Department of Computer Science, University of Strathclyde, Glasgow, Scotland, UK, 1997.

F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86(3):565–594, 1999.